Крис Касперски

Техника защиты компакт-дисков от копирования

+ CD-ROM

- Устройство и организация данных на CD-ROM/CD-R/CD-RW
- Программирование приводов SCSI/IDE на высоком и низком уровнях
- Восстановление поврежденных и поцарапанных дисков
- Уязвимости популярных копировщиков Clone CD и Alcohol 120%
- Ошибки в коммерческих программах защиты CD от копирования
- Разработка собственного копировщика CD

Крис Касперски

Техника защиты компакт-дисков от копирования

Санкт-Петербург

«БХВ-Петербург»

2004

УДК 681.3.06
ББК 32.973
К28

Касперски К.

К28 Техника защиты компакт-дисков от копирования. — СПб.: БХВ-Петербург, 2004. — 464 с.: ил.

ISBN 5-94157-412-6

Рассмотрены устройство и организация данных на дисках CD-ROM/CD-R/CD-RW, секреты профессионального прожига лазерных дисков, а также программирование приводов SCSI/IDE на высоком и низком уровнях. Даны практические советы по восстановлению поврежденных и поцарапанных дисков. Подробно описаны основные механизмы защиты аудиодисков и дисков с данными от копирования и дублирования. Рассмотрены популярные копировщики защищенных дисков (Clone CD, Alcohol 120%) и показано несколько защит, которые предохраняют диски от несанкционированного копирования этими программами. Указаны ошибки, допущенные разработчиками коммерческих пакетов защиты компакт-дисков (StarForce, SecuROM, SafeDisk, Cactus Data Shield, CD-Cops и др.), "благодаря" которым копирование защищенных дисков остается возможным. Показано, как разработать собственный копировщик компакт-дисков.

Для программистов

УДК 681.3.06
ББК 32.973

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.08.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,4.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-412-6

© Касперски К., 2004
© Оформление, издательство "БХВ-Петербург", 2004

Содержание

Предисловие **1**

Введение **3**

Условные обозначения 4
Исторический аспект 5
Мысли о хакерах, защитах и программировании 11

ЧАСТЬ I. АНАТОМИЯ ЛАЗЕРНОГО ДИСКА **13**

Глава 1. Организация CD **15**

Кратко о питах, лендах, EFM-словах, фреймовых кадрах и секторах 16
Каналы подкода 17
Адресация секторов 21
"Сырые" и "сухие" сектора 22
Синхрогруппы, объединяющие биты и DSV 25
Скремблирование 30
F1-, F2- и F3-фреймы, CIRC-кодирование 39
F1-фрейм 39
F2-фрейм 42
F3-фрейм 48
Программная, вводная и выводная области, оглавление диска и область данных 49

Глава 2. Могущество кодов Рида-Соломона или информация, воскресшая из пепла **52**

Корректирующие коды и помехоустойчивое кодирование 54
Идея кодов Рида-Соломона 60
Общее представление 66
Что читать 66

Полиномиальная арифметика и поля Галуа 68
Полиномиальная арифметика 68
Поля Галуа 70
Сложение и вычитание в полях Галуа 72
Умножение в полях Галуа 73
Деление в полях Галуа 78
Простейшие практические реализации 79
Коды Рида-Соломона в практических реализациях 83
Легенда 83
Кодировщик (encoder) 84
Декодер (decoder) 88
Синдромный декодер 89
Полином локатора ошибки 90
Корни полинома 92
Восстановление данных 92
Исходный текст декодера 93
Интерфейс с библиотечкой ElByECC.DLL 100
Подключение библиотеки ElByECC.DLL к своей программе 101
Функция *GenECCAndEDC_Mode1* 101
Функция *CheckSector* 102
Финал 103

ЧАСТЬ II. МЕТОДЫ НИЗКОУРОВНЕВОГО УПРАВЛЕНИЯ ПРИВОДАМИ 107

Глава 3. Практические советы по восстановлению системы в боевых условиях 109

Приложения, недопустимые операции и все, все, все 109
Доктор Ватсон 111
Microsoft Visual Studio Debugger 118
Обитатели "сумеречной зоны", или из "морга в реанимацию" 119
Принудительный выход из функции 120
"Раскрутка" стека 123
Передача управления на функцию обработки сообщений 126
Как подключить дамп памяти 134
Восстановление системы после критического сбоя 145
Подключение дампа памяти 145

Глава 4. Интерфейсы взаимодействия с оборудованием 151

Доступ через CD-ROM-драйвер 152
Доступ через Cooked-Mode (режим блочного чтения) 160
Доступ через SPTI 163

Доступ через ASPI ... 183
Доступ через SCSI-порт ... 195
Доступ через SCSI-мини-порт ... 200
Взаимодействие через порты ввода/вывода ... 210
Доступ через MSCDEX-драйвер ... 221
Взаимодействие через собственный драйвер ... 224
Сводная таблица характеристик различных интерфейсов ... 225

Глава 5. Способы разоблачения защитных механизмов ... 227

"Отжиг" дисков. За, против и немного вокруг ... 229
Блокирование/разблокирование кнопки Eject ... 234
Хакерские секреты. Рецепты "тормозной жидкости" для CD ... 235
Примеры исследования реальных программ ... 237
Alcohol 120% ... 237
Easy CD Creator ... 238
CloneCD ... 238

ЧАСТЬ III. ЗАЩИТЫ ОТ КОПИРОВАНИЯ И СПОСОБЫ ИХ ПРЕОДОЛЕНИЯ ... 239

Глава 6. Механизмы защиты ... 241

Встроенная защита CD-дисков ... 246
Защиты, основанные на нестандартных форматах диска ... 246
Искажение TOC и его последствия ... 246
Некорректный стартовый адрес трека ... 248
Фиктивный трек в настоящем треке ... 275
Фиктивный трек в области данных подлинного трека ... 285
Фиктивный трек в Post-gap подлинного трека ... 291
Фиктивный трек в Pre-gap подлинного трека ... 292
Фиктивный трек в Lead-Out ... 295
Фиктивный трек, совпадающий с подлинным треком ... 303
Искажение нумерации треков ... 305
Некорректный стартовый номер первого трека ... 307
Дважды одинаковый трек ... 310
Некорректный номер последнего трека ... 315
Разрыв в нумерации треков первой сессии ... 317
Разрыв в нумерации треков второй сессии ... 320
Диск, начинающийся не с первого трека ... 324
Диск с нулевым треком ... 326
Трек с нестандартным номером ... 343
Трек с данными, маскирующийся под аудио ... 350
Некорректный Run-out как средство защиты или X-сектор ... 359

Глава 7. Защиты, препятствующие проигрыванию диска в PC CD-ROM 371

Аудио, перекрываемое данными 371
Урезанный Lead-Out 376
Отрицательный стартовый адрес первого аудиотрека 377

Глава 8. Защиты от пофайлового копирования диска (защиты уровня файловой системы) 378

Искажение размеров файлов 379
Шифровка файлов 392

Глава 9. Защиты, основанные на привязке к носителю 400

Нанесение меток vs. Динамическая привязка 401
Защиты, основанные на физических дефектах 402
Защиты, основанные на временных характеристиках чтения 407
Измерение угла между секторами 411
Защиты, основанные на "слабых" секторах 414

Глава 10. Техника восстановления данных с лазерных дисков или практическое знакомство с сессиями 418

Восстановление удаленных файлов с CD-R/CD-RW 418
Получение доступа к удаленным файлам 420
Восстановление целых сессий 425
Ошибки начинающих или то, чего делать не следует 426
Восстановление очищенных CD-RW 426
Как восстановить нечитающийся CD? 433
Общие рекомендации по восстановлению 434
Диск не опознается приводом 436
Диск опознается приводом, но не опознается операционной системой 439
При вставке диска в привод компьютер "зависает" 440
Диск читается с ошибками 440

Описание компакт-диска 443

Предметный указатель 445

Предисловие

Книга "Техника защиты компакт-дисков от копирования" представляет собой практическое руководство по защите лазерных дисков от несанкционированного копирования, ориентированное на самый широкий спектр читательской аудитории: квалифицированных пользователей, прикладных и системных программистов.

Для создания стойкой, дешевой и надежной защиты вовсе не обязательно иметь дорогостоящее спецоборудование или быть экспертом по безопасности. Обыкновенный бытовой рекордер и пара вечеров свободного времени — вот и все, что для этого надо! Окунитесь в подробное, но вместе с тем увлекательное описание архитектуры лазерных дисков и принципов хранения данных на оптических носителях. Книга, которую вы держите в руках, дает исчерпывающее представление о структуре CD и раскрывает множество секретов, известных только профессионалам высочайшего класса (да и то не всем), излагая материал в доступной форме без высшей математики и практически без ассемблера. И это — ее главная уникальность!

Прочитав эту книгу, вы узнаете: как исказить формат диска так, чтобы он нормально читался (воспроизводился) на подавляющем большинстве приводов CD-ROM, но не копировался бы практически ни одним копировщиком; как привязаться к физической структуре диска так, чтобы копировщики не могли ни воссоздать его, ни сымитировать; какими физико-техническими ограничениями обладают бытовые рекордеры и как использовать это обстоятельство в своих целях.

Вы также научитесь управлять читающими/пишущими приводами на низком уровне, получив максимально полный контроль над лазерным диском, который только позволяет осуществить данная модель привода. При прочих равных условиях диск, защищенный на более высокотехнологичном приводе, не может быть скопирован на всех остальных. Книга подробно рассказывает, чем отличается один привод от другого и на какие его характеристики следует обращать внимание в первую очередь.

В книге подробно рассматриваются, можно даже сказать, разбираются по "косточкам", практически все существующие на сегодняшний день коммерческие защитные пакеты (StarForce, SecuROM, SafeDisk, Cactus Data Shield, CD-Cops и т. д.) с указанием ошибок, допущенных при их реализации, "благодаря" которым копирование защищенных дисков остается все-таки возможным. Защитные механизмы, предлагаемые автором, учитывают горький опыт всех его последователей и не копируются ни одним из существующих на сегодняшний день копировщиков.

Кстати о копировщиках. Здесь вы найдете подробное описание наиболее популярных на сегодняшний день копировщиков защищенных дисков: CloneCD и Alcohol 120%, которые по утверждению их создателей "при правильном сочетании читающего и пишущего приводов могут скопировать любую защиту". Автор убедительно показывает, что это не так, и демонстрирует ряд защит, которые не копируются ни CloneCD, ни Alcohol 120%.

Наконец, книга рассказывает о том, как самостоятельно создать копировщик защищенных дисков, без которого тиражирование защищаемых вами дисков оказалось бы весьма нетривиальной задачей.

Введение

Актуальность защиты лазерных дисков сегодня как никогда велика. Широкое распространение бытовых рекордеров позволило пользователям тиражировать диски чуть ли не в промышленных масштабах, ну или, по крайней мере, львиную долю дисков не покупать, а позаимствовать у приятелей. В то же время многие shareware-программисты распространяют свои продукты на CD-R-дисках по почте, что значительно усложняет задачу хакеров (т. к. если программы нет в открытом доступе, то как ее прикажете ломать?).

В итоге, пользователи интересуются, как ломать защищенные диски, а разработчики — как защитить эти диски так, чтобы их не взломали. Данная книга удовлетворяет интересы обоих групп. Она рассказывает о том, как взламываются практически все, существующие на сегодняшний день, защитные пакеты, и предлагает ряд новых, принципиально не взламываемых защитных механизмов.

Эта книга содержит большое количество уникального, ранее нигде не публиковавшегося материала. Она дает читателю исчерпывающее представление о структуре CD и раскрывает множество секретов, известных только профессионалам, причем материал изложен в доступной форме без вышей математики и практически без ассемблера.

Прочитав эту книгу, читатель научится создавать действительно принципиально некопируемые диски и эта принципиальность будет гарантироваться аппаратными ограничениями современных CD-R/CD-RW-рекордеров. Помимо того, читатель узнает, как избежать конфликтов с нестандартным оборудованием, из-за которых защита отказывается работать у некоторых пользователей или, что еще хуже, приводит к порче их оборудования.

Книга ориентирована на широкой спектр читательской аудитории. По минимуму — никакой специальной подготовки от читателя и не требуется, он даже может не знать, из каких секторов состоит CD-ROM (99% программистов этого, кстати, и не знают). Вся информация, необходимая для осмысленной

работы с CD-ROM, изложена непосредственно в самой книге и отсылки к посторонним источникам минимальны. Читатель не обязательно должен уметь программировать, т. к. все необходимые утилиты для анализа/защиты/взлома лазерных дисков уже прилагаются к книге. Наконец, читатель может воспользоваться автоматическими копировщиками, разработанными автором, которые все сделают за него. Так что книгу стоит покупать уже ради одного содержимого прилагаемого к ней CD.

По максимуму — читатель должен знать математику в объеме вузовской программы, "уметь держать в руках" дизассемблер и "свободно говорить" на Си и ассемблере. Чтение настоящей книги, конечно, не сделает его "богом", но: безграничную власть над лазерными дисками он все-таки получит и сможет вытворять с ними то, что другим и не снилось.

Условные обозначения

Для предотвращения путаницы и одновременно с этим для избежания многословия, в книге вводится ряд условных обозначений, расшифровкой которых мы сейчас и займемся. В частности, условные обозначения CD-приводов:

- **NEC** — _NEC CD-RW NR-9100A; firmware version 1.4;
- **ASUS** — ASUS CD-S500/A; firmware version 1.4;
- **TEAC** — TEAC CD-W552E; firmware version 1.09;
- **PHILIPS** — PHILIPS CDRW2412A; firmware version 1.5.

Кроме того, представлены следующие обозначения копировщиков:

- **Alcohol 120%** — отличный копировщик защищенных дисков, условно-бесплатную версию которого можно получить с сайта **http://www.alcohol-soft.com/**. Автоматически ломает более половины всех существующих типов защит от копирования и позволяет динамически монтировать образы защищенных дисков на виртуальный привод CD-ROM, что очень удобно для экспериментов. К сожалению, монтированию подлежат лишь "правильные" образы, коими большинство защищенных дисков отнюдь не являются.
- **CloneCD** — хороший копировщик защищенных дисков, условно-бесплатную версию которого можно скачать по следующему адресу: **http://www.elby.ch/**. С копированием защищенных дисков в полностью автоматическом режиме CloneCD справляется скорее плохо, чем хорошо, однако после ручного шаманства с настройками и непосредственно самим образом защищенного диска он может скопировать добрую половину существующих типов защит. Утверждение о том, что CloneCD "берет" практически все существующие защиты от копирования — ложное и невероятно далекое от действительности.

Исторический аспект

Первые попытки защиты лазерных дисков датируются началом девяностых годов XX века. Пишущих приводов в то время еще не существовало и в основном приходилось бороться с несанкционированным копированием содержимого CD на жесткий диск. А как же пираты? — спросите вы. Да, действительно, уровень пиратства в России всегда был и остается традиционно велик, но пытаться остановить его программными средствами защиты по меньшей мере наивно. Тот, кто копирует диски в промышленном масштабе, всегда держит при себе пару-тройку опытных хакеров, снимающих такие защиты без труда. Интеллектуальный потенциал "отдела по снятию защит" в пиратских конторах практически неограничен, — здесь работают лучшие из лучших (когда-то, до появления соответствующих законов, автор этой книги в таком "отряде" тоже состоял), и финансовый фактор тут, кстати говоря, вторичен. Платили немного, а "вкалывать" приходилось вовсю, но в этом-то весь интерес и был! Где еще вы могли познакомиться с таким количеством разнообразных защит и приобрести навыки по их ликвидации?

Впрочем, насчет количества я немного загнул. Все многообразие защитных механизмов тех дней сводилось к двум основным типам: LaserLock и кодовое колесо (подробности далее). С появлением пишущих приводов актуальность защит от копирования значительно возросла и они поперли, как грибы после дождя. К началу 2003 года на рынке насчитывалось более полусотни разнообразных методик защиты, большая часть из которых выдавалась за ноу-хау, разработавшей их фирмы. Однако стоило пропустить защиту через дизассемблер, как вас отхватывало щемящее чувство ностальгии по тем далеким и, казалось бы, безвозвратно растворившимся в песке истории временам, когда программное обеспечение поставлялось на дискетах и каждая вторая из них оказывалась защищенной. Современный лазерный диск, конечно, не похож на дискеты десятилетней давности, но методики защиты тех и других по сути своей общие!

В современных защитных механизмах используются следующие методики:

- ❒ использование нестандартной разметки;
- ❒ внедрение ключевых меток;
- ❒ привязка к поверхности;
- ❒ "слабые" сектора.

Познакомимся со всем этим семейством поподробнее.

Нестандартная разметка диска в общем случае сводится к умышленному внесению тех или иных искажений, препятствующих нормальной обработке информации. Например, если длину каждого из защищенных файлов искусственно увеличить до ~666 Гбайт, просто скорректировав поле длины, то при попытке копирования таких файлов на винчестер произойдет грандиозный

"облом-с". В то же самое время защита, точно знающая от сих и до сих каждый файл, можно читать, будет работать с ними без особых проблем. Разумеется, такой защитный механизм элементарно взламывается копированием диска на посекторном уровне, однако для этого копировщик должен знать, какое именно количество секторов содержится на диске. Разработчику защиты ничего не стоит исказить служебные структуры диска так, чтобы тот либо представлялся совсем пустым, либо, напротив, разросся до неимоверных размеров. Копировщики, тупо читающие оглавление диска и свято верящие каждому байту служебных данных, просто "заглючат" по полной программе. Те же, кто "поумнее", сумеют определить истинный размер диска по косвенным признакам, двигая оптической головкой до тех пор, пока она еще двигается, а сектора, пролетающие над ней, — читаются. Допустим, защита решит схитрить и в непосредственной близости от конца диска "выроет яму" из множества сбойных секторов. "Ага! — подумают некоторые копировщики, после того как свалятся в нее. — Мы достигли конца!" "А вот и ничего подобного!" — воскликнут другие. Те, что тщательно анализируют чувственную информацию, возвращенную приводом, который-то наверняка знает, в чем причина неудачного чтения — то ли это диск кончился, то ли просто плохой сектор попался.

Другие защиты поступают еще хуже, нагло и самоуверенно записывая оригинальный диск с неустранимыми ошибками (неустанными — значит, не исправляемыми специальными корректирующими кодами, размещенными на CD). Для аудиодисков это означает, что проигрывание последнего будет сопровождаться ожесточенными щелчками. Точнее, должно было бы сопровождаться, но на практике этого не происходит, поскольку разработчики аудиопроигрывателей предусмотрели специальный фильтр, отбрасывающий заведомо искаженные данные и при необходимости прибегающий к *интерполяции* (когда текущая точка отсчета строится на основе усредненных значений предыдущей и последующей точек). Разумеется, это несколько ухудшает качество воспроизведения, но медиамагнатам на это наплевать, да и ухудшение это не такое уж и значительное. С цифровым воспроизведением все обстоит иначе. Ранние версии Стандарта предписывали приводу сообщать лишь о факте возникновения одной или нескольких неустранимых ошибок, но не предусматривали никаких механизмов "маркировки" сбойных байт. Ну считал привод 2352 байта данных, ну убедился, что добрая сотня из них искажена. Что ему дальше-то делать? Интерполировать? Кого и с чем?! Вручную анализировать сигнал и искать "выхлесты"? Слишком сложно, да и качество "восстановленного" звука будет все равно не то. Можно, правда, отважиться "сграбить" аудиопоток с цифрового аудиовыхода, но подавляющее большинство дешевых звуковых карт его не поддерживает, а если и поддерживает, то так "криво", что лучше бы этого вообще не делали. Короче, над хакерами начали сгущаться мрачные тучи без следов присутствия лучика солнца. Но все изменилось, когда производители "выбросили" на рынок приводы, умеющие не только тупо сигнализировать об ошибке чтения,

но и сообщающие позицию сбойных байт в секторе (прямо как в анекдоте: "ты не мудри, ты пальцем покажи!"). Теперь полноценная интерполяция стала возможна и на интерфейсном уровне! Немедленно появились и программы-грабители, использующие новые возможности.

Впрочем, мы сильно забежали вперед. В плане возвращения к анналам перенесемся в те далекие времена, когда никаких оптических приводов еще и в проекте не существовало и все программное обеспечение распространялось исключительно на дискетах, стремительно утекающих как "налево", так и "направо" (собственно говоря, "copyright" именно так и переводится: "скопировано правильно"). Тогда все кому не лень активно "царапали" дискеты всеми подручными предметами: кто побогаче — прожигал магнитное покрытие лазером, кто победнее — орудовал ржавым гвоздем. Защите оставалось лишь проверить присутствие дефекта поверхности в строго определенном месте. Скопировать такой диск без спецоборудования было практически нереально, т. к. даже Левша не смог бы перенести царапины оригинального диска на то же самое место. Правда, хакеры, знающие порты контроллера как свои пять пальцев, быстро сообразили, что если исказить контрольную сумму ключевых секторов, то, несмотря на физическую целостность поверхности, диск будет читаться с ошибкой! Так вот, лазерные диски защищаются тем же самым способом! И абсолютно тем же самым способом они "ломаются"! Производитель может "нафаршировать" диск сбойными секторами, словно рождественского гуся, и при каждом запуске защищенного программного обеспечения проверять их присутствие. Это порождает следующие проблемы: во-первых, далеко не всякий копировщик согласится копировать дефективный диск, а если даже и согласится, то ждать завершения процесса копирования придется ну очень долго (все мы знаем, с какой скоростью читаются дефективные сектора). Но полученная копия окажется все равно неработоспособной, поскольку на ней-то заданных дефектов уже не окажется — а это во-вторых.

Бездумные хакеры просто искажают контрольную сумму сектора, заставляя привод возвращать ошибку (естественно, пишущий привод должен позволять записать сектора с ошибкой контрольной суммы, на что согласится далеко не каждый). Однако это не решает проблемы — ведь "гнутый" сектор читается мгновенно и защита, если она не совсем дура, может сообразить, что здесь, что-то не так! Или, как вариант, она может провести длинное чтение сектора, и тогда сектор с искаженной контрольной суммой начнет читаться!

Как поступают умные хакеры? Ну, это так сразу и не объяснишь. Упрощенно говоря, формат лазерного диска таков, что высокочастотный сигнал, возникающий при чтении последовательности *питов* (pits) и *лендов* (lands), пролетающих над оптической головкой, не имеет опорного уровня, и чтобы привод мог определить, где здесь минус, а где плюс, количество лендов должно быть приблизительно равно количеству питов. О питах и лендах см. далее *главу 1.* Если какой-то участок сектора будет содержать одни питы,

то он окажется катастрофически темным и автоматический усилитель сигнала привода попытается увеличить мощность лазерного луча, ошибочно полагая, что с диском или оптикой не все в порядке. Но ведь тогда часть питов превратится в ленды и привод "обломается" по всем статьям. Сначала он свалится в рекалибровку, поерзает оптической головкой и лишь затем печально констатирует тот факт, что данный сектор не читается. С точки зрения защиты такой сектор будет выглядеть как глубоко дефектный, хотя на физическом уровне поверхность носителя останется и не повреждена.

Теперь самое главное: поскольку привод должен уметь записывать любые мыслимые и немыслимые данные, для преодоления подобных неблагоприятных ситуаций, разработчики были вынуждены предусмотреть специальный механизм их обхода. Существует несколько возможных способов кодирования записываемых на диск данных и привод должен выбрать наиболее благоприятные из них. К счастью (или несчастью), не все приводы столь щепетильны. И поскольку вероятность непредумышленного возникновения неблагоприятных последовательностей исчезающе мала, некоторые (между прочим, достаточно многие) приводы кодируют данные одним-единственным наперед заданным способом. А значит, существует возможность сымитировать сбойные сектора, практически ничем не отличающиеся от настоящих.

Ага! Сказали разработчики защит! Да это же целый клад! Смотрите — если подобрать специальную неблагоприятную последовательность байт, то для ее корректной записи подойдет далеко не всякий привод! При копировании такого диска на обычном приводе оригинал будет изумительно читаться, но копия обнаружит большое количество "бэдов", и скопированный диск запускаться ни за что не будет. Сектора с неблагоприятными последовательностями получили название *"слабых"* (weak), и для их копирования необходимы весьма высокотехнологичные и "навороченные" приводы от "крутых" брэнд-неймов (brand-name). А если такого привода у нас нет и он нам не по карману, тогда что — "кранты", да? А вот и нет! Если только защита не делает дополнительных поползновений, копировщик может рассчитать корректирующие коды для истинной неблагоприятной последовательности, а затем слегка выправить ее и записать на диск. На физическом уровне такой сектор будет читаться без каких-либо проблем, ну а на логическом — привод самостоятельно восстановит его по избыточным кодам в нормальный вид. Правда, если защита прочитает сектор в сыром виде, то она сразу же распознает подлог, так что таким способом копируются далеко не все диски.

Чтобы понять суть следующего защитного механизма, нам тоже придется обратиться к дискетам. Как известно, поверхность дискеты физическим образом делится на концентрические кольца, именуемые *дорожками*, а дорожки в свою очередь делятся на *сектора*. При перемещении головки от последнего сектора одной дорожки к первому сектору следующей дорожки, в силу вращения дискеты мотором, этот сектор успевает "пролететь", и дисководу приходится ждать целый оборот, чтобы дождаться "свидания".

Парни, денно и нощно сидевшие в насквозь прокуренных вычислительных центрах, уже тогда додумались, что если провернуть сектора каждой последующей дорожки, а в случае с винчестером — *цилиндра*, на некоторое расстояние, то скорость последовательного чтения секторов существенно возрастет, поскольку теперь нужный сектор сразу же окажется под головкой. С другой стороны, проворачивая сектора различных цилиндров на различный угол, мы добьемся определенных колебаний скорости обмена, по которым оригинальный диск может быть легко отличен от копии, таких колебаний не содержащей.

Теперь перейдем к лазерным дискам. Никаких цилиндров здесь и в помине нет и последовательность секторов скручена в тугую спираль. Позиционирование на сектора соседних витков дорожки осуществляется путем отклонения лазерной головки магнитной системой (т. е. происходит практически мгновенно), а позиционирование на удаленные сектора вовлекает в движение механизм перемещения головки по специальным "ползункам" — что требует значительного времени. Зная скорость вращения диска и измерив время позиционирования на сектора соседних витков дорожки, мы сможем найти угол между ним, напрямую зависящий от степени закрутки спирали. Различные партии CD-R/CD-RW-дисков обладают различной структурой спирали, и, что самое неприятное, эта структура закладывается непосредственно самим производителем — т. е. диски поступают в продажу с предварительно выполненной разметкой, необходимой для ориентации записывающего привода на "местности". Скопировать защищенный таким образом диск нереально, и приходится прибегать к его эмуляции. Копировщик должен тщательно измерять углы между различными секторами и воссоздать исходную структуру спирали. Процесс сканирования диска занимает чудовищное количество времени (порой несколько суток), но результат того стоит.

Диск также может иметь катастрофически нестандартный формат, — например сектора переменной длины, в результате чего одни из них будут читаться быстрее, другие медленнее. Поскольку всякое изменение длины секторов немедленно отражается на структуре спиральной дорожки, копировщику приходится иметь дело с двумя неизвестными — неизвестным углом спиральной закрутки и неизвестной длиной секторов. С математической точки зрения это уравнение имеет множество возможных решений, но только одно из них правильное. Копировщик может (и должен!) представить несколько вариантов копий, чтобы мы могли самостоятельно решить, какая из них "ломает" защиту, а какая нет. К сожалению, ни один из всех известных мне копировщиков этого не делает.

Впрочем, длинные сектора представляют собой вполне самостоятельную сущность, и некоторые диски используют для своей защиты только их одних. Плохо то, что ни один из всех представленных на рынке пишущих приводов не позволяет управлять длиной записываемых секторов по нашему усмотрению. Правда, одна зацепка все же есть — пусть мы не можем увеличить

длину сектора, но мы в состоянии создать два сектора с идентичными заголовками — привод, успешно прочитав первый из двух секторов, второй просто проигнорирует, тем не менее, видимая длина сектора возрастет вдвое. Минус этой технологии состоит в том, что мы можем увеличить длину секторов лишь на величину, кратную двум, да и то не на всех приводах. Некоторые из них писать спаренные сектора (они, кстати, называются *twin-sectors*) просто отказываются.

Теперь перейдем к ключевым меткам. Помимо пользовательской области сектора, которую исправно копируют практически все копировщики, на лазерных дисках существует множество мест, на которые "не ступала нога человека". Прежде всего это *каналы подкода.* Всего их восемь. Один хранит сервоинформацию, по которой лазерная головка "ориентируется на местности", другой — информацию о паузах, остальные шесть каналов свободны и нормальные копировщики их не копируют, да и не всякие пишущие приводы дают возможность их записывать. Вот сюда ключевые метки защиты и внедряют!

Кстати говоря, каналы подкода хранятся независимо от канала основных данных, и прямого соответствия между ними нет. При чтении канала подхода сектора X, привод может вернуть субканальные данные любого из соседних секторов по своему усмотрению — это раз. А теперь два — большинство приводов обладает крайне плохим постоянством, и при последовательном чтении субканальных данных секторов X, Y и Z нам могут возвратиться, например, данные X, X, X, или Y, Z, X, или Y, Z, Z, или любая другая комбинация последних. Допустим, канал подкода одного из секторов содержит ключевую метку. Допустим, мы пытаемся ее прочитать. Но прочитаем ли? А вот это как раз и не факт! Если сервоинформация окажется слегка искажена, мы вообще не сможем разобраться, субканальные данные каких именно секторов мы прочитали и входит ли наш сектор в их перечень или нет. Единственный выход — воспользоваться качественным читающим приводом, обладающим хорошим постоянством чтения субканальных данных.

И последнее. Записываемые и перезаписываемые диски по ряду характеристик значительно отличаются от штампованных CD-ROM. ATIP[1] представлять, думаю, нет необходимости? Еще существует такая вещь, как TDB (Track Descriptor Block — блок описания трека), среди прочей информации сообщающий мощность лазера и иже с ней. На CD-ROM-дисках ничего подобного, разумеется, нет. Непосредственно подделать природу CD-ROM-диска невозможно, но существует множество утилит, перехватывающих все обращения к приводу и возвращающих то, что нужно, а не то, что есть на самом деле.

На этом нашу краткую экскурсию "по зоопарку защитных механизмов" можно считать законченным. Затем, по мере углубления в книгу, каждый из этих экспонатов будет рассмотрен во всех подробностях.

---

[1] ATIP (Absolute Time In Pre-Groove) — информация о реальном производителе CD-R/RW-носителя, максимальной разрешенной скорости записи и максимальной емкости диска. — *Ред.*

...обход защиты от копирования совсем не то же самое, что нарушение авторских прав! Законы многих стран (в том числе и Российской Федерации) явным образом разрешают создание резервных копий лицензионного носителя. В то же самое время, нет такого закона, который бы запрещал "взлом" легально приобретенного экземпляра программы. Лицензионное соглашение вправе запрещать что угодно, однако статуса закона оно не имеет. Нарушая лицензионное соглашение, вы автоматически разрываете договор с продавцом программы, а значит, лишаетесь всех обещанных им льгот и гарантий. Приблизительно то же самое происходит, когда вы путем замыкания таких-то ножек процессора разблокируете его тактовую частоту. Посадить вас не посадят, но и сожженный процессор (если он вдруг сгорит) вам не обменяют. С другой стороны, распространение взломанных программ уже попадает под статью, и потому лучше не рисковать.

Мысли о хакерах, защитах и программировании

Взломщики и защитники информации не только враги, но и коллеги. Если предположить, что хакеры паразитируют на программистах (пользуясь их неумением строить по-настоящему качественные защитные механизмы), то тогда с неизбежностью придется признать, что программисты паразитируют на пользователях, пользуясь их неумением программировать!

Хакерство и программирование действительно очень тесно переплетены. Создание качественных и надежных защитных механизмов требует навыков низкоуровневой работы с операционной системой, драйверами и оборудованием; знаний архитектуры современных процессоров и учета особенностей кодогенерации конкретных компиляторов, помноженных на "биологию" используемых библиотек. На этом уровне программирования грань между собственно самим программированием и хакерством становится настолько зыбкой и неустойчивой, что я не рисковал бы ее провести.

Начнем с того, что всякая защита, равно как и любой другой компонент программного обеспечения, требует тщательного и всестороннего тестирования на предмет выяснения ее работоспособности. Под "работоспособностью" в данном контексте понимается способность защиты противостоять квалифицированным пользователям, вооруженным хакерским арсеналом (копировщиками защищенных дисков, эмуляторами виртуальных приводов, оконными шпионами и шпионами сообщений, файловыми мониторами и мониторами реестра). Качество защиты определяется отнюдь не ее стойкостью, но *соотношением* трудоемкости реализации защиты к трудоемкости ее взлома. В конечном счете, взломать можно любую защиту — это только вопрос времени, денег, квалификации взломщика и усилий, но грамотно реализованная защита не должна оставлять легких путей для своего взлома.

Конкретный пример. Защита, привязывающая к сбойным секторам (которые действительно уникальны для каждого носителя), бесполезна, если не способна распознать их грубую эмуляцию некорректно заполненными полями EDC/ECC[2]. Еще более конкретный пример. Привязка к геометрии спиральной дорожки лазерного диска, даже будучи реализованной без ошибок, обходится путем создания виртуального CD-ROM-привода, имитирующего все особенности структуры оригинального диска. Для этого даже не нужно быть хакером, — достаточно запустить копировщик Alcohol 120%, ломающий такие защиты автоматически.

Ошибки проектирования защитных механизмов очень дорого обходятся их разработчикам, но гарантированно застраховаться от подобных просчетов невозможно. Попытка применения "научных" подходов к защите программного обеспечения — чистейшей воды фарс и бессмыслица. Хакеры смеются над академическими разработками в стиле "расчет траектории сферического коня в вакууме", и практически любая такая защита "снимается" за 15 минут без напряжения извилин. Вот грубый, но наглядный пример. Проектирование оборонной системы военной крепости без учета существования летательных средств позволяет захватить эту самую крепость чуть ли не на простом "кукурузнике" (MS WDB — "кукурузник"), не говоря уже об "истребителях" (отладчик Soft-Ice — "истребитель", а дизассемблер IDA Pro — еще и "бомбардировщик").

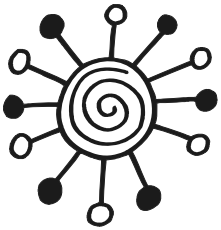
Для разработки защитных механизмов следует иметь хотя бы общее представление о методах работы и техническом арсенале противника, а еще лучше — владеть этим арсеналом не хуже противника (то есть владеть им в совершенстве). Наличие боевого опыта (реально взломанных программ) очень и очень желательно, — пребывание в "шкуре" взломщика позволяет досконально изучить тактику и стратегию наступательной стороны, давая тем самым возможность оптимальным образом сбалансировать оборону. Попросту говоря, определить и усилить направления наиболее вероятного вторжения хакеров, сосредоточив здесь максимум своих интеллектуальных сил. А это значит, что разработчик защиты должен глубоко проникнуться психологией хакеров, настолько глубоко, чтобы начать мыслить, как хакер.

Таким образом, владение технологией защиты информации предполагает владение технологией взлома. Не зная того, как ломаются защиты, не зная их слабых сторон, не зная арсенала хакеров — невозможно создать стойкую, дешевую и, главное, простую в реализации защиту. Книги, рассматривающие вопросы безопасности исключительно со стороны защиты, грешат тем же, что и конструкторы запоминающих устройств, работающих только на запись, — ни то, ни другое не имеет никакого практического применения.

---

[2] EDC/ECC (Error Detection Code/Error Correction Code — коды обнаружения и исправления ошибок. — *Ред.*

[3] MS WDB — это Microsoft Windows Debugger.



Часть I

АНАТОМИЯ ЛАЗЕРНОГО ДИСКА

Глава 1. Организация CD

Глава 2. Могущество кодов Рида-Соломона или информация, воскресшая из пепла

Глава 1

Организация CD

В этой, преимущественно теоретической, главе читатель знакомится с организацией CD и принципами оптической записи, без знания которых осмысленная работа с защитными механизмами попросту невозможна.

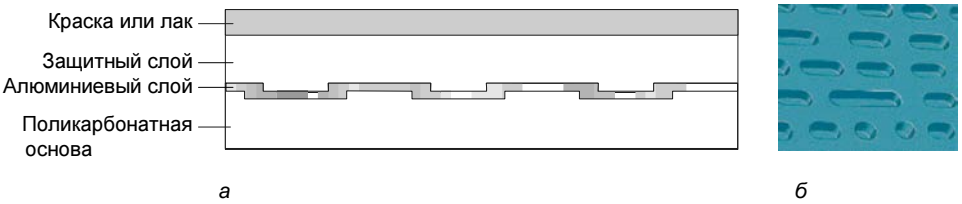


Рис. 1.1. Лазерный диск в разрезе (*а*) и увеличенные питы (*б*)

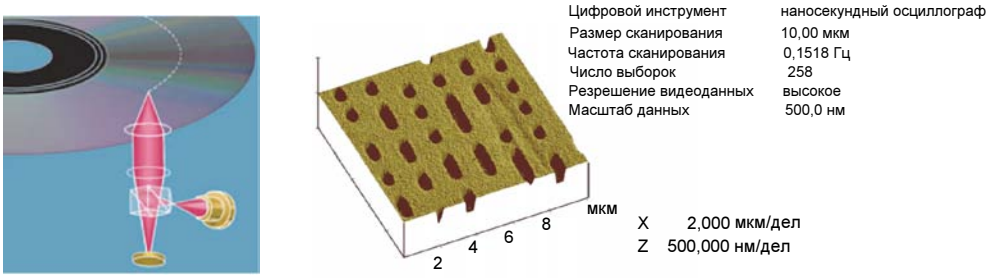


Рис. 1.2. Лазерный диск это все равно что грампластинка

Физически лазерный диск представляет собой пластинку из поликарбоната с нанесенным поверх нее *отражающим* алюминиевым (реже — золотым) *слоем*, защищенным от механических повреждений специальным *защитным слоем* (рис. 1.1). Отражающий слой содержит в себе цепочку углублений или

иначе *питов* (*pits*) и возвышенностей, иначе *лендов* (*lands*), свернутую в спираль, — примерно такую же, как на грампластинке (рис. 1.2), но намотанную в обратном порядке: от центра диска к его краям (фактически лазерный диск является устройством последовательного доступа с ускоренной перемоткой).

Кратко о питах, лендах, EFM-словах, фреймовых кадрах и секторах

Вопреки распространенному заблуждению питы (pits) и ленды (lands) отнюдь не соответствуют двоичному нулю и единице непосредственно. Кодирование информации на CD устроено значительно хитрее и умнее! *Единица представляется переходом от пита к ленду или наоборот, а ноль — отсутствием переходов на данном промежутке* (рис. 1.3). Причем между двумя соседними единицами должно быть расположено не мене двух, но и не более десяти нулей. Ограничение снизу обусловлено технологическими трудностями штамповки, а ограничение сверху — нестабильностью скорости вращения диска. Действительно, пусть стабильность вращения составляет 3%, тогда при считывании последовательности из десяти нулей мы получаем погрешность /3 пита/ленда, что не вызывает никаких проблем. Но уже при чтении пятнадцати нулей погрешность возрастает до половины пита/ленда и приводу остается лишь гадать: в большую или меньшую строну ее следует округлять.

Четырнадцать бит образуют *EFM-слово*, которое по специальной таблице перекодируется в "нормальный" 8-битный байт (собственно ***EFM*** так и расшифровывается: ***E**ight to **F**ourteen **M**odulation — модуляция восемь на четырнадцать*). Между двумя EFM-словами располагаются три *объединяющих бита* (merging bits), которые, во-первых, служат для разрешения конфликтных ситуаций кодирования (например, за одним EMF-словом, оканчивающимся на единицу, следует другое EFM-слово, начинающееся с единицы), а во-вторых, препятствуют появлению ложных *синхрогрупп* (о чем будет рассказано позже).

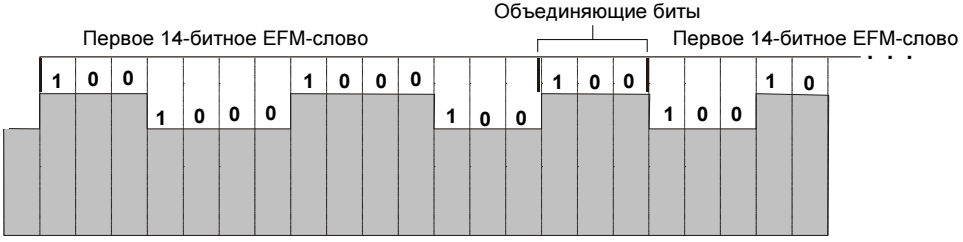


Рис. 1.3. Принцип записи на CD

Группа из 36 байт образует *фреймовый кадр* (*F1 frame*) (рис. 1.4), который состоит из предшествующей ему *синхрогруппы*, байта *субкода* и двух 12-байтных групп *данных*, снабженных 4-байтными полями *контрольных сумм* (или сокращенно CRC — Cyclical Redundancy Check — контроль с помощью циклического избыточного кода).

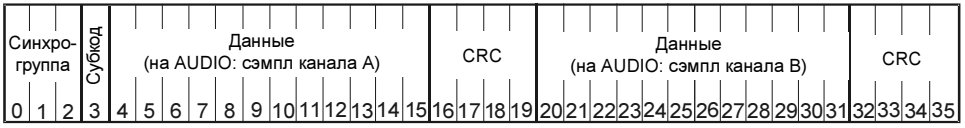


Рис. 1.4. Устройство фреймового кадра

Кадры объединяются в *сектора*, также называемые *блоками*. Каждый сектор состоит из 98 хаотично перемешанных кадров (перемешивание позволяет уменьшить влияние дефектов носителя, поскольку полезная информация как бы "размазывается" вдоль дорожки), причем первые 16-байт всякого сектора занимает специальный *заголовок* (*header*), состоящий из: 12-байтного *поля синхронизации*, 3-байтного *поля адреса* и 1-байтного *поля режима* (рис. 1.5).

Рис. 1.5. Устройство заголовка сектора

Значимость сектора заключается в том, что это наименьший раздел диска, который CD-привод может считать в "сыром" виде. Причем эта "сырость" на ощупь довольно суха. Никакие приводы не позволяют получить содержимое данных кадра как они есть, а принудительно восстанавливают их на аппаратном уровне, используя для этой цели четырехбайтовые поля CRC. Заметим, что отсутствие доступа к действительно "сырым" байтам приводит к невозможности получения побитовой копии диска, а значит, у защитного механизма существует принципиальная возможность отличить, где дубликат, а где оригинал!

Каналы подкода

Среди прочей служебный информации во фреймовый кадр входит один байт *субкода* (*sub-channel byte*), также иногда называемый *байтом субканала* или *управляющим байтом* (рис. 1.6).

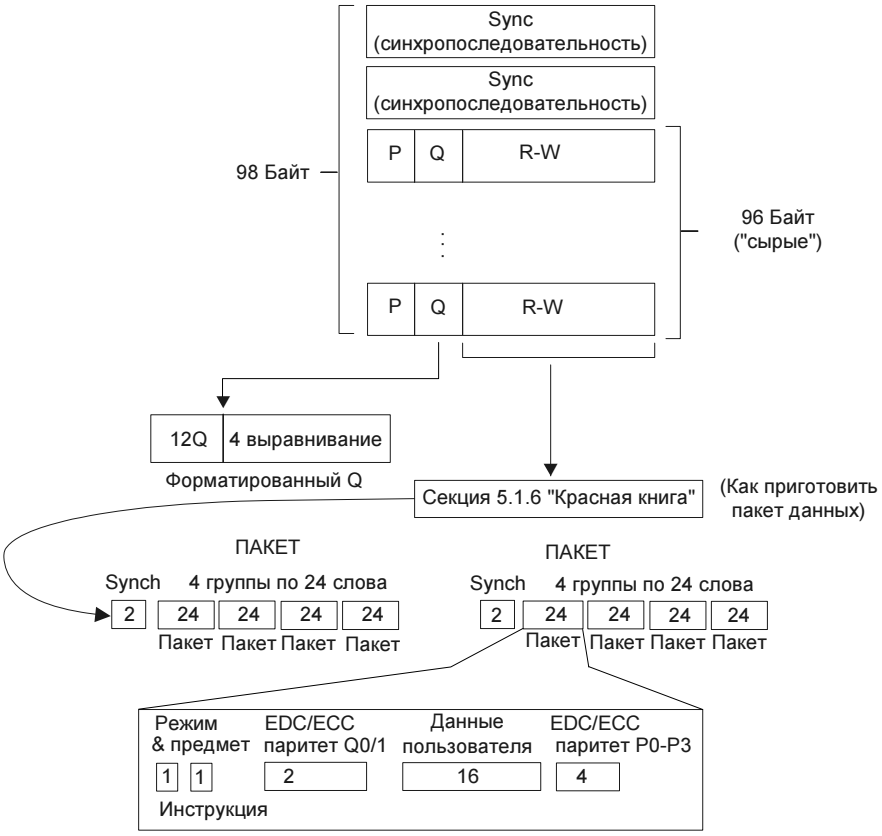


Рис. 1.6. Организация субканальных данных

Субканальные данные полностью изолированы от содержимого сектора и в некотором роде ведут себя точно так же, как и множественные потоки данных в файловой системе NTFS (читайте *"Основы Windows NT и NTFS"* Хелен Кастер). Все это наглядно иллюстрирует рис. 1.7.

Каждый из восьми битов, слагающих байт субкода, обозначается заглавной латинской буковой P, Q, R, S, T, U, V и W соответственно. Одноименные биты субканальных байтов всех фреймов объединяются в так называемые *каналы субкода*. Каналы состоят из *секций*, каждая из которых образуется путем объединения субканальных данных из 98 фреймов, что соответствует одному сектору (см. рис. 1.6). Однако границы секций и секторов могут и не совпадать, поэтому для гарантированного извлечения одной секции с диска мы должны прочесть два сектора. Первые два байта секции задействованы для синхронизации, а 96 отданы под действительные данные. Путем несложных расчетов можно вычислить, что на каждый канал приходится ровно по 16 байт "сырых", еще не обработанных данных.

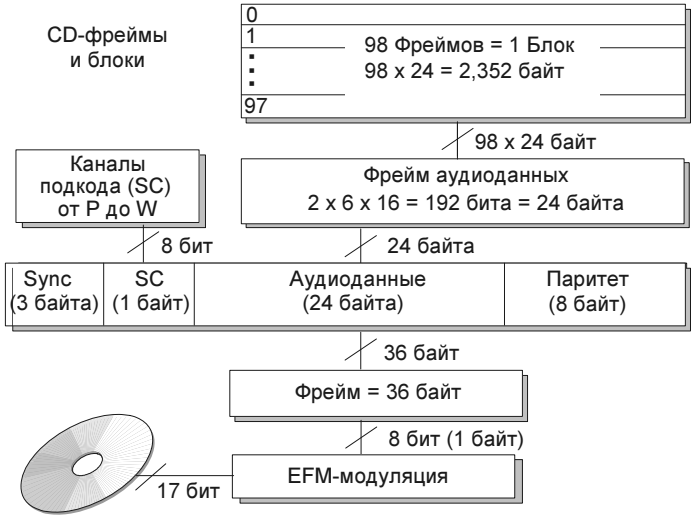


Рис. 1.7. Иерархия различных структур данных

Данные каналов P и Q поступают в виде, уже готовом к употреблению, причем значимыми из них являются только первые 12 байт, а остальные используются для выравнивания. Данные каналов P—W перед употреблением должны быть специальным образом "приготовлены" (cook). 96 составляющих их 6-битных *символов* разбиваются на 4-группы, состоящие их 24 слов. Каждое такая группа называется *пакетом* (pack) и включает в себя 16 символов пользовательских данных и 2 + 4 символа корректирующих кодов ECC/EDC *(Error Correcting Code/Error Detection and Correction)*.

Но что за информация хранится в каналах подкода? Согласно стандарту ECMA-130, "нормальные" компакт-диски задействуют лишь два канала: P и Q.

Канал P содержит в себе маркер окончания текущего трека и указатель на следующий трек, а *канал Q* используется для хранения сервоинформации, определяющей текущую позицию данного блока на диске, и является важнейшим каналом из всех.

Структурно канал Q состоит из следующих частей: *четырех управляющих битов*, соответствующих полю Control; *четырех адресных битов*, соответствующих полю q-Mode (ADR); *72 битов Q-данных*, соответствующих полю q-Data, и *16 битов контрольной суммы*, соответствующих полю CRC (рис. 1.8).

Таблица 1.1. *Формат данных Q-подканала*

Байт	Описание
0	Control/ADR
1	TNO (Track Number — номер трека)

Таблица 1.1 *(окончание)*

Байт	Описание	
2	INDEX (номер индекса)	
3	P-Min	Положение головки относительно начала трека (относительный адрес)
4	P-SEC	
5	Pframe*	
6	ZERO	
7	A-MIN	Положение головки относительно начала диска (абсолютный адрес)
8	A-SEC	
9	AFrame	
10	CRC	
11		
12	Оставлено для выравнивания	
13		
14		
15		

* В "Красной книге", соответствующей стандарту ECMA-130, данное поле именуется FRAC, а в SCSI Multimedia Commands/ATAPI DVD Devices — Frame, что создает определенные неудобства и терминологическую путаницу.

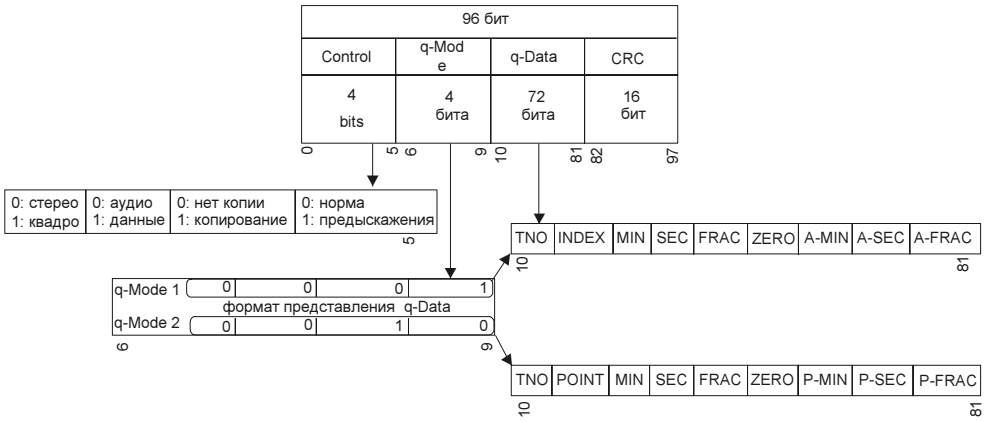


Рис. 1.8. Формат данных Q-подканала

Поле **Control** определяет содержимое трека (аудио или данные), количество аудиоканалов (стерео или квадро), а также указывает, разрешается ли копировать данные или нет. В частности, последовательность "0110" обозначает, что в пользовательской части сектора (user-data) записаны цифровые данные и их копирование не возбраняется. Напротив, последовательность "0100" запрещает копирование данных с диска. Другими словами, если третий слева, считая от нуля, т. е. 2 бит установлен, то копирование разрешено, и, соответственно, запрещено, если сброшен. Забавно, но большинство пишущих приводов всегда сбрасывает этот бит в ноль, даже если на диск записываются файлы, созданные самим пользователем. Впрочем, копировщики (в том числе и штатные) целиком и полностью игнорируют эти нелепые запреты, а потому конечный пользователь даже не догадываться о том, каких проблем он избежал!

Поле **q-Mode** определяет формат представления данных в поле **q-Data**, и для подавляющего большинства CD-ROM дисков оно равно единице.

Поле **q-Data** в режиме q-Mode == Mode 1 состоит из девяти однобайтовых полей, содержащих информацию о секторе (остальные режимы в силу их экзотичности не рассматриваются):

- **TNO** (Track **N**umber) — содержит в себе номер текущего трека, принимающий значения от 01 до 99; магическое число `0xAA` указывает на трек Lead-Out;
- **INDEX** — содержит в себе индекс текущей секции внутри текущего трека: 00 — указывает на паузу, значения от 01 до 99 идентифицируют секцию с полезными данными; однако в настоящее время эта возможность не используется и индекс секции всегда равен либо нулю (audio-pause), либо единице (actual data); индекс трека Lead-Out должен быть равен нулю;
- **MIN, SEC, FRAC** — время проигрывания сектора от начала текущего трека (минуты: секунды: фреймы соответственно), также называемое относительным временем проигрывания;
- **ZERO** — это поле должно всегда быть равно нулю;
- **A-MIN, A-SEC, A-FRAC** — время проигрывания диска от начала области данных (минуты: секунды: фреймы соответственно), также называемое абсолютным временем проигрывания.

Поле **CRC** содержит контрольную сумму содержимого *Q*-канала подкода и вычисляется по следующему полиному: $G(x) = x^{16} + x^{12} + x^5 + 1$.

Адресация секторов

Адресация сектора произошла от аудиодисков и записывается в формате Time — mm:ss:ff (минуты:секунды:доли, где доля в секунде равна от 0 до 74). Отсчет начинается с начала программной области, т. е. адреса секторов вводной области отрицательные.

Для перевода MSF[1]-адреса в LBA[2] можно воспользоваться следующей формулой: Logical Sector Address = (((Minute*60)+Seconds)*75) − 150.

"Сырые" и "сухие" сектора

IEC 908 — стандарт на аудиокомпакт-диски, вышедший в 1982 году в книге с красной обложкой (и потому вполне официально называемой *"Красной книгой"* — *Red Book*), описывал сектор как логический блок с длиной в 2352 байта, не имеющий никаких дополнительных полей и представляющий собой сплошной аудиопоток оцифрованной музыки. Что ж, логично! Сектора всех остальных накопителей (дискет, винчестеров) на логическом уровне устроены совершенно аналогично и различаются разве что длиной (в частности длина сектора гибких/жестких дисков равна 512 байтам).

К сожалению, попытка непосредственного использования аудиодиска для хранения данных потерпела неудачу. Слишком высокая плотность записи вкупе с техническим несовершенством механизма чтения привели к тому, что при воспроизведении диска постоянно возникали ошибки, количество которых на 10-секундном участке трека могло доходить до двухсот! Для аудио это вполне нормально (что русскому хорошо, то немцу — смерть), поскольку сбойные биты легко выправляются интерполяцией, и хотя достоверность воспроизведения аудиопотока при этом уже не гарантируется, человеческое ухо (даже хорошо тренированное!) все равно не замечает разницы, а потому увеличение плотности записи в угоду емкости диска вполне оправданно.

Естественно, для исправления ошибок, возникающих при чтении файлов данных, методика интерполяции абсолютно непригодна, и с вероятностью, близкой к единице, считанный файл окажется безнадежно изуродованным. Для решения этой проблемы пришлось увеличить избыточность записываемой на диск информации и ввести дополнительные корректирующие коды. По соображениям совместимости с уже имеющимся оборудованием (и производственными мощностями в том числе!) существующий формат хранения информации был полностью сохранен, но к нему добавился еще один уровень абстракции.

Стандарт *"Желтой книги"* (*Yellow Book*), вышедший в 1983 году, описывает сектор как сложную структуру, состоящую из 12-байтовой синхропоследовательности, 4-байтового заголовка, 2048-байтовой области данных, 4-байтового поля кода коррекции EDC, 8-байтовой вспомогательной области (Auxiliary) и 276-байтового поля кода коррекции ECC (рис. 1.9).

Естественно, аппаратная начинка привода CD-ROM скрывает все эти подробности и выдает содержимое служебных полей только по специальной

[1] MSF — Minutes Second Frame, минуты : секунды : фреймы.
[2] LBA — Logical Block Address.

команде (которую, кстати говоря, поддерживают не все модели). С программистской точки зрения, 2048 байта пользовательской области данных — это и есть та минимальная порция информации, с которой штатный привод может работать. Замечательно, что в результате урезания "настоящего" сектора, длина логического сектора оказалась кратной размеру секторов остальных устройств! Так какие проблемы и зачем, срывая уровни абстракции, лезть куда-то вглубь? А вот зачем. Манипулируя служебными полями, вы можете как создавать диски, не копируемые штатными средствами, так и взламывать защитные механизмы, препятствующие несанкционированному копированию.

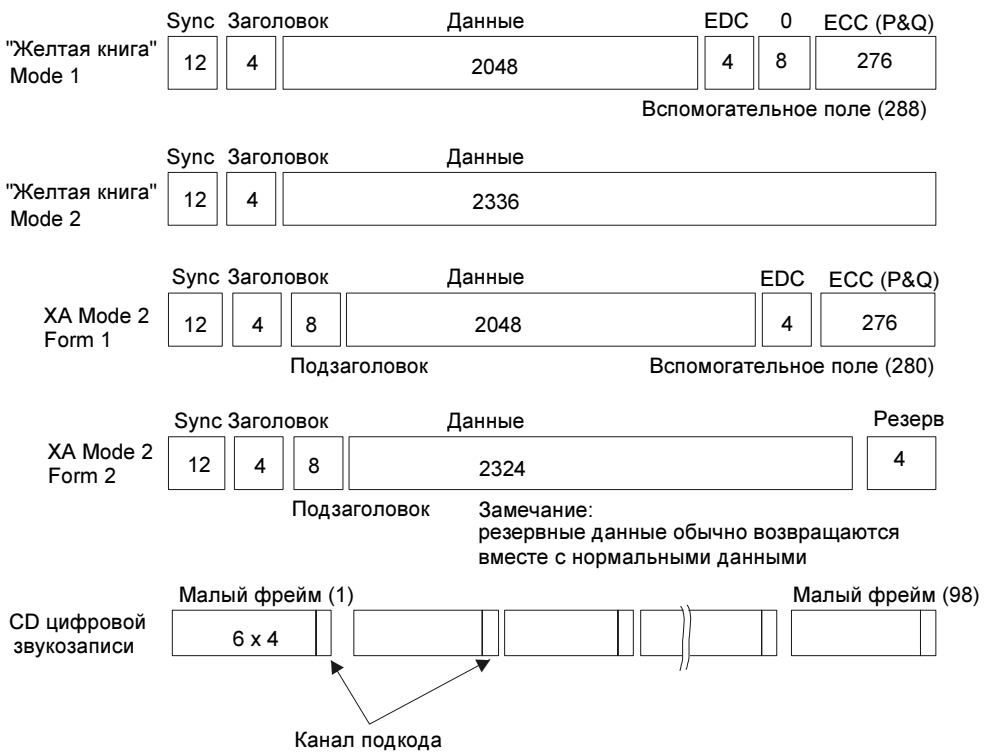


Рис. 1.9. Сектора различных типов

Если вы еще не особенно утомились сухой теорией, то совершим еще один рывок и рассмотрим формат сектора для режима MODE 1 (рис. 1.10) (на случай вашего воодушевления хочу сказать, что теория скоро закончится и начнется увлекательный процесс исследования диска под "микроскопом").

- Поле *синхронизации* (Sync) состоит из следующей последовательности: 00h FFh FFh FFh FFh FFh FFh FFh FFh FFh FFh 00h и служит индикатором начала сектора.

Сектор: 2352 байт							
Sync	Заголовок (Header)		Данные пользователя (User data)	EDC	Вспомогательное поле (Intermediate)	P-parity	Q-parity
	Адрес сектора	Режим					
12 байт	3 байта	1 (01) байт	2048 байт	4 байта	8 байт	172 байта	104 байта

0 11 12 14 15 16 2063 2064 2067 2068 2075 2076 2247 2248 2351

Рис. 1.10. Формат сектора для режима MODE 1

- Поле *заголовка* (Header) состоит из четырех байт, первые три из которых занимает физический *адрес* данного сектора (Sector Address), заданный в формате минуты:секунды:фреймы, а последний, четвертый байт, определяет формат оставшейся части сектора (Mode); если Mode = 0, то остаток сектора (а это без малого 2336 байт) трактуется как User Data и считывается без какой-либо дополнительный обработки; если же Mode = 1, то остаток сектора приобретает вид, изображенный на рис. 1.10 (как раз о нем мы сейчас и говорим!); естественно, существуют и другие режимы, но в силу их невысокой распространенности подробно останавливаться на этом вопросе мы не будем, а любопытствующих отошлем к соответствующим спецификациям.
- Область *пользовательских данных* (User Data) — 2048 байт, которые, как и следует из названия области, представляют собой непосредственно хранимую полезную информацию.
- Четырехбайтовое поле *EDC* содержит в себе контрольную сумму сектора, вычисляемую по следующему полиному: $P(x) = (x^{16} + x^{15} + x^2 + 1) \times \times (x^{16} + x^2 + 1)$, причем наименьший значимый бит четности ($x0$) записывается в наивысший значимый бит контрольной суммы, а сам подсчет контрольной суммы осуществляется с наименьшего значимого бита данных.
- Вспомогательное поле *Intermediate* хранит в себе восемь байт нулей и реально никак не используется.
- Поля *P-parity* и *Q-parity*, с длиной в 172 и 104 байта соответственно, вмещают в себя так называемые корректирующие коды Рида-Соломона (Reed-Solomon), математический принцип действия которых, равно как и сам алгоритм их генерации, здесь излагать было бы слишком утомительно, да и не нужно, поскольку все это уже подробно описано в стандарте ECMA-130. Тем более, что для подавляющего большинства задач умения вычислять корректирующие коды и не требуется, — чаще всего их просто "забивают" всякой бессмысленной чепухой, имитируя тем самым неис-

правимую ошибку чтения сектора (то есть, попросту говоря, эмулируя физические дефекты поверхности за счет создания логически не читающихся секторов — для взлома защит, привязывающихся к физическим дефектам — это самое то!).

Синхрогруппы, объединяющие биты и DSV

Объединяющие биты (*merging bits*) решают по меньшей мере три важнейших задачи, без которых считывание информации с лазерного диска было бы невозможным.

Во-первых, объединяющие биты препятствуют возникновению конфликтных ситуаций, образующихся на стыке двух EFM-слов, одно из которых оканчивается на единицу, а другое с этой самой единицы и начинается (рис. 1.11). Поскольку две единицы (каждая из которых соответствует переходу с пита (pit) в ленд (land) или наоборот) должны быть разделены по меньшей мере двумя нулями, такая комбинация считается запрещенной, — считывающее устройство попросту не заметит, что здесь что-то есть (протяженность одного пита/ленда намного меньше диаметра сфокусированного лазерного пятна и потому для уверенного распознавания его длину приходится увеличивать как минимум до 3T, подробнее см. рис. 1.12). С другой стороны, если "хвост" одного из EFM-слов состоит из восьми подряд идущих нулей, а другое EMF-слово с тех же самых восьми нулей и начинается,

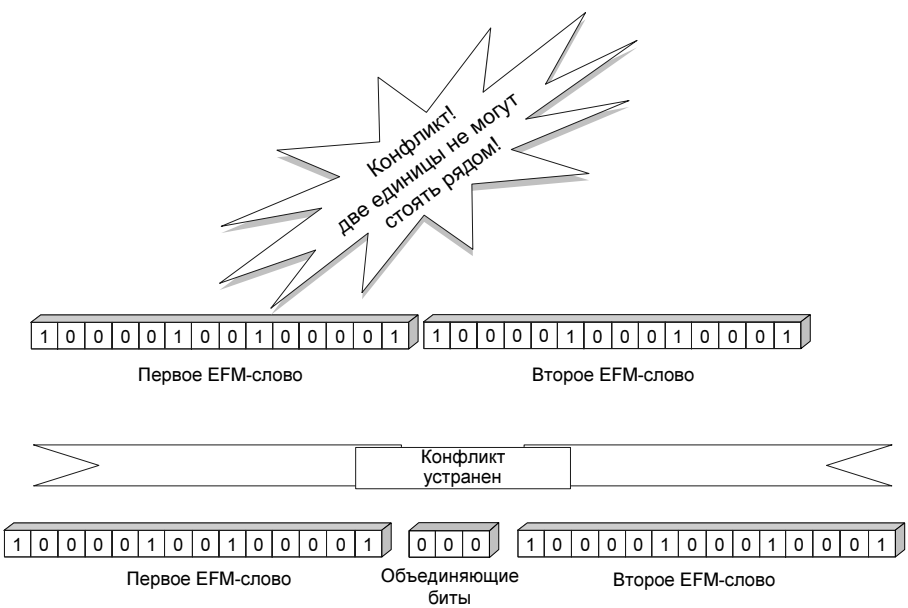


Рис. 1.11. Объединяющие биты в действии

то на их стыке образуется цепочка из шестнадцати нулей, при чтении которой произойдет ошибка, т. к. по стандарту между двумя соседними единицами должно находиться не более одиннадцати нулей, в противном случае погрешность определения длины такой последовательности становится очень большой. Вы не пробовали измерять ученической линейкой расстояние между Ленинградом и Москвой? Попробуйте, и тогда прочувствуете всю суть проблемы целиком. Короче говоря, объединяющие биты выбираются так, чтобы между двумя соседними единицами было не меньше трех, но и не больше одиннадцати нулей.

Во-вторых, объединяющие биты предотвращают возникновение ложных синхрогрупп в неположенном месте. Последовательность бит, образующих синхрогруппу (для справки это — 100000000001000000000010), может встречаться только в заголовке фрейма и служит в качестве своеобразного индикатора его начала. Когда читающая головка перемещается поперек спиральной дорожки в поисках заданного сектора, ей приходится выяснять, в каком именно месте она находится в данный момент: в начале, середине фрейма или даже середине EFM-слова. Как же это осуществляется? Считывающее устройство пропускает через себя поток цифровых данных до тех пор, пока ему не встретится очередная синхрогруппа. *Ага!* — говорит "думательное" устройство CD-ROM-привода, — *теперь-то я точно знаю, что это действительно начало нового фрейма, а не что-нибудь еще!* Представляете, какая "каша" началась, если бы "паразитная" синхрогруппа случайно появилась бы в середине фрейма? А ведь не будь объединяющих битов, такое происшествие происходило бы регулярно! Рассмотрим, например, следующие EFM-слова: 10000000000100 и 00000000100100. Если их "склеить" вместе, то образуется паразитная синхрогруппа 100000000001000000000010 (которая, как было указано ранее, может встречаться только в заголовке фрейма) и при попытке чтения такого фрейма произойдет крах. Объединяющие биты, связывающие такие EMF-слова, позволяют всего этого избежать.

В-третьих, посмотрите на рис. 1.12, — лазерный диск не имеет никакой иной разметки, кроме самой спиральной дорожки, состоящей из чередующихся питов и лендов, при быстрой смене которых возникает высокочастотный, так называемый *HF* (*High Frequency*), сигнал, схематически изображенный на графике (*б*). Этот сигнал имеет большое значение в удержании считывающей головки на спиральной дорожке, поскольку никакого другого способа отличить дорожку от междорожечных "междурядий" и нет. Другая сложность связана с отсутствием опорного сигнала, без которого считывающее устройство не может с уверенностью отличить темные участки поверхности от светлых. Некоторые источники, и в том числе FAQ, по CD-ROM утверждают, что *"если в фотодатчик попадает яркий свет (стандартом предусмотрено, что при полном отражении должно отражаться не менее 70 процентов света, то "проигрыватель" понимает, что это ровное место на диске ("land"), а если в датчик попадает менее яркий свет, это означает, что в данном месте*

на диске находится углубление ("pit")". На самом деле, это довольно вольная интерпретация стандарта, который буквально говорит следующее: "The information contained in the HF signal is extracted in the form of the positions of the crossings of the HF signal with a decision level I_D. This decision level I_D is the level in the middle of the extreme values of I_3" — *"Информация, заключенная в HF-сигнале, извлекается в форме позиции, в которой происходит пересечение сигнала с пороговым уровнем I_D. Пороговый уровень определяется как половина максимального значения I_3 (I_3 — уровень сигнала, соответствующий максимальной частоте смены питов и лендов)"*. Величина в 70% взята совсем из другого раздела стандарта, описывающего, какой ***должна быть*** амплитуда модуляции (Modulation amplitude), но ничего не говорящий о том, как именно она модулируется! Как говорится, — почувствуйте разницу!

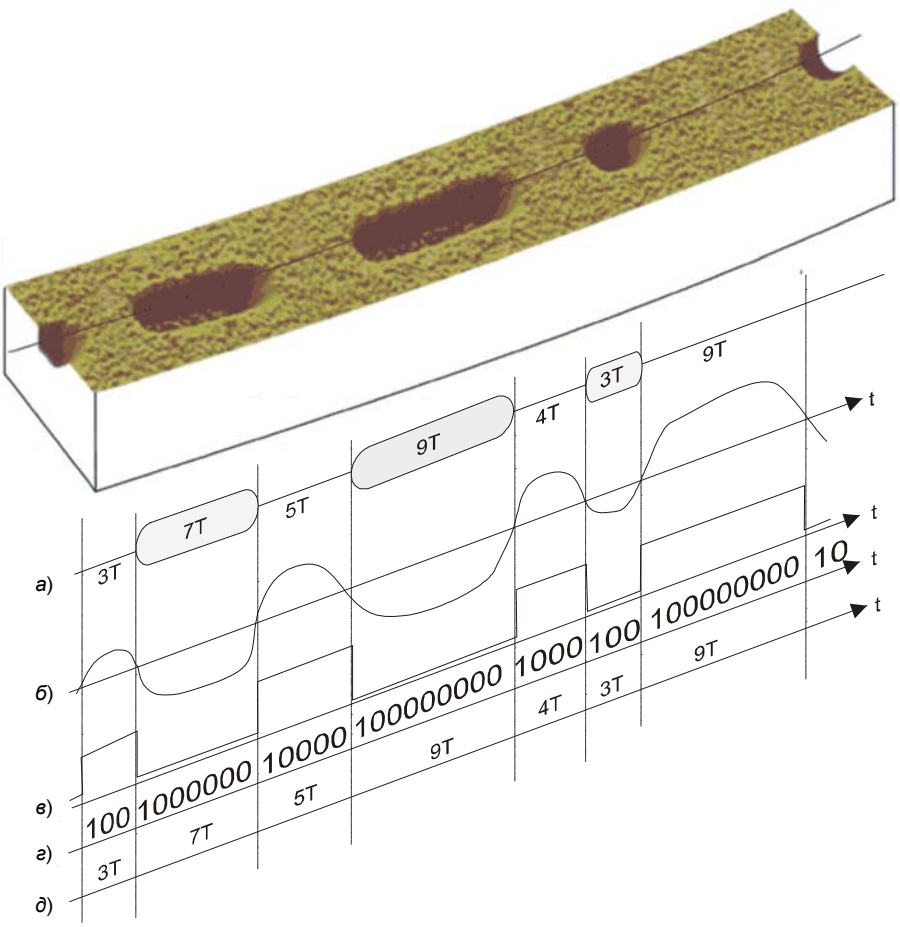


Рис. 1.12. Форма высокочастотного сигнала, образующегося при чтении последовательности питов (pits) и лендов (lands), и его интерпретация

А теперь представьте, что произойдет, если на каком-то участке спиральной дорожки окажется значительный избыток питов по отношению к лендам, или наоборот. Вместо переменного высокочастотного сигнала привод будет считывать постоянный ток высокого или низкого уровня, но в отсутствии опорного сигнала "думающему" устройству привода будет не так-то просто разобраться, какой из них какой! Короче говоря, в пределах одного фрейма питов и лендов должно быть поровну.

Но за счет чего такое равенство достигается?! Не можем же мы в самом деле писать на диск строго упорядоченные последовательности! К тому же, даже беглого взгляда на таблицу кодировки EMF достаточно, чтобы понять, что ноль явно доминирует над единицей, и какие бы EMF-последовательности мы только ни записывали, — "кворума" по единицам мы не наберем. Стоп! Ведь между двоичными битами и питами (лендами) нет прямого соответствия, и двоичный ноль может быть закодирован как питом, так и лендом! Представим себе, что мы записываем на диск EMF-последовательность "10000000100000". Несмотря на явный избыток двоичных нулей, этот EMF-код содержит приблизительно одинаковое количество питов и лендов (рис. 1.13, *а*).

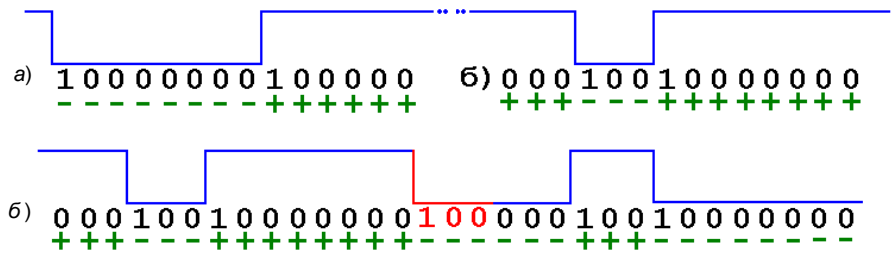


Рис. 1.13. Демонстрация вычисления DSV

Для более точного подсчета этого соотношения ввели специальную величину — **DSV** (***D**igital **S**um **V**alue*), которая вычисляется следующим образом: изначально DSV равно нулю, но каждый пит увеличивает его на единицу, а каждый ленд на единицу уменьшает. В частности, для EFM-последовательности "10000000100000" значение DSV равно двум (см. рис. 1.13, *а*). Именно двум, а не "минус двум", т. к. нас интересует только модуль числа, но не его знак (действительно, если бы эта последовательность начиналась не с пита, а с ленда, то мы бы получили прямо противоположный результат — восемь "+" и шесть "–").

По стандарту, значение DSV, вычисленное для целого фрейма, должно находиться в интервале от 2 до 10, в противном случае, такой сектор будет читаться с трудом, если вообще будет читаться! А ведь далеко не все EFM-коды могут похвастаться невысоким значением DSV, — взять, например,

хотя бы "00010010000000", DSV которого равно 8. Да, эта величина формально удовлетворяет требованиям стандарта, но если таких последовательностей на диске наберется хотя бы с десяток, значение DSV катастрофически возрастет аж до *восьмидесяти!!!*

Снижения уровня DSV до предельного допустимого минимума — это и есть третья задача, с которой справляются объединяющие биты. Как они это делают? Взгляните на рис. 1.13, *б* — где за одним EFM-словом, обладающим резко положительным DSV, следует другое EFM-слово с высоким DSV. На самом деле, как уже говорилось ранее, DSV не имеет знака, точнее, действительный знак EFM-слова зависит не от самого этого слова, но от его контекста! Рассмотрим это на примере вырожденной последовательности: "...00000000...". Поскольку двоичный ноль соответствует отсутствию изменений в данном месте поверхности диска, то эта последовательность может быть закодирована как восемью лендами, так и восемью питами. Теперь предположим, что мы записываем на диск два EFM-слова, каждое из которых имеет значительный избыток лендов. Можно ли превратить ленды второго слова в питы? Да, если один из трех объединяющих битов будет равен единице. А единица, как известно, соответствует переходу из ленда в пит (ну или наоборот), в результате чего второе EFM-слово начнется с пита и его значение DSV станет резко отрицательным (то есть, попросту говоря, она приобретет избыток питов над лендами). В результате, EFM-слово с резко положительным DSV будет в той или иной степени компенсировано EFM-словом с резко отрицательным DSV, и их общее DSV окажется где-то в районе нуля (см. рис. 1.13).

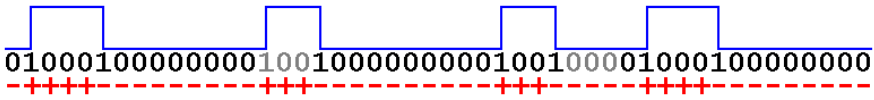


Рис. 1.14. EFM-последовательность с катастрофически низким DSV

Зачем вам, прикладному программисту, знать все эти подробности физического кодирования? А затем, что требования, предъявляемые к объединяющим битам, взаимно противоречивы и могут существовать крайне неблагоприятные EFM-последовательности с неустранимо высоким значением DSV. Одна из таких последовательностей и показана на рис. 1.14. Смотрите, на конце первого EFM-слова находится восемь нулей, а поскольку более десяти подряд идущих нулей категорически недопустимо, первый или второй из объединяющих битов обязательно должен быть равен единице. Но в этом случае, следующее EFM-слово приобретает резко отрицательное значение DSV, которое никак не удается скомпенсировать, поскольку между вторым и третьим EFM-словами может присутствовать лишь одна-единственная комбинация объединяющих бит "000". Все остальные обречены на "провал",

т. к. нарушают правило "не менее двух нулей между соседними единицами". В результате, третье EMF-слово также выходит с резко отрицательным DSV-значением, и, если мы заполним этой последовательностью весь сектор целиком, его суммарное DSV окажется катастрофически отрицательным!

Для предотвращения появления подобных последовательностей, все записываемые на диск данные предварительно *скремблируются*, т. е. преобразуются в псевдослучайную последовательность, близкую по своим характеристикам к "белому шуму". Соответственно, при чтении данных выполняется обратная операция. Однако при желании скремблер легко обойти! Некоторые защиты от копирования именно так и поступают *(см. разд. "Защиты, основанные на «слабых» секторах" главы 9)*.

Скремблирование

Перед записью на диск, содержимое сектора в обязательном порядке подвергается *скремблированию* (от английского *scrambling — перемешивание*), т. е. преобразуются в псевдослучайную последовательность, близкую по своим характеристикам к "белому шуму", что исключает непреднамеренное образование регулярных последовательностей с высоким значением DSV — такие последовательности с точки зрения считывающего устройства считаются крайне неблагоприятными и читаются весьма нестабильно *(подробнее об этом см. разд. "Синхрогруппы, объединяющие биты и DSV" этой главы)*.

Скремблируются все поля сектора, кроме 12-байтовой синхрогруппы в его начале (если скремблировать и синхрогруппу, то как потом ее прикажете находить в потоке данных?), что в совокупности составляет 2340 байт данных (рис. 1.15).

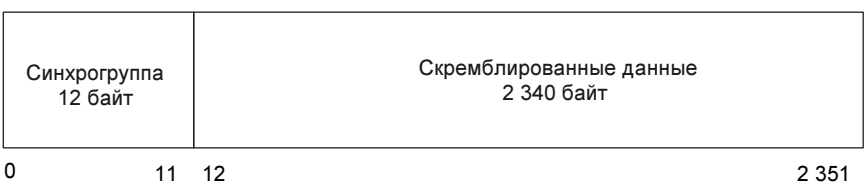


Рис. 1.15. Формат сектора с точки зрения скремблера

Скремблирование осуществляется записывающим приводом на аппаратном уровне и совершенно незаметно для программиста. Соответственно, при чтении сектора выполняется обратная процедура, т. е. происходит "скремблирование наоборот" или "дескремблирование", в процессе чего сектор очищается от "белого шума" и преобразуется к своему исходному виду.

Прозрачность механизма скремблирования создает обманчивое впечатление, что значение его алгоритма совершенно бесполезно для программиста

и представляет интерес лишь для разработчиков аппаратуры. На самом же деле это не так. Коль скоро скремблер для того и придуман, чтобы исключить ***непреднамеренное*** появление неблагоприятных последовательностей, то, стало быть, умение ***умышленно*** формировать такие последовательности позволяет создавать диски, не читаемые на аппаратном уровне. — Тоже мне новость! — хмыкнете вы. Создать нечитаемый диск — много ума не надо. Вжик циркулем по отражающему слою — и "родная мама его не узнает". А еще есть пресс-папье и просто кувалда. Шутки шутками, но весь фокус в том, что благодаря наличию корректирующих кодов можно создать неблагоприятную последовательность, вычислить соответствующие ей корректирующие коды и в слегка измененном виде записать эту последовательность на диск так, чтобы с одной стороны она из неблагоприятной превратилась в благоприятную, а с другой — при прохождении сквозь декодер Рида-Соломона восстанавливать в свой исходный — неблагоприятный вид. Попытка скопировать такой диск штатным копировщиком ни к чему хорошему не приведет, т. к. он запишет неблагоприятную последовательность в том виде, как она есть, при чтении которой наверняка произойдет ошибка! Захватывающая перспектива, не правда ли? Подробнее об этом приеме мы поговорим в *главе 6*, а пока же сосредоточимся на скремблере.

Алгоритм скремблирования, согласно стандарту ECMA-130, выглядит так: "Each bit in the input stream of the scrambler is added modulo 2 to the least significant bit of a maximum length register. The least significant bit of each byte comes first in the input stream. The 15-bit register is of the parallel block synchronized type, and fed back according to polynomial $x^{15} + x + 1$. After the Sync of the Sector, the register is pre-set with the value 0000 0000 0000 0001, where the ONE is the least significant bit" *("Каждый бит входного потока скремблера суммируется по модулю 2 с наименее значимым битом максимальной длины регистра. Наименее значимый бит каждого байта проходит первым во входном потоке. 15-битный регистр представляет собой параллельный блок синхронизированного типа и заполняется в соответствии с полиномом $x^{15} + x + 1$. После прохождения синхрогруппы сектора, которая не скремблируется, регистр инициализируется значением 0000.0000.000.0001, где ЕДИНИЦА есть наименее значимый бит"* (рис. 1.16, листинг 1.1).

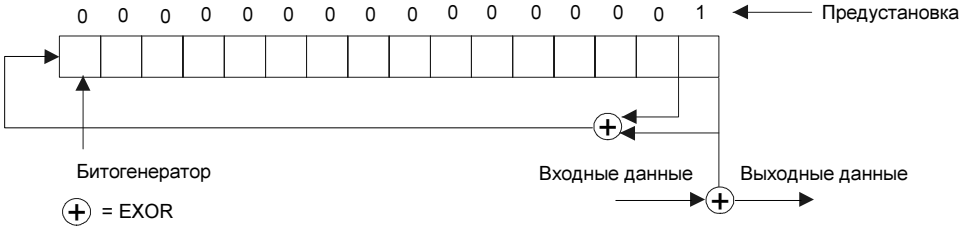


Рис. 1.16. Блок-схема скремблера

Листинг 1.1. Программная модель скремблера к рис. 1.16

```
UpdateShiftRegister()
{
     int i;
     for(i = 0; i < 8;i++)
     {
          int hibit = ((ShiftRegister & 1)^((ShiftRegister & 2)>>1)) << 15;
          ShiftRegister = (hibit | ShiftRegister) >> 1;
     }
}

void Scramble()
{
     int i;
     for (i=12;i<2352;i++)
     {
          Scrambled[i] = Scrambled[i] ^ (ShiftRegister&0xFF);
          UpdateShiftRegister();
     }
}
```

Непонятно? Мне тоже было непонятно, во всяком случае до тех пор, пока я не "взял в руки дизассемблер" и не "распотрошил" CloneCD. Как известно, CloneCD великолепно справляется с защитами, основанными на слабых секторах, а раз так, то он должен содержать в себе свой собственный скремблер.

Действительно, среди функций, экспортируемых динамической библиотекой ElbyECC.dll (входящей в состав Clone CD), обнаруживается одна очень любопытная функция: `RawSrcambleSector`, дизассемблерный листинг которой выглядит следующим образом (листинг 1.2).

Листинг 1.2. Пример реализации алгоритма скремблирования, позаимствованный из Clone CD

```
.text:100020E0 RawScrambleSector proc near
.text:100020E0
.text:100020E0 arg_0 = dword ptr  4
.text:100020E0
.text:100020E0 mov   eax, [esp+arg_0] ; загружаем переданный аргумент в EAX
.text:100020E4 mov   ecx, offset ScrmblrTbl; в ECX - указатель на ScrmblrTbl
.text:100020E9 add   eax, 0Ch         ; пропускаем 12 байт синхропослед.
```

```
.text:100020EC push  esi              ; сохраняем ESI
.text:100020ED push  edi              ; сохраняем EDI
.text:100020EE sub   ecx, eax         ; вычисляем дельту
.text:100020F0 mov   edx, 249h        ; 2340 / 4 скремблируемых байт
.text:100020F5
.text:100020F5 loc_100020F5:          ; CODE XREF: RawScrambleSector+22↓j
.text:100020F5 mov   esi, [ecx+eax]   ; взять очередной DWORD из таблицы
.text:100020F8 mov   edi, [eax]       ; взять очередное скремблируемое DWORD
.text:100020FA xor   edi, esi         ; ксорим
.text:100020FC mov   [eax], ed        ; записываем результат
.text:100020FE add   eax, 4           ; следующий DWORD
.text:10002101 dec   edx              ; уменьшить счетчик
.text:10002102 jnz   short loc_100020F5    ; мотать цикл
.text:10002104 pop   edi              ; восстанавливаем регистр EDI
.text:10002105 pop   esi              ; восстанавливаем регистр EDI
.text:10002106 retn                   ; возвращаемся из функции
.text:10002106 RawScrambleSector endp
```

Анализ дизассемблерного листинга показывает, что громоздкой и ресурсоемкой "возне" с полиномом разработчики CloneCD предпочли быстрый табличный алгоритм, сводящийся к наложению на скремблируемый сектор псевдослучайной последовательности посредством операции `XOR`. Фактически, мы получили не что иное, как "одноразовый блокнот" Вернама, длина "секретного" ключа которого равна длине скремблируемой части сектора (2340 байт для справки). Будучи переведенным на язык высокого уровня, данный алгоритм будет выглядеть приблизительно так, как это показано в листинге 1.3.

Листинг 1.3. Пример реализации табличного алгоритма скремблирования на языке Си

```
RawScrambleSector (char *raw_sector)
{
     int a;
     DWORD *p;
     DWORD *MyScramblerTable = (DWORD *) ScramblerTable;

     p = (DWORD*)(raw_sector + 12);
     for (a = 0; a < 2340 / 4; a++)
     {
          p[a] ^= MyScramblerTable[a];
     }
}
```

Теперь остается разобраться непосредственно с самой псевдослучайной последовательностью, первые восемь членов которой (полученные с помощью дизассемблера из того же CloneCD) выглядят следующим образом (листинг 1.4).

Листинг 1.4. Первые восемь членов псевдослучайной последовательности, используемой для скремблирования сектора, полученные из копировщика CloneCD

```
dd 060008001h
dd 01E002800h
dd 006600880h
dd 081FE02A8h
dd 028606080h
dd 0889E1E28h
dd 0AAAE6668h
dd 0E0017FFCh
```

Вся таблица слишком велика, для того чтобы быть приведенной здесь целиком (даже напечатанная самым мелким шрифтом, который только допускают санитарные нормы, она занимает целую страницу, — свыше четырех тысяч знаков, которые очень трудно "перебить" с бумаги в компьютер, не допустив при этом ошибок). Поэтому представляет интерес найти закономерность, которой связаны члены данной последовательности, и воссоздать алгоритм, вычисляющий все члены последовательности по первому из них. Эта маленькая программистская головоломка отнюдь не так сложна, какой кажется поначалу. Да, беглый взгляд на первые восемь членов псевдослучайной последовательности не обнаруживает и намека на их природу — числа меняются хаотично и сильно смахивают на "пляшущих человечков", над которыми ломал голову герой Шерлока Холмса. Только частотный анализ в данном случае бесполезен и в "лоб" эту задачу не решить. Но ведь мы начинаем анализ отнюдь не на пустом месте! Во-первых, нам достоверно известно, что скремблирование осуществляется по 16-разрядным словам (разрядность регистра скремблера как раз и составляет 16 бит), а раз так, то и анализировать мы должны именно слова, а не двойные слова. То, что скремблирование осуществляется 32-битными кусками, ничего не меняет, ведь `XOR`-битовая операция и потому конкретная разрядность операндов никак не влияет на конечный результат! Во-вторых, анализ закономерностей выгоднее всего проводить на битовом уровне, т. к. именно на битовом уровне эта псевдослучайная последовательность и генерируются.

Скрипт, показанный в листинге 1.5, автоматически преобразует все элементы таблицы в 16-разрядные слова, отображаемые в двоичном виде. Запустите дизассемблер IDA Pro, нажиме <F2> и загрузите файл, содержащий этот

скрипт. Затем, подогнав курсор к первому элементу таблицы, нажмите <Shift-F2> и введите следующую команду: `x2bin(ScreenEA(), ScreenEA()+2340, 2)`. Комбинация <Ctrl-Enter> (в ранних версиях IDA Pro просто <Enter>) запустит скрипт на выполнение.

Листинг 1.5. Скрипт на IDA-Си, преобразующий элементы таблицы в двоичный вид

```
// x_start - начальный адрес для преобразования
// x_len   - кол-во байт для преобразования
// x_pow   - кол-во байт в одном элементе
static x2bin(x_start, x_end, x_pow)
{
    auto a,p;
    for(p=x_start;;)
    {
         // преобразуем в элемент нужной разрядности
         if (x_pow == 1) MakeByte(p); else if (x_pow == 2) MakeWord(p); else
         if (x_pow == 4) MakeDword(p); else return 0;

         // в двоичный вид
         OpBinary(p, 0);

         // след. элемент
         p = p + x_pow;

         // выход, если все уже сделано
         if (p>x_end) break;
    }
}
```

"Обновленная" псевдослучайная последовательность скремблера должна выглядеть так, как показано в листинге 1.6 (в нем приведены 16 ее первых членов).

Листинг 1.6. Псевдослучайная последовательность, записанная в виде 16-разрядных слов, отображаемых в двоичной нотации

```
dw 1000000000000001b
dw 0110000000000000b
dw 0010100000000000b
dw 0001111000000000b
dw 0000100010000000b
```

```
dw 0000011001100000b
dw 0000001010101000b
dw 1000000111111110b
dw 0110000010000000b
dw 0010100001100000b
dw 0001111000101000b
dw 1000100010011110b
dw 0110011001101000b
dw 1010101010101110b
dw 0111111111111100b
dw 1110000000000001b
```

Теперь определенная закономерность сразу же бросается в глаза (вот что значит правильно отформатировать листинг!). Биты каждого последующего элемента смещаются на одну позицию вправо, прижимаясь к логическому "востоку", образуя своеобразную битовую "струю", которая в своем диагональном течении линейно увеличивается в размерах (каждый последующий элемент добавляет ей еще один бит ширины), но на определенном этапе внезапно разбивается на ряд более мелких ручейков, причудливые переплетения которых образуют бессмысленную мешанину. Тем не менее, "физические" принципы, лежащие в основе этого "гидрологического" сооружения, все еще скрыты непроницаемой вуалью тумана, и нам не остается ничего иного, кроме как брести наугад, полагаясь лишь на свою интуицию и удачу.

Что мы знаем? Немного Скремблер выполняет операцию `XOR` над содержимым своего внутреннего регистра и потоком скремблируемых данных и после каждого "отскремблированного" 16-разрядного слова модифицирует значение этого регистра, но вот как?! Давайте возьмем два соседних элемента нашей псевдослучайной последовательности и попробуем подобрать такую последовательность операций, которая из предыдущего элемента порождала бы следующий. Возможных вариантов не так уж и много: сдвиг, `XOR`, `AND` и `OR`. Навряд ли создатели CD-ROM использовали в скремблере что-то еще.

Итак, будем отталкиваться от того, что "Исаак родил Абрама", то есть от того, как скремблер из числа `0110000000000000b` неизвестным науке образом получил число `0010100000000000b`. Для компенсации сдвига (который явно имел место) сместим последующий элемент на один бит влево и запишем оба числа одно под другим (по типу сложение по модулю два в столбик):

```
dw 0110000000000000b
dw ????????????????b  XOR
-------------------
dw 0101000000000000b
```

Неизвестное слагаемое здесь не найдет только ленивый. Операция `XOR` над `0110000000000000b` и `0101000000000000b` дает `0011000000000000b`. Постой!

Но ведь это же наш исходный член, только сдвинутый на одну позицию вправо! Ну-ка, а как поведет себя следующая пара чисел? Сгорая от нетерпения, складываем:

```
dw 0101000000000000b
dw ?????????????????b  XOR
-------------------
dw 0001111000000000b
```

Ага, операция `XOR` над `0101000000000000b` и `0001111000000000b` дает `0010100000000000b`, значит, мы на верном пути! Наскоро написав простейший скрипт, вычисляющий последующие члены на основе предыдущих, мы получим верный результат для всех членов последовательности со второго по седьмой включительно, а вот дальше... Дальше теория и практика разойдутся, как большинство супругов. Неожиданно в старшем разряде возникнет единица, которую никто не ждал и которая, естественно, в следующей итерации порождает паразитный "ручеек". Может быть, сдвиг бит в слове происходит циклически, т. е. младший бит "заворачивается" наверх? Но нет, попытка вычисления последующих членов не подтверждает этой теории.

Провозившись пару часов в попытках найти в доступной мне литературе хоть какую-то информацию о полиномах и особенностях их реализации, я так ни к чему и не пришел. В конце концов, вспомнив, что "лучше за день долететь, чем за час добежать", я просто распечатал первую сотню членов псевдослучайной последовательности и вручную рассчитал каждый последующий элемент на основе предыдущего данного. Затем просто отметил все обнаруженные исключения. Выяснилось, что 15-й и 14-й биты (считая от нуля) временами совершают "самопроизвольные" перескоки из нуля в единицу и наоборот. Остальные биты вели себя в полном соответствии с теорией.

Осталось всего-то ничего — выяснить при каких условиях происходят эти битовые "мутации". Быстро обнаружилось, что если первый, считая от нуля, бит вычисленного члена последовательности равен единице, то его 15-й бит инвертируется (это действительно легко заметить, особенно на распечатке). Чуть сложнее далось второе исключение из правил: если нулевой бит вычисленного члена равен единице, то его 15-й и 14-й биты инвертируются. Соответственно, если и 0-й и 1-й биты содержат по единице, до в силу двойной инверсии 15-го бита реально инвертируется лишь 14-й бит (рис. 1.17). Все! Теперь мы можем рассчитать нашу псевдослучайную последовательность целиком!

Исходный текст программы, генерирующей скремблерную последовательность, приведен в листинге 1.7. Здесь и далее в квадратных скобках приведено название файла, находящегося на прилагаемом к книге компакт-диске с соответствующим листингом. Конечно, это далеко не верх оптимизации, но в плане наглядности данный пример весьма удачен.

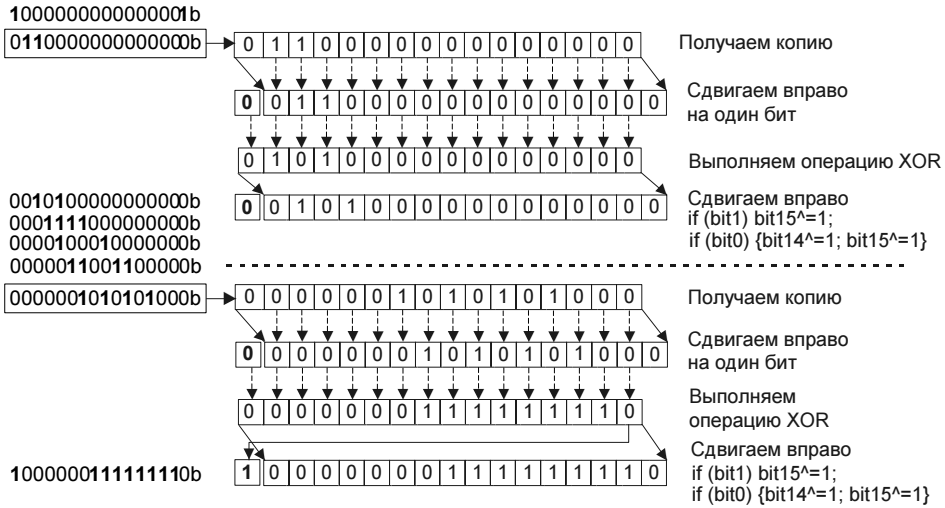


Рис. 1.17. Расчет скремблируемой последовательности

Листинг 1.7. Программа для расчета скремблируемой последовательности

```
/*-----------------------------------------------------------------------------
 *
 *          ГЕНЕРИРУЕТ ПОСЛЕДОВАТЕЛЬНОСТЬ ДЛЯ CD-СКРЕМБЛЕРА
 *          ==============================================
 *
 * build 0x001 @ 07.06.2003
-----------------------------------------------------------------------------*/
#include <stdio.h>

// контрольный фрагмент истинной последовательности для проверки программы
// ---------------------------------------------------------------------
//0x8001,0x6000,0x2800,0x1e00,0x0880,0x0660,0x02a8,0x81fe,0x6080,0x2860,0x1e28,
//0x889e,0x6668,0xaaae,0x7ffc,0xe001,0x4800,0x3600,0x1680,0x0ee0,0x04c8,0x8356,
//0xe17e,0x48e0,0x3648,0x96b6,0xeef6,0xccc6,0xd552,0x9ffd,0xa801,0x7e00,0x2080,

printf_bin(int a)
{
      int b;
      for(b = 15; b >= 0; b--) printf("%x",(a & (1<<b))?1:0);printf(" %x\n",a);
```

```
}

main()
{
      int a, tmp;
      int reg = 0x8001; // первый элемент скремблерной последовательности

      for(a = 1; a < 1170/* длинна скремблируемой части сектора в словах*/; a++)
      {
            // вывод на печать
            printf_bin(reg);
            if ((a % 8) == 0) printf(".............%03d................\n",a /8);

            // сложение по модулю два со свдигом
            tmp = reg >> 1; tmp = reg ^ tmp; reg = tmp >> 1;

            // обработка полинома x^15+x+1, что эквив. 1<<15 + 1<<1 + 1<<0
            if (reg & 1<<1) reg = reg ^  (1<<15);
            if (reg & 1<<0) reg = reg ^ ((1<<15) | (1<<14));
      }
}
```

F1-, F2- и F3-фреймы, CIRC-кодирование

Следующий "производственный" цикл начинается с того, что наскоро отскремблированный сектор "нарезается" на 24-байтовые "ломтики" данных, называемые *F1-фреймами* (*F1-frames*). Путем несложных математических подсчетов можно заключить, что одного 2352-байтового сектора аккурат хватает на 98 F1-фреймов.

F1-фрейм

Структура F1-фреймов чрезвычайно проста: каждый фрейм состоит из 24 байтов (12 слов), пронумерованных от 0 до 23, последовательно отображаемых на соответствующие ячейки сектора (рис. 1.18, *а*). Причем границы фреймов и секторов не обязаны совпадать, и сектор может начинаться с любой из следующих позиций первого фрейма: 0, 4, 8, 12, 16 или 20 (рис. 1.18, *б*). Стартовая позиция сектора в F1-фрейме нигде не хранится (да и где ее здесь хранить-то?), вместо этого начало сектора опознается по синхрогруппе, которую трудно не заметить!

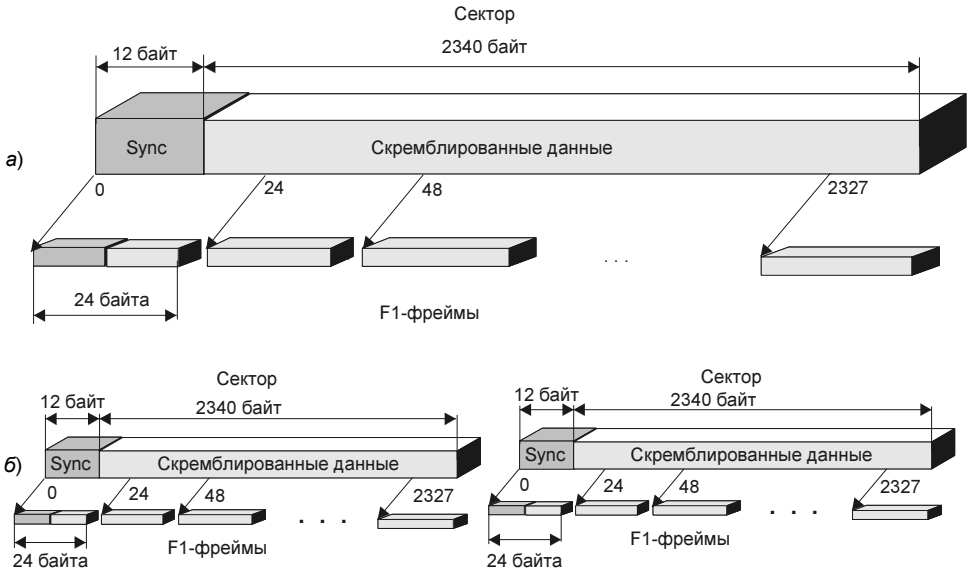


Рис. 1.18. Схема отображения секторов на фреймы

Стандарт довольно туманно описывает процесс отображения секторов на фреймы, но дает понять, что за концом одного сектора непосредственно следует начало другого (*Byte 2.351 of a Sector is immediately followed by byte 0 of the next Sector*), следовательно, изменение стартовой позиции сектора не "заворачивает" хвост сектора в начало первого фрейма, а переносит его на следующий фрейм. Короче говоря, если стартовая позиция сектора не равна нулю, то каждый 49-й фрейм содержит байты сразу *двух* секторов (как нетрудно сообразить, это будут первый и последний фреймы сектора, а поскольку в одном секторе содержится 98 фреймов, то 98/2 = 49)!

Изменение стартовой позиции начального байта сектора во фрейме приводит к значительным изменениям его DSV *(см. разд. "Синхрогруппы, объединяющие биты и DSV" этой главы)*, в результате чего у записывающей аппаратуры появляется возможность "нормализации" секторов с неприлично высоким значением DSV. Микропрограмма привода должна выбирать стартовую позицию с наименьшим возможным DSV или, по крайней мере, следить за тем, чтобы величина DSV не выходила за рамки предельно допустимых значений. К сожалению, подавляющее большинство пишущих приводов CD-R бытового класса слишком тупы для подобной задачи и всегда начинают фрейм с нулевого байта сектора. Как следствие, — копия диска на фреймовом уровне может очень значительно отличаться от своего оригинала. Несмотря на то, что программно определить стартовую позицию невозможно (штатные приводы CD-ROM отказываются "разглашать" эту

информацию), разработчикам защитного механизма ничего не стоит сформировать "слабый" (weak) сектор, имеющий крайне высокое DSV при стартовой позиции 0, но вполне нормальное — при всех остальных *(см. разд. "Защиты, основанные на «слабых» секторах" главы 9)*. Скопировать такой сектор обычным пишущим приводом практически невозможно, — лишь немногие модели приводов CD-ROM окажутся способными прочесть сектор с высоким DSV. Мой ASUS-50x как будто бы читает такие сектора, но нестабильно и не на всех дисках. Причем возможности "ручного" выбора стартовой позиции у известных мне пишущих приводов нет (во всяком случае, в стандарте она отсутствует, да и не спустились еще записывающие приводы на такой низкий уровень). Можно, конечно, схитрить и умышленно исказить несколько байт сектора, не трогая при этом корректирующие коды (даже незначительная модификация исходных данных ведет к чудовищному изменению DSV), так что микропроцессорная начинка привода самостоятельно восстановит первоначальное содержимое искаженных данных "налету". Однако защита (если она, конечно, не совсем дура) может запросто разоблачить такую грубую подделку от оригинала, — ведь при чтении сектора в "сыром" виде все махинации с корректирующими кодами сразу же становятся видны!

В то же время большинство пишущих приводов CD-RW (если не все) тщательно следят за значением DSV и корректно выбирают стартовую позицию. Что ж! Все логично, — контрастность CD-RW-носителей и так до безобразия низка, а потому и требования к величине DSV здесь значительно жестче, чем у CD-R-дисков. Отсюда: если защищенный диск не копируется на CD-R, попробуйте скопировать его на приводе CD-RW на одноименную болванку, ибо ряд пишущих приводов CD-RW (Plextor, PHILIPS) при записи CD-R-болванок всегда начинают фрейм с нулевого байта сектора, но вместе с этим грамотно определяют стартовую позицию сектора при записи на болванку типа CD-RW! (маразм, конечно, но против него не попрешь).

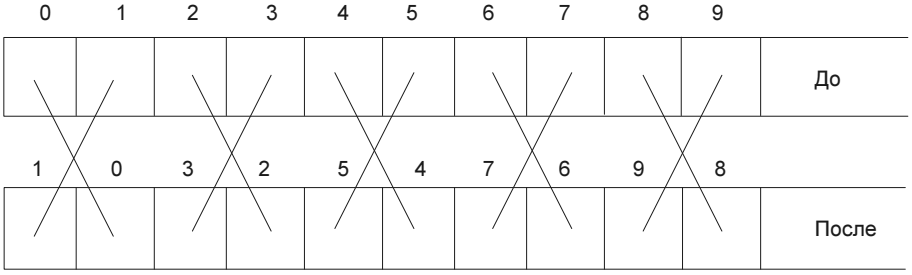


Рис. 1.19. Схема перемешивания байтов в F1-фрейме

Порядок следования байт в секторе отличен от порядка следования байт в F1-фрейме. То есть, попросту говоря, при отображении содержимого сектора на фрейм, четные и нечетные байты меняются местами (рис. 1.19).

Такое перемешивание призвано уменьшить пагубное влияние дефектов диска, затрагивающих оба соседних байта сразу.

Программная реализация разбиения сектора на фреймы приведена в листинге 1.8 (здесь: `sector` — указатель на исходный сектор, `F1_frames` — массив из 98 фреймов по 24 байта каждый).

Листинг 1.8. Пример, демонстрирующий технику формирования F1-фреймов (рассмотрен случай, когда границы фреймов и секторов совпадают)

```
/* generate F1 frames */
for (a = 0; a < 98; a++)
{
        for (b = 0; b < 24; b++)
        {

F1_frame[a][b]=((*sector&0xff00ff00UL)>>8)|((*sector&0x00ff00ffUL)<<8);
        }
}
```

F2-фрейм

Только что формированные F1-фреймы поступают на вход специального кодера (*Cross Interleaved Reed-Solomon Coder*, по своим первым буквам так же называемый *CIRC-кодером*), где к их 24 байтам добавляют еще 8 байт контрольной суммы Рида-Соломона, в результате чего на выходе кодера образуются F2-фреймы в 32 байта длиной.

Содержимое байт, слагающих F1-фрейм на битовом уровне, остается неизменным *("The bit pattern of each of the 24 8-bit bytes of an F1-Frame remains unchanged")*, но сами эти байты перераспределяются по 106 F2-фреймам, в результате чего F1-фреймы как бы "размазываются" вдоль спиральной дорожки, становясь менее чувствительными к радиальным царапинам диска (и вообще всяким локальным дефектам).

Перемешивание достигается за счет так называемых *линий задержки* (*delay line*) и осуществляется по следующей схеме (рис. 1.20). Первая линия задержки (first delay section) "заглатывает" поступающие на ее вход F1-фреймы, уже разбитые на 12 двухбайтовых слов, младший и старший байты которых условно обозначены литерами A и B соответственно, а сами слова последовательно пронумерованы от W12n до W12n+11. Таким образом, первый байт фрейма имеет номер "W12n,A", а последний — "W12n+11,B".

Первая линия задержки расщепляет содержимое фрейма на две группы слов, одна из которых (W12n+2, W12n+3, W12n+6, W12n+7, W12n+10, W12n+11) беспрепятственно проходит на выход, а другая (W12n+0, W12n+1,

W12n+4, W12n+5, W12n+8, W12n+9) принудительно удерживается на время обработки двух последующих F1-фреймов. Слова с номера W12n+1W12n+10 тщательно перемешиваются по строго определенной схеме, которую легче изобразить графически (см. рис. 1.20), чем описать словесно.

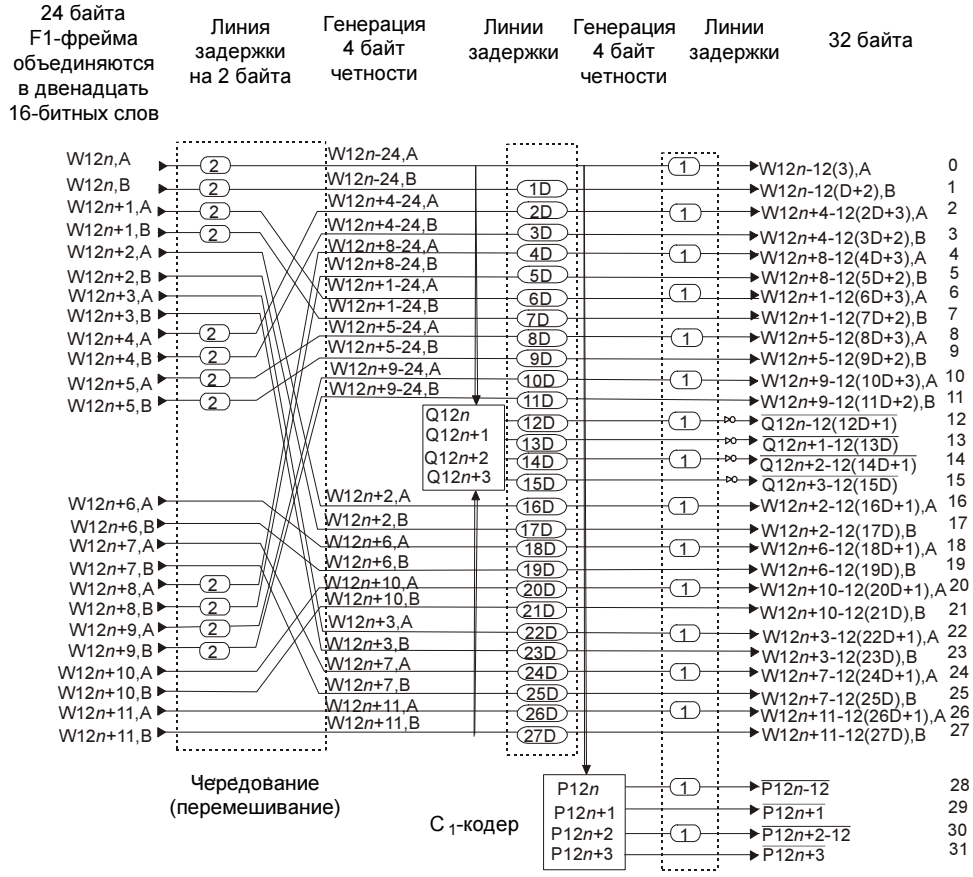


Рис. 1.20. Процесс кодирования байт

Перемешанные слова поступают на вход *C_1-кодера* (C1 encoder), где к ним добавляют четыре байта четности, рассчитанные по кодам Рида-Соломона и последовательно пронумерованные от Q12n+0 до Q12n+3.

Затем слова, обогащенные Q-байтами четности, попадают на вторую линию задержки (second delay lines), где они задерживаются на промежуток от 1D до 27D времени обработки F1-фрймов, где D равно 4.

Вышедшие "на свободу" слова направляются в очередной кодер Рида-Соломона, условно обозначаемый "C2", где к ним добавляют четыре байта

четности, последовательно пронумерованных от Pn+0 до Pn+3. Подобная двухуровневая схема избыточного кодирования значительно уменьшает вероятность возникновения неустранимых ошибок, ведь между C1 и C2 кодерами обрабатываемые данные хорошенько перемешиваются!

Наконец, последняя, третья линия задержки задерживает все четные байты потока данных на время обработки одного F1-фрейма. Все! С выхода третьей линии задержки сходит новехонький F2-фрейм, состоящий из 32 байт, последовательно пронумерованных от 0 до 32: 24 байт полезных данных, 4 Q-байт четности и 4 P-байта четности. Причем в 24 байта полезных данных F2-фрейма входят данные множества различных F1-фреймов! Другими словами говоря, F2-фрейм нельзя рассматривать как F1-фрем с контрольной суммой.

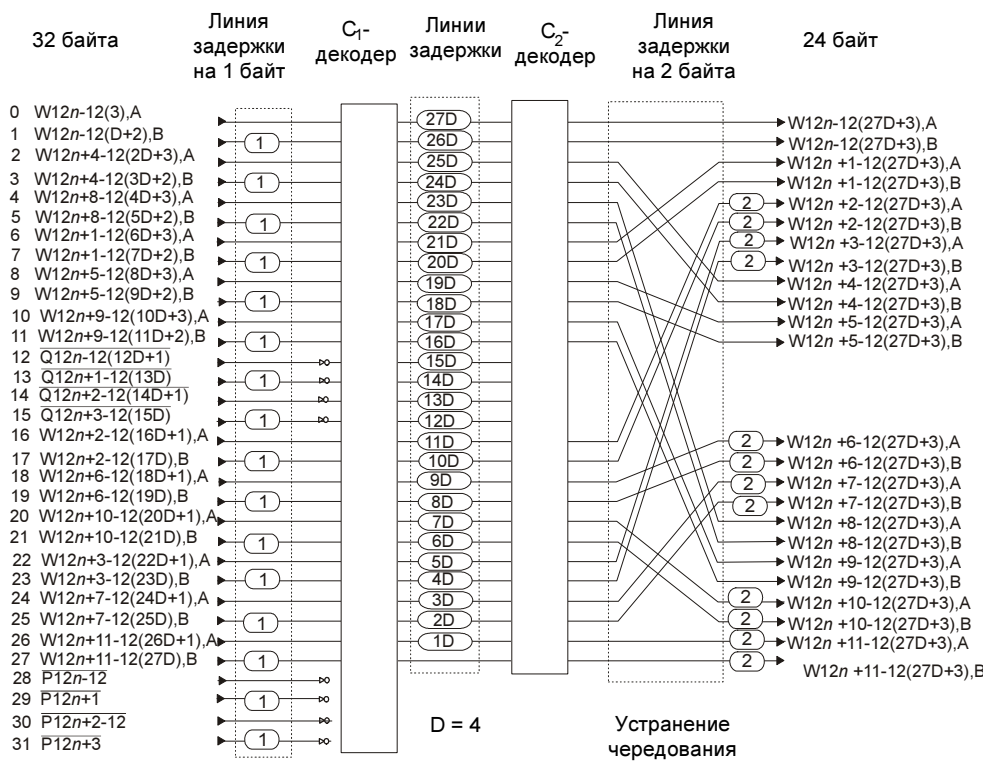


Рис. 1.21. Процесс декодирования байт

При считывании данных с лазерного диска происходит обратный процесс (рис. 1.21). Сначала считываемые байты проходят через линию задержки, "захватывающую" четные байты на один фрейм, затем они поступают в C1-декодер (C1 decoder), проверяющий истинность контрольной суммы и пы-

тающийся восстановить сбойные байты при необходимости. Далее следует еще одна линия задержки (1D-27D Delay lines) и еще один декодер (C2 decoder), восстанавливающий то, что не удалось восстановить еще предшественнику. Наконец, с выхода последней линии задержки сходят вполне готовые к употреблению F1-фреймы, которые впоследствии собираются в сектора, но о секторах мы уже говорили ранее, так что не будем повторяться.

При данной избыточности корректирующие коды в состоянии исправлять до двух сбойных байт на каждый 24/28-байтовый "ломтик" исходных данных. При искажении трех и более байт, декодер способен лишь констатировать сам факт искажения, но бессилен восстановить оригинальное содержимое разрушенных байт. Правда, установить, *какие именно* байты были разрушены, все-таки возможно, а значит, их приблизительное значение может быть определено путем интерполяции. Конечно, для дисков с данными такой способ "восстановления" не пригоден, но для Audio-CD он дает вполне удовлетворительный результат. Количество неисправимых ошибок даже на качественных носителях достаточно велико, и CD-приводам приходится активно заниматься их интерполяцией, о присутствии которой подавляющее большинство меломанов даже и не подозревают.

Компьютерные приводы CD-ROM первого поколения формально поддерживали аудиодиски и даже ухитрялись качественно проигрывать их, но при попытке цифрового "грабежа" диска в динамиках стоял сплошной треск, подобный тому, что издается поцарапанным винилом. Что ж, ностальгия — это прекрасно, но, черт возьми, время пластинок давно прошло и причина их стремительного вытеснения компакт-дисками не в последнюю очередь объясняется тем, что качество их звучания не падает с течением времени, в то время как пластинки неизбежно уродуются со временем. Теперь же оказывается, что приобретаемые нами компакт-диски *уже* изуродованные и *уже* "шипят".

Так происходит потому, что приводы CD-ROM первого поколения читали аудиодиски "как есть" и при возникновении неисправимых ошибок не осуществляли никаких попыток самостоятельного исправления сбойных байт. Более того, они не сообщали и номера этих байт, в результате чего прикладное программное обеспечение просто не знало, что и с чем следует интерполировать! Если искажение затрагивало младшие биты разряда, то для человеческого уха оно оставалось практически незаметным (даже для уха меломана), но если сбойный бит угодил в старший разряд, — такое искажение воспринималось человеком как резкий щелчок, даже если по его ушам потоптался китайский медведь панда. Вообще-то, сбойный сектор можно было попробовать перечитать несколько раз (вдруг с одной из попыток он почитается нормально?) или проанализировать считанные данные на предмет поиска и "сглаживания" всех крутых "пиков" и "спадов", но это же все полумера, вы понимаете! Качественный "грабеж" аудио на таких приводах

невозможен. К тому же некоторые производители дисков, озабоченные проблемой их несанкционированного копирования, стали умышленно вносить на них большое количество неисправимых ошибок, в результате чего в аудиорежиме диск читался нормально (даже на компьютерном приводе), но попытка его "грабежа", равно как и попытка копирования на болванку CD-R, терпела неудачу, ибо качество звучания было на редкость удручающим и едва ли могло соперничать с дедушкой-граммофоном.

Сейчас, однако, ситуация понемногу начала исправляться и некоторые из современных приводов CD-ROM уже способны возвращать в потоке данных указатели на искаженные биты. На программном уровне это достигается передачей приводу команды `BEh` (`READ CD`) с ненулевым значением поля "Error Flags" (оно расположено в 1—2 битах 9-го байта ATAPI/SCSI-пакета, см. также демонстрационный пример [/etc/RAW.CD.READ/aspi32.C2.c]). Подробнее об этом можно прочитать в Стандарте на DVD/CD-ROM-приводах (**http://www.stanford.edu/~csapuntz/specs/INF-8020.PDF** page 143), здесь же мы сосредоточимся не столько на описании формата полей команды `READ CD`, сколько на самих C2-указателях (C2-pointers), которые, строго говоря, никакими указателями не являются, а представляют собой обычную битовую карту, помещаемую в самый конец возвращаемых приводом данных. Каждому байту исходных данных соответствует "свой" бит C2-pointers, анализ значения которого и позволяет однозначно установить: является ли данный байт сбойным или нет. Учитывая, что длина сектора составляет 2352 байта, становится нетрудно рассчитать совокупный размер всех битов C2-pointers вместе взятых, который равен 2352/8 = 294 байтам, причем первый байт сектора соответствует первому биту C2-pointers (рис. 1.22).

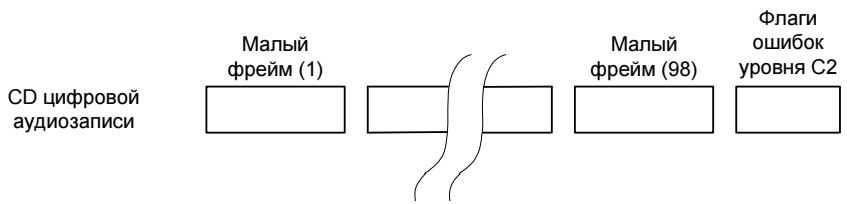


Рис. 1.22. Последовательность фреймов, образующая блок

Узнать, поддерживает ли данный конкретный привод такую возможность или нет, можно посылкой команды `MODE SENSE 10` (`5Ah`) с кодом страницы (Page Code), равным `2Ah` (C/DVD Capabilities and Mechanical Status Page Format), тогда единичное значение 4-го бита 13-го байта возвращенных данных укажет на то, что C2 Pointers is Supported и, соответственно, наоборот (см. также демонстрационный пример [/etc/RAW.CD.READ/aspi32.cfg.c]). В частности, мой PHILIPS CDRW 2400 ничего подобного, увы, не поддерживает.

Но довольно о грустном. Лучше вернемся к нашим C1- и C2-декодерам, а точнее, к методике подсчета количества ошибок. Существует как минимум шесть типов ошибок, по три для каждой из двух стадий исправления: а) *односимвольные (исправимые) ошибки*, соответствующие первой стадии восстановления (т. е. исправляемые декодером C1); б) двухсимвольные (исправимые) ошибки, соответствующие первой стадии исправления, и в) трехсимвольные (уже, увы, неисправимые) ошибки, соответствующие все той же стадии. Аналогичная картина наблюдается и на втором уровне восстановления, относящемуся к декодеру C2. Ошибки принято обозначать заглавной буквой "E" (лат. — от слова Error) с непосредственно примыкающим к ней двузначным числом, первая цифра которого обозначает число ошибок (1, 2 или 3), а вторая — стадию исправления (1 или 2). Для большей наглядности все возможные комбинации этих цифр сведены в таблицу (табл. 1.2).

Таблица 1.2. *Условное обозначение различных типов ошибок*

Обозначение	Пояснение
E11	Количество односимвольных (исправимых ошибок) на стадии C1
E21	Количество двухсимвольных (исправимых ошибок) на стадии C1
E31	Количество трехсимвольных (**неисправимых** ошибок) на стадии C1
E12	Количество односимвольных (исправимых ошибок) на стадии C2
E22	Количество двухсимвольных (исправимых ошибок) на стадии C2
E32	Количество трехсимвольных (**неисправимых** ошибок) на стадии C2

Трехсимвольные ошибки, неустранимые на стадии C1 (т. е. ошибки E31) в большинстве случаев могут быть успешно восстановлены на следующей стадии исправления, однако одна-единственная ошибка E31 может вызвать до 30 ошибок E12, ведь между C1- и C2-декодерами данные 160 F1-фреймов тщательно перемешиваются!

Трехсимвольные ошибки, неустранимые на стадии C2 (т. е. ошибки E32), свидетельствует о серьезном физическом дефекте поверхности диска, которые (в силу несовершенства технологических процессов) даже на "свежих" дисках отнюдь не так редки, как это может показаться на первый взгляд. Именно поэтому и приходится использовать дополнительные корректирующие коды на дисках с данными (для аудиодисков в этом случае используется интерполяция, но для данных интерполяция бессмысленна). Подробнее об этом можно прочесть в *разд. "«Сырые» и «сухие» сектора" этой главы*, здесь же мы не будет возвращаться к уже рассмотренным вопросам.

F3-фрейм

Когда к вышедшему из CIRC-декодера F2-фрейму добавляют еще один байт служебных данных, называемый *контрольным байтом* (*Control Byte*), образуется F3-фрейм, с которым мы уже сталкивались в *разд. "Кратко о питах, лендах, EFM-словах, фреймовых кадрах и секторах" этой главы.* Структура F3-фрейма до предела проста: первым идет контрольный байт, а за ним следуют 32 байта, доставшихся в наследство от F2-фрейма. Контрольный байт содержит в себе восемь бит подкода, которые в свою очередь образуют каналы, простирающиеся в длину на 98 байт, т. е. захватывающие весь сектор целиком, поскольку сектор как раз и состоит из **98** F3-фреймов (подробнее *см. разд. "Каналы подкода" этой главы*).

Структура, образованная 98 F3-фреймами, называется *секцией* (*section*) и представляет собой вполне самодостаточную сущность, не привязанную к границам сектора. Позвольте зачитать пару цитат из Стандарта ECMA-130: "These Sections are asynchronous with the Sectors, i.e. there is no prescribed relation between the number of the F1-Frame in which the first byte of a Sector is placed and the number of the F3 -Frame in which the first Control byte of the table is placed" *("Эти секции асинхронны по отношению к секторам, т. е. между номером F1-фрейма, хранящим первый байт сектора, и номером F3-фрейма, хранящим первый контрольный байт таблицы, нет жестко заданного отношения").* Так же: "The address of a Sector is recorded in the Sector Header, also as an absolute time. It has no prescribed relation to the addresses of the Sections, because the mapping of a Sector on the Sections during recording is implementation-dependent due to the freedom left in clause 16. Therefore, the address of a Sector is filled in just before the Sector enters the CIRC encoder" *("Адрес сектора, записанный в его заголовке, также выражается в абсолютном времени. Он не имеет никакого заранее заданного отношения к адресу секции, поскольку отображение секторов на секции, происходящее в процессе записи последних, есть implementation-dependent, то есть зависит от воли левого мизинца правой пятки разработчика записывающего привода, а если говорить серьезно, оно в значительной мере обусловлено свободой выбора стартовой позиции начала сектора в F1-фрейме. Поэтому адрес сектора заполняется непосредственно перед входом сектора на CIRC-кодер").*

Первые байты двух F3-фреймов каждой секции (т. е. первые два контрольных байта секции) обрабатываются особым образом. В то время как остальные байты секции (коих насчитывается 3 232) преобразуются в 14-битовые EFM-последовательности, непосредственно записываемые на диск, эти два "товарища" замещаются фиксированными синхрогруппами, именуемые *SYNC0* (00100000000001) и *SYNC1* (00000000010010) соответственно (рис. 1.23).

№ байта	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
	p	q	r	s	t	u	v	w
1	SYNC 0							
2	SYNC 1							
. . . 96 97								

Рис. 1.23. Устройство секции

Программная, вводная и выводная области, оглавление диска и область данных

Последовательность секторов одного формата объединяется в *дорожку*, иначе *трек (track)*, минимально возможная длина которой составляет 300 секторов, а максимальная — весь диск целиком. Первый и последний треки диска, т. е. вводная (Lead-In) и выводная (Lead-Out) области соответственно, используются для служебных целей, хотя большинство современных приводов способны обходиться и без них (а пишущие приводы это делать вообще обязаны).

Вводная область диска — (Lead-In area). Служебная область диска (рис. 1.24), по сути своей представляющая нулевой трек, всегда предшествующий первому треку PMA *(Program Memory Area)*. Каждая сессия многосессионного диска имеет собственную вводную область. Размер вводной области по стандарту составляет 9 Мбайт (60 секунд или 4500 секторов). Q-канал подкода вводной сессии содержит оглавление диска (TOC), среди прочей полезной информации указывающей либо на адрес выводной области (закрытый диск), либо на адрес вводной области следующей сессии (открытый диск). Содержимое вводной области недоступно для чтения на программном уровне (доступно у приводов MSI). Визуально вводная область выглядит равномерно освещенным блестящим кольцом.

Внимание

Не всякое блестящее кольцо — это область Lead-In! Настоящая область Lead-In всегда находится на расстоянии 23 мм от края диска, а перед ней идет еще всякий "мусор".

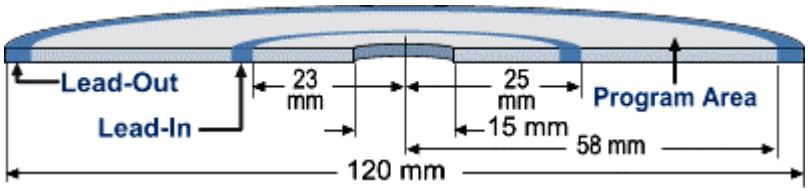


Рис. 1.24. Строение лазерного диска

Выводная область диска — (Lead-Out area). Служебная область диска, условно обозначаемая треком номер `AAh` и замыкающая собой всякую закрытую сессию. Выводная область служит своеобразным индикатором конца сессии и/или диска и помогает оптической головке не вылететь за пределы диска. Пишущие приводы должны корректно обрабатывать диски с незакрытыми сессиями, однако обыкновенные приводы CD-ROM и аудиопроигрыватели это делать не обязаны.

Внимание

Отсутствие выводной сессии (равно как и некорректное задание ее адреса) может повредить некоторые модели приводов (один из них PHILIPS).

Емкость выводной области односессионного диска по стандарту составляет 13,5 Мбайт (6750 секторов или 1,5 минуты). Емкость выводных областей для второй и последующих сессий многосессионных дисков уменьшена до 4 Мбайт (0,5 минуты или 2250 секторов). Содержимое выводной области недоступно на программном уровне (доступно у приводов MSI). Визуально выводная область выглядит равномерно освещенным блестящим кольцом.

Оглавление диска или иначе *таблица содержимого* (TOC — *Table Of Content*). Служебная область диска, записанная в Q-канале подкода вводной области диска, также называемой областью Lead-In (такое блестящее кольцо у внутреннего края диска). Многосессионный диск имеет несколько независимых TOC — по одному TOC на каждую закрытую сессию.

TOC содержит информацию о стартовых адресах вводной/выводной областей диска и атрибуты всех его треков (как-то тип трека: аудио или данные, а если данные, то в каком режиме — Mode 1, Mode 2 и т. д., абсолютном стартовом адресе трека и номере соответствующей ему сессии). Также TOC содержит часть ATIP и указатели на местоположения ее продолжения.

Непосредственно (т. е. на секторном уровне) для чтения TOC недоступен, но для извлечения его содержимого в "сыром" виде можно воспользоваться следующей SCSI/ATAPI командной `READ TOC/PMA/ATIP` (операционный код: 43h) с `format field == 2h`.

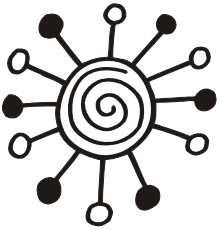
Однако не стоит путать TOC с файловой системой — между ними нет ничего общего! Файловые системы лазерных дисков хранятся непосредственно в PMA и свободно доступны для чтения на секторном уровне.

Программная область — (Program area или Program Memory Area (PMA)). Область диска, расположенная между областями Lead-In и Lead-Out и содержащая информационные треки с музыкой или данными. Это основная область диска, целиком доступная на секторном уровне с паузами между аудиотреками включительно. Подавляющее большинство CD-данных (Data-CD) содержат один-единственный трек данных, хранящий в себе всю необходимую информацию, записанную в той или иной файловой системе. Впрочем, файловые системы лежат за гранью темы нашего разговора. Что же касается аудио CD (Audio-CD), то никакой файловой системы они не имеют, а используют для этой цели TOC, помещая каждую песню в отдельный трек.

Если за областью Lead-Out располагается область Lead-In, то такой диск называется *многосессионным* (*multi-session*). Каждая *закрытая* сессия имеет собственные области Lead-In, Lead-Out и TOC, причем указатель на выводную область, находящийся в TOC, может содержать как действительный адрес выводной области текущей сессии, так и адрес вводной области следующей сессии! Количество сессий в принципе не ограничено, однако, в силу сквозной нумерации треков, количество сессий не может превышать 99. Сессия может быть независимой (TOC указывает только на треки внутри сессии) или связанной (TOC содержит адреса треков из предыдущих сессий). Однако далеко не все приводы "знают" о существовании сессий. В частности, подавляющее большинство Audio-проигрывателей "видят" только первую сессию диска и игнорируют все остальные. Благодаря этому обстоятельству существует возможность создания дисков, не читающихся на компьютерных приводах CD-ROM, но нормально "перевариваемых" CD-плеерами.

Сессия называется *закрытой*, если ее область данных обрамлена вводной и выводной областью. Незакрытые сессии могут читаться только устройствами записи (необходим доступ к PMA). Указатель в TOC сессии на выводную область может содержать либо действительно адрес выводной области данной сессии (закрытый диск), либо адрес вводной области следующей сессии. Запись ограничивается местом на диске, местом в PMA и числом треков (треки имеют сквозную нумерацию по всему диску от 01 до 99). Сессии могут быть связаны также на уровне файловой системы. Механизм сессий позволяет "изменять" информацию на болванке CD-R, дописывая новую сессию.

Глава 2



Могущество кодов Рида-Соломона или информация, воскресшая из пепла

Энтропия слепа, но терпелива. Рано или поздно, обстреливая наши позиции по квадратам, она нанесет удар по штабу, по центру связи. И тогда первая линия обороны будет уничтожена. И приходится отходить на запасные позиции. Иными словами, доставать из магнитотеки пакет дисков с копией тома.

Е. В. Лишак
"Тридцать второй день года.
(Записки парасистемного программиста)"

Все вы наверняка слышали о существовании помехозащитных кодов Рида-Соломона, широко использующихся в устройствах передачи и хранения данных для обнаружения и исправления как одиночных, так и групповых (!) ошибок. Область их применения необычайно широка — кодеры/декодеры Рида-Соломона можно найти и в ленточных запоминающих устройствах, и контроллерах оперативной памяти, и в модемах, и в жестких дисках, и в приводах CD-ROM/DVD и т. д. Благодаря им некоторые "продвинутые" архиваторы безболезненно переносят порчу нескольких секторов носителя, содержащего архив, а подчас — и полное разрушение целого тома многотомного архива. Еще коды Рида-Соломона позволяют защитному механизму автоматически восстанавливать байтики, "хакнутые" взломщиком и/или искаженные в результате сбоя программного/аппаратного обеспечения.

Короче говоря, если владение техникой помехозащитного кодирования не превращает вас в бога, то, по крайней мере, поднимает на Олимп, где среди бесшумных вентиляторов и "безглючных" операционных систем снуют великие компьютерные Гуру.

В то же время, лишь немногие программисты могут похвастаться собственной реализацией алгоритмов Рида-Соломона. Да и зачем? Готовых библиотек

море — от прагматичных коммерческих пакетов, до бесплатных "исходников", распространяемых по лицензии GNU. Как говорится, бери — не хочу.

Замечание

"…из-за ошибок в реализации данный код вместо исправления ошибок добавляет новые. Поэтому данный код больше недоступен" — комментарий к GNU исходным текстам кодера/декодера Рида-Соломона (Reed-Solomon). Вот и верь после этого в надежность Linux в целом и в "GNU-тый" библиотечный код в частности.

Что ж, в использовании библиотек есть вполне определенный практический смысл, но никакой хакер не доверит управление программе до тех пор, пока не поймет, *как именно* она работает (а эта публикация именно для хакеров и предназначена, естественно, "хакеров" в хорошем значении этого слова). С другой стороны, при анализе программного обеспечения, распространяемого без исходных кодов, вы не сможете идентифицировать алгоритм Рида-Соломона, если только заранее не разберетесь во всех его тонкостях. Допустим, вам встретилась защита, хитрым образом манипулирующая с EDC/ECC-полями ключевых секторов, считанных ею с лазерного диска, и каждый такой сектор содержит две умышленно внесенные ошибки (плюс еще ошибки, естественным путем возникающие при небрежном обращении с CD), причем одна из этих ошибок *ложная* и исправлять ее *не нужно*. При штатном копировании защищенного диска микропроцессорная начинка привода CD-ROM автоматически исправляет все ошибки, которые она только может исправить, в результате чего происходит искажение ключевых меток и — как следствие, — защищенная программа перестанет работать. Можно, конечно, скопировать диск в "сыром" режиме, т. е. без исправления ошибок, но тогда копия будет содержать как не предумышленные, так и предумышленные ошибки, в результате чего даже при незначительном повреждении оригинала, корректирующих возможностей кодов Рида-Соломона уже окажется недостаточно и диск просто перестанет читаться (А как вы хотели? Копирование дисков в сыром режиме ведет к *накоплению* ошибок и потому крайне непрактично с любой точки зрения.). Владение базовыми принципами помехозащитного кодирования позволит вам разобраться с логикой работы защитного механизма и понять, какие конкретно ошибки следует исправлять, а какие нет.

К сожалению, подавляющее большинство публикаций на тему кодов Рида-Соломона написаны на языке высшей математики, для постижения которой и университетских знаний подчас оказывается недостаточно (да и все ли хакеры знают математику?), в результате чего все эти сильно теоретизированные руководства забрасываются на полку, если вообще не отправляются в туалет (не то, чтобы я разделял такие намерения, но все-таки).

Программная реализация корректирующих кодов Рида-Соломона действительно очень сложна и на самом деле требует определенной математической подготовки, изложение основ которой может показаться скучным и неинтересным для "системщиков" и "железячников", но иного пути, по-видимому,

нет. В конце концов, никто не обещал вам, что быть программистом — легко, а хорошим программистом быть еще труднее. Так что не говорите потом, что я вас не предупреждал! Шутка! Расслабьтесь и отгоните свой страх перед высшей математикой прочь. По ходу описания вам повстречается пара формул (ну куда же в математике без формул?), но во всех остальных случаях я буду говорить на интернациональном программистском языке — языке Си, понятном любому системщику. В общем, пристегивайте ремни и поднимайте свои головы с клавиатуры, — мы поехали!

Корректирующие коды и помехоустойчивое кодирование

Персональные компьютеры с их битами и байтами настолько прочно вошли в нашу жизнь, что программисты вообще перестали задумываться о теории кодирования информации, принимая ее как должное. Между тем, здесь все не так просто, как может показаться на первый взгляд.

Фактически, кодирование есть не что иное, как преобразование *сообщения* в последовательность *кодовых символов*, также называемых *кодовыми словами*. Любое *дискретное* сообщение состоит из конечного числа элементов: в частности, текст состоит из букв, изображение состоит из пикселов, машинная программа состоит из команд и т. д., — все они образуют *алфавит источника сообщения*. При кодировании происходит преобразование элементов сообщения в соответствующие им числа — *кодовые символы*, причем каждому элементу сообщения присваивается уникальная совокупность кодовых символов, называемая *кодовой комбинацией*. Совокупность кодовых комбинаций, образующих сообщение, и есть *код*. Множество возможных кодовых символов называется *кодовым алфавитом*, а их количество (далее по тексту обозначаемое малой латинской *m*) — *основанием кода*.

Впрочем, все это вы уже наверняка знаете (а если не знаете, то без труда найдете исчерпывающее объяснение основ кодирования в любом учебнике по информатике), но знаете ли вы, что такое *расстояние Хемминга*? Это — *минимальное количество различий между двумя различными допустимыми кодовыми словами*, и в теории помехоустойчивого кодирования расстояние Хемминга играет основополагающую роль. Рассмотрим, например, следующий четырехбитный код (листинг 2.1).

Листинг 2.1. Пример простейшего четырехбитного кода с расстоянием Хемминга, равным единице

```
0 → 0000;    4 → 0100;    8 → 1000;    12 → 1100;
1 → 0001;    5 → 0101;    9 → 1001;    13 → 1101;
2 → 0010;    6 → 0110;   10 → 1010;    14 → 1110;
3 → 0011;    7 → 0111;   11 → 1011;    15 → 1111;
```

Такой код широко используется в вычислительной технике, несмотря на его невозможность обнаружить ошибки.

Это обыкновенный двоичный код, который можно встретить в некоторых "однокристалках", вмещающий в свои 4 бита 16 символов (т. е. с его помощью можно закодировать 16 букв алфавита). Как нетрудно убедиться, что два любых символа этого кода отличаются, по меньшей мере, на один бит, следовательно, расстояние Хемминга для такого кода равно единице (что условно обозначает как $d = 1$).

А вот другой четырехбитный код с расстоянием Хемминга, равным двум, способный обнаруживать одиночные ошибки (листинг 2.2).

Листинг 2.1. Пример четырехбитного кода с расстоянием Хемминга, равным двум

```
0 → 0000;    4 → 1001;
1 → 0011;    5 → 1010;
2 → 0101;    6 → 1100;
3 → 0110;    7 → 1111;
```

На этот раз, два произвольных символа отличаются как минимум в двух позициях, за счет чего информационная емкость такого кода сократилась с 16 до 8 символов. "Постойте-постойте! — воскликнет иной читатель. — Что это за бред? Куда девалась комбинация 0001 или 0010 например?" Нет, это не бред и указанных комбинаций бит в данном коде действительно *нет*, точнее, они есть, но объявлены *запрещенными*. Благодаря этому обстоятельству наш подопечный код способен обнаруживать любые одиночные ошибки. Возьмем, например, символ "1010" и исказим в нем произвольный бит (но только один!). Пусть это будет второй слева бит, — тогда искаженный символ станет выглядеть так: "1$\boxed{1}$10". Поскольку комбинация "1110" является запрещенной, декодер может засвидетельствовать наличие ошибки. Увы, только засвидетельствовать, но не исправить, т. к. для исправления даже одного-единственного сбойного байта требуется увеличить расстояние Хемминга как минимум до трех. Поскольку 4-битный код с $d = 3$ способен вмещать в себя лишь два различных символа, то он крайне ненагляден, и потому нам лучше выбрать код с большей разрядностью. Хорошо, пусть это будет 10-битный код с $d = 5$ (листинг 2.3).

Листинг 2.2. Пример 10-битного кода, с расстоянием Хемминга, равным пяти, способного обнаруживать четырехбитные ошибки, а исправлять — двухбитовые

```
0000000000    0000011111    1111100000    1111111111
```

Возьмем, к примеру, символ `0000011111` и изменим два его любых бита, получив в итоге что-то наподобие: `0100110111`. Поскольку такая комбинация является запрещенной, декодер понимает, что произошла ошибка. Достаточно очевидно, что если количество сбойных бит меньше расстояния Хемминга хотя бы наполовину, то декодер может гарантированно восстановить исходный символ. Действительно, если между двумя любыми разрешенными символами существует не менее пяти различий, то искажение двух бит всякого такого символа приведет к образованию нового символа (обозначим его k), причем расстояние Хемминга между k и оригинальным символом равно числу непосредственно искаженных бит (т. е. в нашем случае двум), а расстояние до ближайшего соседнего символа равно: $d - k$ (т. е. в нашем случае трем). Другими словами, пока $d - k > k$, декодер может гарантированно восстановить искаженный символ. В тех случаях, когда $d > k > d - k$, успешное восстановление уже не гарантируется, но при удачном стечении обстоятельств все-таки оказывается в принципе возможным.

Возвращаясь к нашему символу `0000011111`, давайте на этот раз исказим не два бита, а четыре: `0100110101` и попробуем его восстановить. Изобразим процесс восстановления графически (листинг 2.4).

Листинг 2.3. Восстановление четырехбитной ошибки

```
0000000000    0000011111    1111100000    1111111111
0100110101    0100110101    0100110101    0100110101
----------    ----------    ----------    ----------
5 отличий     4 отличия     6 отличий     5 отличий
```

Грубо говоря, обнаружив ошибку, декодер последовательно сличает искаженный символ со всеми разрешенными символами алфавита, стремясь найти символ, наиболее "похожий" на искаженный. Точнее — символ с наименьшим числом различий, а еще точнее — символ, отличающийся от измененного не более чем в $(d - 1)$ позициях. Легко увидеть, что в данном случае нам повезло и восстановленный символ совпал с истинным. Однако если бы четыре искаженных бита распределились бы так: `0111111111`, то декодер принял бы этот символ за `1111111111` и восстановление оказалось бы неверным.

Таким образом, исправляющая способность кода определяется по следующей формуле: для обнаружения r ошибок расстояние Хемминга должно быть больше или равно r, а для коррекции r ошибок, расстояние Хемминга должно быть, по крайней мере, на единицу больше удвоенного количества r (листинг 2.5).

Листинг 2.4. Корректирующие способности простого кода Хемминга

```
обнаружение      ошибок: d >= r
исправление      ошибок: d > 2r
информационная емкость: 2^(n/d)
```

Теоретически количество обнаруживаемых ошибок не ограничено, практически же информационная емкость кодовых слов стремительно тает с ростом d. Допустим, у нас есть 24 байта данных и мы хотели бы исправлять до двух ошибок на каждый такой блок. Тогда нам придется добавить к этому блоку еще 49 байт, в результате чего реальная информационная емкость блока сократится всего до 30%! Хорошенькая перспектива, не так ли? Столь плачевный результат объясняется тем, что биты кодового слова изолированы друг от друга и изменение одного из них никак не сказывается на окружающих. А что если...

Пусть все биты, номера которых есть степень двойки, станут играть роль контрольных битов, а оставшиеся и будут обычными ("информационными") битами сообщения. Каждый контрольный бит должен отвечать за четность суммы (т. е. если сумма проверяемых бит — четна, то контрольный бит будет равен нулю и, соответственно, наоборот) некоторой, принадлежащей ему группы битов, причем один и тот же информационный бит может относиться к различным группам. Тогда один информационный бит сможет влиять на несколько контрольных и потому информационная емкость слова значительно (можно даже сказать, чудовищно) возрастет. Остается только выбрать наиболее оптимальное разделение сфер влияния.

Согласно методу помехозащитного кодирования, предложенного Хеммингом, для того чтобы определить, какие контрольные биты контролируют информационный бит, стоящий в позиции k, мы должны разложить k по степеням двойки, как это показано в таблице 2.1.

Таблица 2.1. *Разделение бит на контрольные и информационные*

Позиция	Какими битами контролируется	
1 (A)	$2^0 = 1$	Это контрольный бит, никто его не контролирует
2 (B)	$2^1 = 2$	Это контрольный бит, никто его не контролирует
3	$2^0+2^1 = 1 + 2 = 3$	Контролируется 1 и 2 контрольными битами
4 (C)	$2^2 = 4$	Это контрольный бит, никто его не контролирует
5	$2^0+2^2 = 1 + 4 = 5$	Контролируется 1 и 4 контрольными битами
6	$2^1+2^2 = 2 + 4 = 6$	Контролируется 2 и 4 контрольными битами
7	$2^0+2^1+2^2 = 1 + 2 + 4 = 7$	Контролируется 1, 2 и 4 контрольными битами
8 (D)	$2^3 = 8$	Это контрольный бит, никто его не контролирует

Давайте в порядке закрепления материала попробуем "пощупать" коды Хемминга в "живую" и вручную рассчитаем контрольную сумму 4-битного символа "0101". После резервирования "квартир" для контрольных битов (выделенных в тексте жирным шрифтом) наш символ будет выглядеть так: **AB**0**C**101**D**. Теперь остается только рассчитать значения битов A, B, C и D.

- Бит A, контролирующий биты 3, 5 и 7, равен *нулю*, т. к. их сумма (0 + 1 + 1) четна.
- Бит B, контролирующий биты 3, 6 и 7, равен *единице*, т. к. их сумма (0 + 0 + 1) нечетна.
- Бит C, контролирующий биты 5, 6 и 7, равен *нулю*, т. к. их сумма (1 + 0 + 1) четна.

Таким образом, "новоиспеченное" кодовое слово будет выглядеть так: "**01**0**0**101", где жирным шрифтом выделены контрольные биты (листинг 2.6).

Листинг 2.5. Кодовое слово вместе с информационными битами

```
AB0C101
1234567
```

Допустим, при передаче наше слово было искажено в одной позиции и стало выглядеть так: **01**0**0**1$\boxed{1}$1. Сможем ли мы обнаружить такую ошибку? А вот сейчас и проверим! Так, бит A должен быть равен: (0 + 1 + 1) % 2 = 0, что соответствует истине. Бит B должен быть равен (0 + 1 + 1) % 2 = 0, а в нашем слове он равен единице. Запомним номер "неправильного" контрольного бита и продолжим. Бит C должен быть равен (1 + 1 + 1) % 2 = 1, а он равен нулю! Ага, значит, контрольные биты в позициях 2 (бит B) и 4 (бит C) обнаруживают расхождение с действительностью. Их сумма (2 + 4 = 6) и дает позицию сбойного бита. Действительно, в данном случае номер искаженного бита будет равен 6, — инвертируем его, тем самым, восстанавливая наше кодовое слово в исходный вид.

А что, если искажение затронет не информационный, а контрольный бит? Проверка показывает, что позиция ошибки успешно обнаруживается и в этом случае и контрольный бит при желании может быть легко восстановлен по методике, уже описанной выше (только есть ли в этом смысл? ведь контрольные биты все равно выкусываются в процессе декодирования кодового слова).

На первый взгляд кажется, что коды Хемминга жутко неэффективны, ведь на 4 информационных бита у нас приходится 3 контрольных, однако поскольку номера контрольных бит представляют собой степень двойки, то с ростом разрядности кодового слова они начинают располагаться все реже и реже. Так, ближайший к биту C контрольный бит D находится в позиции 8

(т. е. в "трех шагах"), зато контрольный бит E отделен от бита D уже на $2^4 - 2^3 - 1 = 7$ "шагов", а контрольный бит F и вовсе — на $2^5 - 2^4 - 1 = 15$ "шагов".

Таким образом, с увеличением разрядности обрабатываемого блока, эффективность кодов Хемминга стремительно нарастает, что и показывает следующая программа (листинг 2.7) и результаты расчетов, выполненные с ее помощью (листинг 2.8).

Листинг 2.6. Расчет эффективной информационной емкости кодов Хемминга для слов различной длины

```
main()
{
    int a;
    int _pow = 1;
    int old_pow = 1;
    int N, old_N = 1;

    printf( "* * * hamming code efficiency test * * * by Kris Kaspersky\n"\
                  " BLOCK_SIZE    FUEL UP   EFFICIENCY\n"\
                  "-----------------------------------\n");
    for (a = 0; a < MAX_POW; a++)
    {
           N = _pow - old_pow - 1 + old_N;

           printf("%8d   %8d   %8.1f%%\n",_pow, N, (float) N/_pow*100);

           // NEXT
           old_pow = _pow; _pow = _pow * 2; old_N = N;

    } printf("-----------------------------------\n");
}
```

Листинг 2.7. Результат расчета эффективной информационной емкости кодов Хемминга для слов различной длины

```
BLOCK_SIZE    FUEL UP   EFFICIENCY
-----------------------------------
       1          0        0.0%
       2          0        0.0%
       4          1       25.0%
       8          4       50.0%
```

```
      16          11        68.8%
      32          26        81.3%
      64          57        89.1%
     128         120        93.8%
     256         247        96.5%
     512         502        98.0%
    1024        1013        98.9%
    2048        2036        99.4%
    4096        4083        99.7%
    8192        8178        99.8%
   16384       16369        99.9%
   32768       32752       100.0%
   65536       65519       100.0%
  131072      131054       100.0%
  262144      262125       100.0%
  524288      524268       100.0%
-----------------------------------
```

Из приведенной распечатки (см. листинг 2.8) видно, что при обработке блоков, "дотягивающихся" хотя бы до 1024 бит, накладными расходами на контрольные биты можно полностью пренебречь.

К сожалению, коды Хемминга способны исправлять лишь одиночные ошибки, т. е. допускают искажение всего лишь одного сбойного бита на весь обрабатываемый блок. Естественно, с ростом размеров обрабатываемых блоков увеличивается и вероятность ошибок. Поэтому выбор оптимальной длины кодового слова является весьма нетривиальной задачей, как минимум требующей знания характера и частоты возникновения ошибок используемых каналов передачи информации. В частности, для ленточных накопителей, лазерных дисков, винчестеров и тому подобных устройств, коды Хемминга оказываются чрезвычайно неэффективными. Зачем же тогда мы их рассматривали? А затем, что понять прогрессивные системы кодирования (к которым в том числе относятся и коды Рида-Соломона), ринувшись атаковать их "с нуля", практически невозможно, ибо они завязаны на сложной, действительно высшей математике, но ведь не Боги горшки обжигают, верно?

Идея кодов Рида-Соломона

Если говорить упрощенно, то основная идея помехозащитного кодирования Рида-Соломона заключается в умножении информационного слова, представленного в виде полинома D, на неприводимый полином G (т. е. такой полином, который не разлагается в произведение полиномов меньшей степени), известный обоим сторонам, в результате чего получается кодовое слово C, опять-таки представленное в виде полинома.

Декодирование осуществляется с точностью до наоборот: если при делении кодового слова C на полином G декодер внезапно получает остаток, то он может "рапортовать наверх" об ошибке. Соответственно, если кодовое слово разделилось нацело, то его передача завершилась успешно.

Если степень полинома G (называемого также *порождающим полиномом*) превосходит степень кодового слова по меньшей мере на две степени, то декодер может не только обнаруживать, но и исправлять одиночные ошибки. Если же превосходство степени порождающего полинома над кодовым словом равно четырем, то восстановлению поддаются и двойные ошибки. Короче говоря, степень полинома k связана с максимальным количеством исправляемых ошибок t следующим образом: $k = 2t$. Следовательно, кодовое слово должно содержать два дополнительных символа на одну исправляемую ошибку. В то же время, максимальное количество распознаваемых ошибок равно t, т. е. избыточность составляет один символ на каждую распознаваемую ошибку.

В отличие от кодов Хемминга, коды Рида-Соломона могут исправлять любое разумное количество ошибок при вполне приемлемом уровне избыточности. Спрашиваете, за счет чего это достигается? Смотрите, в кодах Хемминга контрольные биты контролировали лишь те информационные биты, что находятся по правую сторону от них, и игнорировали всех "левосторонних товарищей". Обратимся к таблице 2.1, — добавление восьмого контрольного бита D ничуть не улучшило помехозащищенность кодирования, поскольку контрольному биту D было некого контролировать. В кодах же Рида-Соломона контрольные биты распространяют свое влияние на все информационные биты и потому, с увеличением количества контрольных бит, увеличивается и количество распознаваемых/устраняемых ошибок. Именно благодаря последнему обстоятельству, собственно, и вызвана ошеломляющая популярность корректирующих кодов Рида-Соломона.

Теперь о грустном. Для работы с кодами Рида-Соломона обычная арифметика, увы, не подходит и вот почему. Кодирование предполагает вычисления по правилам действия над многочленами, с коэффициентами которых надо выполнять операции сложения, вычитания, умножения и деления, причем все эти действия не должны сопровождаться каким-либо округлением промежуточных результатов (даже при делении!), чтобы не вносить неопределенность. Причем и промежуточные, и конечные результаты не имеют права выходить за пределы установленной разрядной сетки. "Постой! — воскликнет внимательный читатель. — Да ведь это невозможно! Чтобы при умножении и не происходило "раздувания" результатов, — кто же в этот бред поверит?!"

Впрочем, если как следует подумать головой, частично призвав на помощь и другие части тела, можно сообразить, что умножать информационное слово

на порождающий полином вовсе и не обязательно, можно поступить гораздо хитрее:

1. Добавляем к исходному информационному слову D справа k нулей, в результате чего у нас получается слово длиной $n = m + r$ и полином $X^{r*}D$, где m — длина информационного слова.
2. Делим полученный полином $X^{r*}D$ на порождающий полином G и вычисляем остаток от деления R, такой, что: $X^{r*}D = G*Q + R$, где Q — частное, которое мы благополучно игнорируем за ненадобностью, — сейчас нас интересует только остаток.
3. Добавляем остаток R к информационному слову D, в результате чего получаем "симпатичное" кодовое слово C, *информационные биты которого хранятся отдельно от контрольных бит*. Собственно, *тот остаток, который мы получили в результате деления, — и есть корректирующие коды Рида-Соломона*. Между нами говоря, способ кодирования, при котором информационные и контрольные символы хранятся раздельно, называется *систематическим кодированием*, и такое кодирование весьма удобно с точки зрения аппаратной реализации.
4. Мысленно прокручиваем пункты 1, 2 и 3, пытаясь обнаружить, на какой же стадии вычислений происходит выход за разрядную сетку, и такой стадии нет! Все нормально! Остается лишь отметить, что информационное слово плюс корректирующие коды можно записать как: `T == Xr*D + R = GQ`.

Декодирование полученного слова T осуществляется точно так же, как уже и было описано ранее. Если при делении слова T (которое в действительности является произведением G на Q) на порождающий полином G образуется остаток, то слово T искажено и, соответственно, наоборот.

Теперь — вопрос на засыпку. Как вы собираетесь осуществлять деление полиномов в рамках общепринятой алгебры? В целочисленной арифметике деление определено не для всех пар чисел (вот, в частности, 2 нельзя разделить на 3, а 9 нельзя разделить на 4, — без потери значимости естественно). Что же касается "плавучки", — то ее точность еще та (в смысле точность катастрофически недостаточная для эффективного использования кодов Рида-Соломона), к тому же она довольно сложна в аппаратной реализации. Ладно, если вы работаете с IBM PC с процессором Pentium, который оснащен быстродействующим математическим сопроцессором, но что делать разработчикам ленточных накопителей, винчестеров, CD-приводов наконец? Использовать в них процессор Pentium 4! Нет уж, увольте, — лучше воспользоваться специальной арифметикой, — *арифметикой конечных групп*, называемых *полями Галуа*. Достоинство этой арифметики в том, что операции сложения, вычитания, умножения и деления определены для всех членов поля (естественно, исключая ситуацию деления на ноль), причем число, полученное в результате любой из этих операций, обязательно присутствует в группе! Таким образом, при делении любого целого числа A,

принадлежащего множеству 0255, на любое целое число B из того же множества (естественно, B не должно быть равно нулю), мы получим число C, входящее в данное множество. А поэтому потерь значимости не происходит и никакой неопределенности не возникает!

Таким образом, корректирующие коды Рида-Соломона основаны на полиномиальных операциях в полях Галуа и требуют от программиста владения сразу несколькими аспектами высшей математики из раздела теории чисел. Как и все "высшее", придуманное математиками, поля Галуа есть суть абстракция, которую невозможно ни наглядно представить, ни "пощупать" руками. Ее надо просто принять как набор аксиом, не пытаясь вникнуть в смысл, достаточно всего лишь знать, что она работает — вот и все. А еще есть полиномы "немерянных" степеней и матрицы в "пол-Европы", от которых нормального системщика, извините, тошнит (увы, программист-математик скорее исключение, чем правило).

Поэтому прежде чем ринуться в непроходимые джунгли математического леса абстракций, давайте сконструируем макет кодера/декодера Рида-Соломона, работающий по правилам обычной целочисленной алгебры. Естественно, за счет неизбежного в этом случае расширения разрядной сетки, такому кодеру/декодеру будет очень трудно найти практическое применение, но зато он нагляден и позволяет не только понять, но и почувствовать принцип работы корректирующих кодов Рида-Соломона.

Мы будем исходить из того, что если $g = 2^n + 1$, то для любого a из диапазона 02^n произведение $a*g = c$ (где c — кодовое слово) будет представлять по сути полную мешанину битов обоих исходных чисел.

Допустим, $n = 2$, тогда $g = 3$. Легко видеть: на что бы мы не умножали g — хоть на 0, хоть на 1, хоть на 2, хоть на 3, полученный результат делится нацело на g в том и только в том случае, если никакой из его битов не инвертирован (т. е., попросту говоря, одиночные ошибки отсутствуют).

Остаток от деления однозначно указывает на позицию ошибки (при условии, что ошибка одиночная, групповые же ошибки данный алгоритм исправлять не способен). Точнее, если ошибка произошла в позиции x, то остаток от деления k будет равен $k = 2^x$. Для быстрого определения x по k можно воспользоваться тривиальным табличным алгоритмом. Впрочем, для восстановления сбойного бита знать его позицию совершенно необязательно, достаточно сделать $R = e \wedge k$, где e — искаженное кодовое слово, ^ — операция `XOR`, а R — восстановленное кодовое слово.

В общем, законченная реализация кодера/декодера Рида-Соломона, работающего по обычной арифметике (т. е. с неоправданным расширением разрядной сетки) и исправляющим любые одиночные ошибки в одном 8-битном информационном слове (впрочем, программу легко адаптировать и под 16-байтовые информационные слова), может выглядеть так, как показано в листинге 2.9. Обратите внимание, что кодер реализуется чуть ли не

на порядок проще декодера. В настоящем декодере Рида-Соломна, способном исправлять групповые ошибки, этот разрыв еще значительнее.

Листинг 2.9. Простейший пример реализации кодера/декодера Рида-Соломона

```
/*-----------------------------------------------------------------------------
 *
 *        ПРОСТЕЙШИЙ КОДЕР/ДЕКОДЕР РИДА-СОЛОМОНА
 *        ======================================
 *
 * Build 0x001 @ 02.07.2003
-----------------------------------------------------------------------------*/
// ВНИМАНИЕ! Данный кодер/декодер построен на основе обычной арифметики,
// _не_ арифметики полей Галуа, в результате чего его практические возможности
// более чем ограничены, тем не менее он нагляден и удобен для изучения
#include <stdio.h>

#define SYM_WIDE    8       // ширина входного информационного символа (бит)

#define DATAIN   0x69       // входные данные (один байт)

#define ERR_POS    3        // номер бита, который будет разрушен сбоем

// неприводимый полином
#define MAG (1<<(SYM_WIDE*1) + 1<<(SYM_WIDE*0))

// ---------------------------------------------------------------------------
// определение позиции ошибки x по остатку k от деления кодового слова на полином
// k = 2^x, где «^» - возведение в степень
// функция принимает k и возвращает x
// ---------------------------------------------------------------------------
int pow_table[9] = {1,2,4,8,16,32,64,128,256};
lockup(int x) {int a;for(a=0;a<9;a++) if(pow_table[a]==x)return a; return -1;}

main()
{
    int i; int g; int c; int e; int k;

    fprintf(stderr,"simplest Reed-Solomon endoder/decoder by Kris Kaspersky\n\n");
    i = DATAIN;                 // входные данные (информационное слово)
    g = MAG;                    // неприводимый полином
```

```
    printf("i = %08x    (DATAIN)\ng = %08x    (POLYNOM)\n", i, g);

    // КОДЕР РИДА-СОЛОМОНА (простейший, но все-таки кое-как работающий)
    // вычисляем кодовое слово, предназначенное для передачи
    c = i * g;    printf("c = %08x    (CODEWORD)\n", c);
    // конец КОДЕРА

    // передаем с искажениями
    e = c ^ (1<<ERR_POS); printf("e = %08x    (RAW RECIVED DATA+ERR)\n\n", e);
    /*       ^^^^ искажаем один бит, имитируя ошибку передачи */

    // ДЕКОДЕР РИДА-СОЛОМОНА
    // проверяем на наличие ошибок передачи
    // (фактически это простейший декодер Рида-Соломона)
    if (e % g)
    {
        // ошибки обнаружены, пытаемся исправить
        printf("RS decoder says: (%x) error detected\n{\n", e % g);
        k = (e % g);  // k = 2^x, где x - позиция сбойного бита
        printf("\t0 to 1 err  position: %x\n", lockup(k));
        printf ("\trestored codeword is: %x\n}\n", (e ^= k));
    }
    printf("RECEIVED DATA IS: %x\n", e / g);
    // КОНЕЦ ДЕКОДЕРА
}
```

Результат работы простейшего кодера/декодера Рида-Соломона показан в листинге 2.10. Обратите внимание — искаженный бит удалось успешно исправить, однако для этого к исходному информационному слову пришлось добавить не два, а целых три бита (если вы возьмете в качестве входного слова максимально допустимое восьмибитное значение `0xFF`, то кодовое слово будет равно `0x1FE00`, а так как $2^{10} = 1024$, то свободных разрядов уже не хватает и приходится увеличивать разрядную сетку до 2^{11}, в то время как младшие биты кодового слова фактически остаются незадействованными и "правильный" кодер должен их "закольцевать", грубо говоря, замкнув обрабатываемые разряды на манер кольца.

Листинг 2.10. Результат работы простейшего кодера/декодера Рида-Соломона

```
i = 00000069    (DATAIN)
g = 00000200    (POLYNOM)
c = 0000d200    (CODEWORD)
```

```
e = 0000d208    (RAW RECIVED DATA+ERR)

RS decoder says: (8) error detected
{
    0 to 1 err  position: 3
    restored codeword is: d200
}
RECEIVED DATA IS: 69
```

Общее представление

Коды Рида-Соломона представляют собой недвоичные совершенные систематические линейные блочные коды, относящиеся к классу циклических кодов с числовым полем, отличным от GF(2), и являющиеся подмножеством кодов Боуза-Чоудхури-Хоквингема. Корректирующие способности кодов Рида-Соломона напрямую зависят от количества контрольных байт. Добавление ***r*** *контрольных байт позволяют обнаруживать* ***r*** *произвольным образом искаженных байт, гарантированно восстанавливая из них* ***r/2*** *байт.*

Что читать

Несмотря на то, что данный раздел является вполне самодостаточным и весь минимально необходимый математический аппарат излагает самостоятельно без отсылок к сторонней литературе, желание углубить свои знания вполне естественно, и его можно только приветствовать. А потому будет лучше, если вы не ограничитесь одной этой книгой, но "перевернете целые горы" специализированной литературы, с каждым разом все больше и больше ужасаясь глубине той пропасти, что отделяет ваши поверхностные представления от действительно настоящих знаний. Теория помехоустойчивого кодирования столь обширна, что для ее изучения потребуется как минимум целая жизнь.

Итак, с чего начать?

- Blahut Richard "Theory and Practice of Error Control Codes", Mass.: Addison-Wesley, 1983.

  Очень хорошая книжка из категории "must have"; по слухам есть в электронном виде в сети, однако, к сожалению, самой книжки я так и не нашел, но тучи ссылок на нее убедительно свидетельствуют о высоком качестве последней. Также имеется ее русскоязычный перевод, выпущенный издательством "Мир" (см. далее).

- Блейхут Р. "Теория и практика кодов, контролирующих ошибки". — М.: Мир, 1986. — 576 с.

Технически грамотный и добротный перевод уже упомянутой ранее книги Блейхута (Blahut Richard) (ах, какие в издательстве Мир были переводчики!), электронной копии в сети, к сожалению, нет.

- James Plank "A tutorial on Reed-Solomon Coding for fault-tolerance in RAID-like systems".

  Неплохое руководство по использованию кодов Рида-Соломона для построения отказоустойчивых RAID-подобных систем, ориентированное на математически неподготовленных системных программистов и доходчиво объясняющее суть помехоустойчивого кодирования с примерами исходных текстов на языке Си. Электронная копия руководства доступна по адресу: **http://www.cs.utk.edu/~plank/plank/papers/CS-96-332.pdf**. Настоятельно рекомендую прочитать, даже если вы и не собираетесь заниматься сборкой RAID.

- Joel Sylvester "Reed-Solomon Codes".

  Предельно кратное описание принципов работы кодов Рида-Соломона с блок-схемами вместо исходных текстов. На практическое руководство не тянет, но общую картину все-таки дает, почитайте. Руководство доступно по адресу: **http://www.elektrobit.co.uk/pdf/reedsolomon.pdf**.

- Tom Moore "REED-SOLOMON PACKAGE" (old tutorial).

  Роскошный сборник разнообразных руководств по кодам Рида-Соломона, наверное, лучший из всех, что я видел. Включает в себя краткое описание основ теории полей Галуа, базовые принципы построения кодеров/декодеров Рида-Соломона и законченные примеры реализации самих кодеров/декодеров на языке Си (правда, недостаточно добросовестно прокомментированные). Сей материал (stuff) неоднократно промелькивал в сети ФИДО и последний раз был замечен 28 декабря 1994 года в конференции comp.compression. Его легко найти с помощью поисковой системы Google по ключевым словам "Reed-Solomon+main+ECC". Настоятельно рекомендую.

- Ross N.Williams "A painless guide to CRC error detection algorithms".

  Подробное руководство по CRC полезное достаточно внятным и доступным описанием полиномиальной арифметики, без которой работа с кодами Рида-Соломона просто не мыслима. Доступно в электронной форме по следующему адресу: **ftp://www.internode.net.au/clients/rocksoft/papers/crc_v3.txt**. Также имеется его неплохой перевод на русский язык, легко отыскивающийся в сети по запросу "**Элементарное руководство по CRC-алгоритмам обнаружения ошибок**". Настоятельно рекомендую.

- ftape (драйвер ленточного накопителя из дистрибутива Linux).

  Ну какая же запись на магнитную ленту обходится без корректирующих кодов? Представить себе такое, прямо-таки скажем, довольно затруднительно. Поэтому анализ исходных текстов драйверов ленточных накопи-

телей дает довольно-таки богатую пищу для размышлений (при условии, конечно, если исследуемый драйвер действительно использует коды Рида-Соломона, а не что-нибудь другое). Драйвер ftape, из дистрибутива Linux, как раз и является тем драйвером, что вам нужен, а непосредственно сам код, ответственный за кодирование/декодирование кодов Рида-Соломона вынесен в файл ftape-ECC.c/ftape-ECC.h. Это достаточно аккуратный, хорошо структурированный и даже местами слегка комментируемый код, также рекомендую.

- James S. Plank GFLIB "C Procedures for Galois Field Arithmetic and Reed-Solomon Coding".

  Библиотечка для работы с кодами Рида-Соломона. Содержит в себе полные исходные тексты всех необходимых функций и распространяется по лицензии GPL *(**G**eneral [GNU] **P**ublic **L**icense)*. Найти ее можно на любом сайте GNU, например, по адресу: **http://www.cs.utk.edu/~plank/plank/gflib/gflib.tar**.

Полиномиальная арифметика и поля Галуа

В прошлом разделе этой главы мы говорили о том, что помехоустойчивые коды Рида-Соломона основаны на двух фундаментальных математических составляющих: *полиномиальной арифметике* и *арифметике полей Галуа*. До тех пор, пока эти вопросы не будут нами всесторонне рассмотрены, мы не сможем двигаться дальше и потому наберемся чуточку терпения, чтобы совершить решительный штурм математических вершин. После чего начнется чистое программирование, практически без примесей всяких инородных математик.

Полиномиальная арифметика

Полиномиальной арифметике посвящен шестой раздел третьего тома "Искусства программирования" Дональда Кнута, где полиному дается следующее определение: *"Формально говоря, полином над S представляет собой выражение вида: $u(x) = u_n x^n + \ + u^1 x + u_0$, где коэффициенты un, u1, u0 — элементы некоторой алгебраической системы S, а переменная x может рассматриваться как формальный символ без определяющего значения. Будем полагать, что алгебраическая система S представляет собой коммутативное кольцо с единицей. Это означает, что S допускает операции сложения, вычитания и умножения, удовлетворяющие обычным свойствам: сложение и умножение являются ассоциативными и коммутативными бинарными операциями, определенными на S, причем умножение дистрибутивно по отношению к сложению. Существует также единичный элемент по сложению 0 и единичный элемент по умножению 1, такие, что a + 0 == a и a * 1 == a для всех a из S.*

Вычитание является обратной по отношению к сложению операцией, но о возможности деления как операции, обратной по отношению к умножению, ничего не предполагается. Полином $0x^{n+m} + \ + 0x^{n+1} + u_n x^n + \ + u^1 x + u_0$ *рассматривается как идентичный* $u_n x^n + \ + u^1 x + u_0$*, хотя формально он отличается от него".*

Таким образом, вместо того, чтобы представлять информационное слово *D*, кодовое слово *C* и остаток от деления *R* в виде целых чисел (как это делалось нами ранее), мы можем связать их с соответствующими коэффициентами двоичного полинома, выполняя все последующие математические манипуляции по правилам полиномиальной арифметики. Выигрыш от такого преобразования на первый взгляд далеко не очевиден, но не будем спешить, а лучше преобразуем любое пришедшее нам в голову число (например, 69h) в двоичный полином. Запустив "Калькулятор" или любое другое подходящее приложение по вашему вкусу, переведем наше число в двоичный вид (при соответствующих навыках эту операцию можно выполнить и в уме, см. "Техника и философия хакерских атак" Криса Касперски): 69h → **1101001**.

Ага, крайний правый коэффициент равен единице, затем следуют два нулевых коэффициента, потом единичный коэффициент, короче говоря, получается следующее: $\mathbf{1}x^6 + \mathbf{1}x^5 + \mathbf{0}x^4 + \mathbf{1}x^3 + \mathbf{0}x^2 + \mathbf{0}x + \mathbf{1}$. По сути говоря, битовая строка "1101001" является одной из форм записи вышеуказанного полинома, — ненаглядной с точки зрения неподготовленного человека, но удобной для машинной обработки. Постойте, но если `69h` *уже* представляет собой полином, то в чем разница между сложением полиномов `69h` и `27h` и сложением целых чисел `69h` и `27h`?! Разница, несомненно, есть. Как еще показал Ницше: фактов нет, а есть одни лишь интерпретации. Интерпретация же чисел и полиномов различна, и математические операции над ними выполняются по совершенно независимым правилам.

Коэффициенты в полиномиальной арифметике строго типизированы, и коэффициент при x^k имеет иной тип, нежели при x^m (конечно, при том условии, что $k \neq m$). А операции над числами различных типов категорически недопустимы! Все коэффициенты обрабатываются независимо, а возникающий при этом перенос в старший разряд (заем из старшего разряда) попросту не учитывается. Покажем это на примере сложения чисел `69h` и `27h` (листинг 2.11). Сложение, выполненное по правилам полиномиальной двоичной арифметики (слева), и сложение, выполненное по правилам обычной арифметики (справа).

Листинг 2.11. Пример сложения чисел 69h и 27h

```
 1101001 (69h)          1101001 (69h)
+0100111 (27h)         +0100111 (27h)
 -------                -------
 1001110 (4Eh)         10010000 (90h)
```

Простейшие расчеты показывают, что сложение полиномов по модулю два дает тот же самый результат, что их вычитание, и "волшебным" образом совпадает с битовой операцией `XOR`. Впрочем, совпадение с этой операцией — чистая случайность, но вот эквивалентность сложения и вычитания заставляет заново пересматривать привычную природу вещей, вспоминая задачки из серии "у Маши было одно яблоко, Петя отнял у нее его, затем ей подарил еще одно, спрашивается: сколько всего яблок у Маши осталось? А сколько у нее было бы, если бы первое яблоко осталось не отнятым?". С точки зрения арифметики по модулю два ответ: один и ноль соответственно. Да! Не отними бы Петя у Маши яблоко, `1 + 1 == 0`, и бедная Маша вообще осталась бы ни с чем. Так что, мальчики, почаще отнимайте яблоки у девушек — учите их компьютерной грамотности!

Впрочем, мы отвлеклись. Ладно, оставим в покое половые молодежные разборки и вернемся к фиктивному члену *x* нашего полинома и его коэффициентам. Благодаря их типизации и отсутствию взаимных связей, мы можем осуществлять обработку сколь угодно длинных чисел, просто выполняя операцию `XOR` (исключающее ИЛИ) над составляющими их битами на потоке. Это и есть одно из тех достоинств полиномиальной арифметики, которые не видны с первого взгляда, но благодаря которым полиномиальная арифметика стала так широко распространена.

Однако в нашем случае одной лишь полиномиальной арифметикой дело не обходится и для реализации кодера/декодера Рида-Соломона нам потребуется активная помощь со стороны полей Галуа. Что же это за поля такие, спросите вы?

Поля Галуа

В далеких шестидесятых, когда компьютеры были большими, а двадцатимегабайтовые винчестеры напоминали собой стиральные машины, родилась одна из красивейших легенд о зеленом инопланетном существе, прилетевшим со звезд и записавшим всю Британскую энциклопедию на тонкий металлический стержень нежно-серебристого цвета, который существо и увезло с собой. Сегодня, когда габариты 100 ГГб жестких дисков сократились до размеров сигаретной пачки, такая плотность записи информации уже не кажется удивительной и даже вызывает улыбку. Но! Все дело в том, что инопланетное существо обладало технологией записи *бесконечного количества* информации на *бесконечно крошечном отрезке* и Британская энциклопедия была выбрана лишь для примера. С тем же успехом инопланетянин мог скопировать содержимое всех серверов Интернета, нанеся на свой металлический стержень всего одну-единственную риску. Не верите? А зря! Переводим Британскую энциклопедию в цифровую форму, получая огромное-преогромное число. Затем — ставим впереди него запятую, преобразуя записываемую информацию в длиннющую десятичную дробь. Теперь только

остается найти два числа *A* и *B*, таких, что результат деления *A* и *B* как раз и будет равен данному числу с точностью до последнего знака. Запись этих чисел на металлический стержень осуществляется нанесением риски, делящей последний на два отрезка с длинами, кратными величинам *A* и *B* соответственно. Для считывания информации достаточно всего лишь измерить длины отрезков *A* и *B*, а затем — поделить один на другой. Первый десяток чисел после запятой будет более или менее точен, ну а потом... Потом жестокая практика опустит абстрактную теорию по самые помидоры, окончательно похоронив последнюю под толстым слоем информационного мусора, возникающего из невозможности точного определения геометрических размеров объектов реального мира.

В цифровом мире дела обстоят еще хуже. Каждый программист знает, что на деление целых и вещественных чисел наложены достаточно жесткие ограничения. Помимо того, что деление весьма прожорливая в плане процессорных ресурсов операция, так она еще и математически неточная! То есть если `c = a * b`, то еще не факт, что `a == c/b`! Таким образом, для практической реализации кодов Рида-Соломона обычная арифметика непригодна, и приходится прибегать к помощи особой математики — математики конечных *групп Галуа*.

Под *группой* здесь понимается совокупность целых чисел, последовательно пронумерованных от 0 до $2^n - 1$, например: {0, 1, 2, 3} или {00h, 01h, 02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 0Ah, 0Bh, 0Ch, 0Dh, 0Eh, 0Fh}. Группы, содержащие 2^n элементов, называются *полями Галуа (Galois Field)* и обозначаются так: **GF(2^n)**.

Замечание

На самом же деле, полями Галуа называют любые конечные поля, но в данном контексте мы будем говорить лишь о тех полях, количество членов которых равно 2^n.

Члены групп в обязательном порядке подчиняются ассоциативному, коммутативному и дистрибутивному законам, но обрабатываются довольно противоестественным на первый взгляд образом:

1. Сумма двух любых членов группы всегда присутствует в данной группе.
2. Для каждого члена *a* группы существует *тождественный (identity)* ему член, обычно записываемый как *e*, удовлетворяющий следующему условию: `a + e = e + a = a`.
3. Для каждого члена *a* группы существует *обратный (inverse)* ему член $-a$, такой что: `a + -a == 0`.

Начнем с первого тезиса. Не кажется ли он вам бредом? Допустим, у нас есть группа {0, 1, 2, 3}. Это каким же в дупель пьяным нужно быть, чтобы при вычислении значения 2 + 3 получить число меньшее или равное 3?!

Оказывается, сложение в полях Галуа осуществляется *без* учета переноса и сумма двух членов группы равна: $\boxed{c = (a + b) \% 2^n}$, где операция "%" обозначает взятие остатка. Применительно к нашему случаю: `(2 + 3) % 4 == 1`. У математиков это называется "сложением по модулю 4".

Естественно, вас интересует: а применяется ли сложение по модулю на практике или используется лишь в абстрактных конструкциях теоретиков? Хороший вопрос! Сложение по модулю мы машинально выполняем десятки раз на дню, даже не задумываясь о том, что это и есть сложение без учета переноса. Вот, например, проснувшись в шесть вечера поутру, вы просидели за компьютером девять часов кряду, а потом неожиданно бросили взгляд на свои наручные часы. Какое положение занимала часовая стрелка в это время, при условии, что часы идут точно? Искомое значение со всей очевидностью представляет собой сумму 6 и 9 по модулю 12, и равно оно: `(6 + 9) % 12 == 3`. Вот вам наглядный пример практического использования арифметики Галуа. А теперь давайте в порядке эксперимента вычтем из числа 3 число 6 (если не догадались, как это правильно сделать, то возьмите в руки часы).

Теперь самое главное: раз результат деления одного члена группы на другой, естественно, неравный нулю член, в обязательном порядке должен присутствовать в данной группе, то несмотря на то, что деление осуществляется в целых числах, оно будет точным. Точным, а не округленным! Следовательно, если `c = a * b`, то `a == c/b`. Другими словами, умножение и деление непротиворечивым образом определено для всех членов группы, конечно, за исключением невозможности деления на нуль, причем расширения разрядной сетки при умножении не происходит!

Конечно, это не совсем обычное умножение (и далеко не во всяком поле Галуа дважды два будет рано четырем), однако никто и не требует от арифметики Галуа ее соответствия "здравому смыслу" и "житейскому опыту". Главное — что она работает, причем работает хорошо. И существование жестких дисков, CD-ROM/DVD-приводов — лучшее тому подтверждение, ибо все они так или иначе используют эту арифметику в своих целях.

Как уже говорилось, в вычислительной технике наибольшее распространение получили поля Галуа с основанием 2, что объясняется естественностью этих полей с точки зрения машинной обработки, двоичной по своей природе.

Для реализации кодера/декодера Рида-Соломона нам потребуются четыре базовых арифметических операции: *сложение*, *вычитание*, *умножение* и *деление*. Далее они будут рассмотрены во всех подробностях.

Сложение и вычитание в полях Галуа

Сложение по модулю два в полях Галуа тождественно вычитанию и реализуется битовой операцией `XOR`. Этот вопрос мы уже обсуждали при изучении полиномиальной арифметики, поэтому не будем лишний раз повторяться,

а просто приведем законченный пример программной реализации функции сложения/вычитания (листинг 2.12).

Листинг 2.8. Функция, реализующая сложение/вычитание в полях Галуа

```
// функция возвращает результат сложения (вычитания)
// двух полиномов a и b по модулю 2
int gf_sum(int a, int b)
{
        return a ^ b;
}
```

Умножение в полях Галуа

Открыв учебник математики за третий класс (если мне не изменяет память), мы найдем, что умножение представляет собой многократное сложение, и, коль скоро сложение в полях Галуа мы выполнять уже научились, мы имеем все основания считать, что реализация функции умножения не создаст особого труда. Так? А вот и нет! Я всегда *знал*, что дважды два равно четырем, до конца никогда *не верил* в это и, впервые столкнувшись с полями Галуа, понял, насколько был прав.

Примечание

Другими словами говоря, щелкая выключателем, я *знаю*, что сейчас загорится свет. Но я не уверен в этом (монтер перерезал провода, лампочка перегорела и т. д.). Вот так и с математикой. Та жвачка, которой пичкают нас в школе и позже в институте — это не математика. Это набор шаманских обрядов, который нас заставляют совершать, но который не позволяет приникнуть в самую суть — в дао математики. Может, оно и к лучшему, не знаю, но во всяком случае считаю долгом сказать, что "математика", преподаваемая в средних и высших учебных заведениях, имеет к математике не больше отношения, чем программирование к терзанию мыши в Word и установке операционной системы Windows.

Выяснилось, что существуют и такие математики, где дважды два не равно четырем, а операция умножения определяется не через сложение, а совсем по-другому.

Действительно, если попытаться "обернуть" функцию `gf_sum` в цикл, мы получим то же самое сложение, только в профиль. $a * b$ будет равно a, если b четно, и нулю, если b — нечетно. Ну, и кому такое умножение нужно? Собственно, функция "настоящего" умножения Галуа настолько сложна и ресурсоемка, что для упрощения ее реализации приходится прибегать к временному преобразованию полиномов в индексную форму, последующему сложению индексов, выполняемому по модулю GF, и обратному преобразованию суммы индексов в полиномиальную форму.

Что такое *индекс*? Это — показатель степени при основании два, дающий искомый полином. Например, индекс полинома 8 равен 3 ($2^3 = 8$), а индекс полинома 2 равен 1($2^1 = 2$). Легко показать, что $a * b = 2^i + 2^j = 2^{(i+j)}$. В частности, $2 * 8 = 2^3 + 2^1 = 2^{(3+1)} = 4^4 = 16$. Составим, к примеру, табличку (табл. 2.2) и немного поэкспериментируем с ней.

Таблица 2.2. *Таблица полиномов (левая колонка) и соответствующих им степеней двойки (правая колонка)*

i	alpha_of[i]
001	0
002	1
004	2
008	3
016	4

До сих пор мы оперировали понятиями привычной нам арифметики, и потому добрые две трети полей таблицы остались незаполненными. В самом деле, уравнения типа $2^x = 3$ в целых числах неразрешимы, и ряд индексов не соответствует никаким полиномам! Так-то оно так, но в силу того, что количество полиномов всякого поля Галуа равно количеству всевозможных индексов, мы можем определенным образом сопоставить их друг другу, закрыв глаза на то, что, с точки зрения обычной математики, такое действие не имеет никакого смысла. Конкретная схема сопоставления может быть любой, главное — чтобы она была внутренне непротиворечивой, то есть удовлетворяла всем правилам групп, перечисленным ранее *(см. разд. "Поля Галуа" этой главы)*.

Естественно, поскольку от выбранной схемы сопоставления напрямую зависит и конечный результат, обе стороны (кодер и декодер Рида-Соломона) должны соблюдать определенные договоренности. Однако различные кодеры/декодеры Рида-Соломона могут использовать различные схемы сопоставления, несовместимые друг с другом.

В частности, декодер Рида-Соломона, встроенный в привод CD-ROM, выполняет умножение по следующей таблице. Встретив такую таблицу в дизассемблерном листинге исследуемой вами программы, вы сможете быстро и надежно отождествить использующие ее функции (листинг 2.13). Первая слева колонка — полиномы/индексы (обычно обозначается как `i`), вторая — таблица степеней примитивного полинома 2 (обычно обозначается как `alpha_of`), третья — индексы, соответствующие данному полиному (обычно обозначается как `index_of`).

Листинг 2.13. Таблица lock-up для GF(256)

i	alpha	index	i	alpha	index	i	alpha	index	i	alpha	index	i	alpha	index	i	alpha	index
000	001	-1	043	119	218	086	177	219	129	023	112	172	123	220	215	239	170
001	002	0	044	238	240	087	127	189	130	046	192	173	246	252	216	195	251
002	004	1	045	193	18	088	254	241	131	092	247	174	241	190	217	155	96
003	008	25	046	159	130	089	225	210	132	184	140	175	255	97	218	043	134
004	016	2	047	035	69	090	223	19	133	109	128	176	227	242	219	086	177
005	032	50	048	070	29	091	163	92	134	218	99	177	219	86	220	172	187
006	064	26	049	140	181	092	091	131	135	169	13	178	171	211	221	069	204
007	128	198	050	005	194	093	182	56	136	079	103	179	075	171	222	138	62
008	029	3	051	010	125	094	113	70	137	158	74	180	150	20	223	009	90
009	058	223	052	020	106	095	226	64	138	033	222	181	049	42	224	018	203
010	116	51	053	040	39	096	217	30	139	066	237	182	098	93	225	036	89
011	232	238	054	080	249	097	175	66	140	132	49	183	196	158	226	072	95
012	205	27	055	160	185	098	067	182	141	021	197	184	149	132	227	144	176
013	135	104	056	093	201	099	134	163	142	042	254	185	055	60	228	061	156
014	019	199	057	186	154	100	017	195	143	084	24	186	110	57	229	122	169
015	038	75	058	105	9	101	034	72	144	168	227	187	220	83	230	244	160
016	076	4	059	210	120	102	068	126	145	077	165	188	165	71	231	245	81
017	152	100	060	185	77	103	136	110	146	154	153	189	087	109	232	247	11
018	045	224	061	111	228	104	013	107	147	041	119	190	174	65	233	243	245
019	090	14	062	222	114	105	026	58	148	082	38	191	065	162	234	251	22
020	180	52	063	161	166	106	052	40	149	164	184	192	130	31	235	235	235
021	117	141	064	095	6	107	104	84	150	085	180	193	025	45	236	203	122
022	234	239	065	190	191	108	208	250	151	170	124	194	050	67	237	139	117
023	201	129	066	097	139	109	189	133	152	073	17	195	100	216	238	011	44
024	143	28	067	194	98	110	103	186	153	146	68	196	200	183	239	022	215
025	003	193	068	153	102	111	206	61	154	057	146	197	141	123	240	044	79
026	006	105	069	047	221	112	129	202	155	114	217	198	007	164	241	088	174
027	012	248	070	094	48	113	031	94	156	228	35	199	014	118	242	176	213
028	024	200	071	188	253	114	062	155	157	213	32	200	028	196	243	125	233
029	048	8	072	101	226	115	124	159	158	183	137	201	056	23	244	250	230
030	096	76	073	202	152	116	248	10	159	115	46	202	112	73	245	233	231
031	192	113	074	137	37	117	237	21	160	230	55	203	224	236	246	207	173
032	157	5	075	015	179	118	199	121	161	209	63	204	221	127	247	131	232
033	039	138	076	030	16	119	147	43	162	191	209	205	167	12	248	027	116
034	078	101	077	060	145	120	059	78	163	099	91	206	083	111	249	054	214
035	156	47	078	120	34	121	118	212	164	198	149	207	166	246	250	108	244
036	037	225	079	240	136	122	236	229	165	145	188	208	081	108	251	216	234
037	074	36	080	253	54	123	197	172	166	063	207	209	162	161	252	173	168

038 148 15	081 231 208	124 151 115	167 126 205	210 089 59	253 071 80
039 053 33	082 211 148	125 051 243	168 252 144	211 178 82	254 142 88
040 106 53	083 187 206	126 102 167	169 229 135	212 121 41	255 000 175
041 212 147	084 107 143	127 204 87	170 215 151	213 242 157	
042 181 142	085 214 150	128 133 7	171 179 178	214 249 85	

С помощью данной таблицы вы легко сможете осуществлять преобразование из полиномиальной формы в индексную и наоборот. Как пользоваться этой таблицей? Допустим, мы хотим умножить полиномы 69 и 96. Находим в первой колонке число 69. Ему соответствует `alpha` 47, запоминаем (записываем его на бумажке) и переходим к числу 96, `alpha` которого равен 217. Складываем 47 и 217 по модулю 256, получая в результате: (217 + 47) % 256 = 8. Теперь переводим результат произведения из индексной формы в полиномиальную: находим в первой колонке число 8 и в третьей колонке видим соответствующий ему полином: 3. (Если же мы выполним обратную операцию, разделив 3 на 69, — мы получим 96, что доказывает непротиворечивость операций деления и умножения, а также всей арифметики Галуа в целом.) Быстро, не правда ли, хотя местами и не совсем понятно, почему таблица составлена именно так, а не иначе? Хуже всего, что достоверность результата нельзя почувствовать "в живую", поскольку все это — абстракции чистейшей воды, что серьезно осложняет отладку программы (сложно отлаживать то, чей принцип работы до конца не понимаешь).

Впрочем, таблицу умножения не обязательно набивать с клавиатуры вручную, и ее вполне можно генерировать и "на лету" по ходу исполнения программы. Один из примеров реализации генератора выглядит так, как показано в листинге 2.14.

Листинг 2.14. Процедура генерации таблицы look-up быстрого умножения полиномов

```
#define m    8     // степень RS-полинома    (согласно Стандарту ECMA-130 - восемь)
#define n    255   // n=2**m-1        (длина кодового слова)
#define t    1     // количество ошибок, которые мы хотим скорректировать
#define k    253   // k = n-2*t    (длина информационного слова)

// несократимый порождающий полином
// согласно Стандарту ECMA-130: P(x) = x^8 + x^4 + x^3 + x^2 + 1
int p[m+1]={1, 0, 1, 1, 1, 0, 0, 0, 1 };

int alpha_to[n+1];            // таблица степеней примитивного члена
int index_of[n+1];            // индексная таблица для быстрого умножения
```

```
//-----------------------------------------------------------------------------
// генерируем look-up-таблицу для быстрого умножения для GF(2^m) на основе
// несократимого порождающего полинома P(c) от p[0] до p[m].
//
// look-up-таблица:
//            index->polynomial из alpha_to[] содержит j=alpha^i,
//            где alpha есть примитивный член, обычно равный 2
//            a ^ - операция возведения в степень (не XOR!);
//
//            polynomial form -> index из index_of[j=alpha^i] = i;
//
// © Simon Rockliff
//-----------------------------------------------------------------------------

generate_gf()
{
    int i, mask;

    mask = 1; alpha_to[m] = 0;

    for (i = 0; i < m; i++)
    {
         alpha_to[i] = mask;
         index_of[alpha_to[i]] = i;

         if (p[i] != 0) alpha_to[m] ^= mask;
         mask <<= 1;
    } index_of[alpha_to[m]] = m; mask >>= 1;

    for (i = m+1; i < n; i++)
    {
         if (alpha_to[i-1] >= mask)
               alpha_to[i] = alpha_to[m] ^ ((alpha_to[i-1]^mask)<<1);
         else
               alpha_to[i] = alpha_to[i-1]<<1;

         index_of[alpha_to[i]] = i;
    } index_of[0] = -1;
}
```

Сама же функция умножения выглядит тривиально (листинг 2.15), укладываясь всего в пяток строк. В большинстве программных реализаций кодера/декодера Рида-Соломона, которые мне только доводилось видеть, операция умножения даже не выносится в отдельную процедуру, а реализуется непосредственно по месту вызова.

Листинг 2.15. Функция быстрого табличного умножения полиномов в полях Галуа

```
// функция возвращает результат умножения
// двух полиномов a на b в полях Галуа
int gf_mul(int a, int b)
{
    int sum;
    if (a == 0 || b == 0) return 0;      // немного оптимизации не повредит
    sum = alpha_of[a] + alpha_of[b];     // вычисляем сумму индексов полиномов
    if (sum >= GF-1) sum -= GF-1;        // приводим сумму к модулю GF
    return index_of[sum];                // переводим результат в полиномиальную…
                                         // …форму и возвращаем результат
}
```

Деление в полях Галуа

Деление в полях Галуа осуществляется практически точно так, как и умножение, с той лишь разницей, что индексы не прибавляются, а вычитаются друг из друга. В самом деле: $a/b == 2^i/2^j == 2^{(i-j)}$. Для перевода из полиномиальной в индексную форму и наоборот может использоваться уже приводимая выше таблица look-up.

Естественно, не забывайте о том, что какими бы извращенными поля Галуа ни были, а на нуль даже в абстрактной арифметике делить нельзя и функция деления должна быть снабжена соответствующей проверкой.

Листинг 2.16. Функция быстрого табличного деления полиномов в полях Галуа

```
// функция возвращает результат деления
// двух полиномов a на b в полях Галуа
// при попытке деления на ноль функция
// возвращает -1
int gf_div(int a, int b)
{
    int diff;
    if (a == 0) return 0;                // немного оптимизации не повредит
    if (b == 0) return -1;               // на ноль делить нельзя!
```

```
    diff = alpha_of[a] - alpha_of[b];     // вычисляем разность индексов
    if (diff < 0) diff += GF-1;           // приводим разность к модулю GF
    return index_of[diff];                // переводим результат в полиномиальную…
                                          // …форму и возвращаем результат
}
```

Простейшие практические реализации

Хорошим примером воплощения кодера/декодера Рида-Соломона являются "древние" модели жестких дисков, разработанных в недрах фирмы IBM. Модель **IBM 3370** имела простой и наглядный кодер/декодер Рида-Соломона типа (174,171) в поле Галуа GF(256). Другими словами, он оперировал 8-битными ячейками ($2^8 = 256$), и на 171 информационный байт приходилось 3 байта суммы четности, что в результате давало кодовое слово с размером 174 байт, причем, как мы увидим далее, все три байта контрольной суммы рассчитывались совершенно независимо друг от друга, поэтому фактически кодер/декодер Рида-Соломона оперировал *одним байтом*, что значительно упрощало его архитектуру.

В современных же винчестерах кодер/декодер Рида-Соломона стал слишком навороченным, а количество контрольных байтов многократно возросло, в результате чего пришлось работать с числами противоестественных разрядностей (порядка 1408 бит и более). Как следствие — программный код ощетинился толстым слоем дополнительных проверок, циклов и функций, чрезвычайно затрудняющих его понимание (к тому же большинство производителей железа в последнее время перешли на аппаратные кодеры/декодеры Рида-Соломона, полностью реализованные в одной микросхеме). В общем, прогресс — прогрессом, а для изучения базовых принципов работы лучше использовать "древние" модели.

Ниже (листинги 2.17 и 2.18) приведены два фрагмента оригинальной прошивки жесткого диска IBM 3370 (только не спрашивайте, откуда они у меня взялись).

Листинг 2.17. Ключевой фрагмент кодера Рида-Соломона, вырванный из прошивки IBM 3370

```
for (s0 = s1 = sm1 = i = 0; i < BLOCK_SIZE; ++i)
{
    s0  =                           s0 ^ input[i];
    s1  =         GF_mult_by_alpha[ s1 ^ input[i] ];
    sm1 = GF_mult_by_alpha_inverse[sm1 ^ input[i] ];
};
```

Листинг 2.18. Ключевой фрагмент декодера Рида-Соломона, вырванный из IBM 3370

```
err_i = GF_log_base_alpha[ GF_divide[s1][s0] ];     // вычисляем синдром ошибки
input[err_i] ^= s0;                                 // исправляем сбойный байт
```

Ну, что, слабо нам разобраться: как он работает? Что касательно переменной `s0` — с ней все предельно ясно: она хранит контрольную сумму, рассчитанную по тривиальному алгоритму. Как вы, наверное, помните, сложение в полях Галуа осуществляется логической операцией `XOR` и потому: `s0 += input[i]`.

Назначение переменной `s1` выяснить сложнее, и чтобы понять суть разворачивающегося вокруг нее "метаболизма", мы должны знать содержимое таблицы `GF_mult_by_alpha`. Несмотря на то, что по соображениям экономии бумажного пространства она здесь не приводится, ее имя говорит само за себя: содержимое `s1` суммируется с очередным байтом контролируемого потока данных и умножается на так называемый *примитивный член*, обозначаемый как `alpha` и равный двум. Другими словами: `s1 = 2 * (s1 + input[i])`.

Допустим, один из байтов потока данных впоследствии будет искажен (обозначим его позицию как `err_i`), тогда индекс искаженного байта можно определить тривиальным делением `s1` на `s0`. Почему? Так ведь выражение `s1 = 2 * (s1 + input[i])` по своей сути есть не что иное, как завуалированное умножение информационного слова на порожденный полином, динамически генерируемый на основе своего примитивного члена `alpha`. А контрольная сумма информационного слова, хранящаяся в переменной `s0`, фактически представляет собой то же самое информационное слово, только представленное в более "компактной" форме. И, как уже говорилось в предыдущей главе: если ошибка произошла в позиции x, то остаток от деления кодового слова на порожденный полином будет равен $k = 2^x$. Остается лишь по известному k вычислить x, что в данном случае осуществляется путем обращения к таблице `GF_log_base_alpha`, хранящей пары соответствий между k и 2^x. Коль скоро позиция сбойного байта найдена, его можно исправить путем выполнения операции `XOR` с рассчитанной контрольной суммой `s0` (`input[err_i] ^= s0`). Конечно, сказанное справедливо только для одиночных ошибок, а искажения двух и более байт на блок данный алгоритм исправить не в силах. Собственно, для этого и присутствует третий байт контрольной суммы — `sm1`, — защищающий декодер от "политнекорректных" попыток исправления ошибок, когда их больше одной. Если выражение `s1/s0 == sm1 * s0` становится ложным, — контроллер винчестера может засвидетельствовать факт наличия множественных ошибок, констатируя невозможность их исправления.

Однако, как хорошо известно, дефекты магнитной поверхности имеют тенденцию образовывать не одиночные, а групповые ошибки. И, чтобы хоть

как-то компенсировать слабость корректирующего алгоритма, парни из IBM прибегли к *чередованию* байт. Винчестер IBM 3370 имел чередование 3:1, означающее то, что сначала шел первый байт первого блока, за ним первый байт второго блока, за ним — первый байт третьего и только потом — второй байт первого блока. Такой трюк усиливал корректирующую способность винчестера с одной одиночной ошибки до трех *последовательно* искаженных байт. Однако если разрушению подвергались не соседние байты, то корректирующая способность вновь опускалась до значений в один искаженный байт на блок, но вероятность такого события была несравненно меньше.

Естественно, что данный алгоритм может быть реализован не только в самом жестком диске, но и вне его. Варьируя размер блоков и степень чередования, вы обеспечите себе лучшую или худшую защищенность при большей или меньшей избыточности информации. Действительно, пусть у нас есть N секторов на диске. Тогда, разбив их на блоки по 174 сектора в каждом и выделив 3 сектора для хранения контрольной суммы, мы сможем восстановить по меньшей мере $N/174$ секторов диска. Исходя из средней емкости диска в 100 Гбайт (что соответствует 209 715 200 секторам), мы сможем восстановить до 1 205 259 секторов даже при их полном физическом разрушении, затратив всего лишь 2% дискового пространства для хранения контрольных сумм. Согласитесь, что редко когда процесс "осыпания" магнитного слоя дисков винчестера проходит столь стремительно, чтобы корректирующих способностей кода Рида-Соломона оказалась недостаточно для его воскрешения. Конечно, если это "осыпание" вовремя заметить и если коэффициент чередования выбран правильно: так, что сектора, принадлежащие одному магнитному диску ("блину"), обслуживались бы разными корректирующими блоками, в противном случае при повреждении поверхности одного из "блинов" возникнет групповая ошибка, уже не исправляемая данной программой.

А как быть, если выйдет из строя весь жесткий диск целиком? Наиболее разумный выход — создать массив из нескольких дисков, хранящих полезную информацию вперемешку с корректирующими кодами. Главный минус такого подхода — его неэффективность на массивах, состоящих из небольшого количества жестких дисков. Разумный минимум — это четыре информационных диска и один контрольный, когда потеря любого из информационных дисков компенсируется оставшимся "в живых" контрольным. Ну, а потерянный контрольный диск элементарным образом заменяется на новый, с последующим пересчетом всех контрольных кодов. Правда, одновременный выход двух дисков из строя — это "кранты". Массив из пятнадцати дисков RAID, двенадцать из которых — информационные, а оставшиеся три — контрольные, намного более отказоустойчив и допускает одновременный крах двух *любых* дисков, а при благоприятном стечении обстоятельств — и трех.

Собственно, во всем этом ничего нового нет, и соответствующие RAID-контроллеры можно купить буквально в любом магазине. Однако мне трудно представить себе, сколько будет стоить RAID-контроллер уровня 15 и удастся ли его вообще заставить работать (по личному опыту могу сказать, что RAID-контроллеры даже начальных уровней — вещь крайне глючная, капризная и требовательная как к железу, так и к операционному окружению). Наконец, практически все RAID-контроллеры требуют наличия абсолютно идентичных, ну или близких по своим характеристикам и/или интерфейсам дисков. А коли таковых нет?

Программный RAID, активно пропагандируемый настоящим автором, всех этих недостатков лишен. Вы можете использовать диски различной геометрии и даже различной емкости, причем никто не обязывает вас сосредотачивать их в одном месте — доступ к дискам может осуществляться и по сети, причем совершенно необязательно отводить под RAID-хранилище весь диск целиком! Вы вольны произвольным образом выделять ту или иную часть дискового пространства.

Как это можно реально использовать на практике? Первое, что приходит на ум, использовать часть емкости жестких дисков под хранение избыточной информации, помогающей восстановить их в случае аварии. Если несколько компьютеров объединить в сеть (что уже давным-давно сделано и без нас), то при относительно небольших накладных расходах мы сможем восстановить любой из жестких дисков членов сети даже при полном его разрушении лишь за счет одной избыточной информации, распределенной между остальными компьютерами. Более надежного хранилища для ваших данных нельзя и придумать! Подобная схема была реализована автором в локальных сетях нескольких фирм и доказала свою высокую живучесть, гибкость и функциональность. Необходимость в постоянном резервировании содержимого жестких дисков автоматически отпала, что в условиях одноранговой сети с отсутствующим выделенным сервером более чем актуально! А ведь такие локальные сети — не редкость (нет, я не утверждаю, что такие сети хороши, просто я констатирую факт, что они существуют в природе и в обозримом будущем "вымирать" не собираются).

Единственный минус программного RAID — его невысокая производительность. В частности, поставив программный RAID на сервер, обрабатывающий тысячи запросов ежесекундно и интенсивно модифицирующий большое количество файлов, вы не выиграете ничего, но ведь само понятие "производительности" очень относительно и при достаточно быстром процессоре кодирование/декодирование информации вполне реально осуществлять и "на лету" безо всяких потерь в пропускной способности! С другой стороны, если операции чтения доминируют над операциями записи, то ставить программный RAID сам "Крестный Отец" велел, поскольку контроль целостности считываемой информации осуществляется на "железном" уровне самим приводом и при использовании систематического кодирования

(т. е. информационные слова — отдельно, байты четности — отдельно) декодеру Рида-Соломона нет никакой нужды как-то вмешиваться в этот процесс и его помощь требуется лишь тогда, когда часть информации оказывается безнадежно разрушена, что случается прямо-таки скажем не часто. Так что, право же, не стоит "перекармливать" фирмы, специализирующиеся на выпуске RAID, тем более что на домашний и мелко-офисный рынок они все равно не обращают внимания.

Коды Рида-Соломона в практических реализациях

В предыдущих разделах мы рассмотрели базовый математический аппарат, на который опираются коды Рида-Соломона, и исследовали простейший кодер/декодер, способный исправлять одиночные ошибки и работающий с двумя символами четности. Для подавляющего большинства задач такой корректирующей способности оказывается катастрофически недостаточно, и тогда приходится задумываться о реализации более мощного кодера/декодера.

Кодер/декодер, рассматриваемый в настоящей главе, чрезвычайно конфигурабелен и может быть настроен на работу с любым количеством символов четности, а это означает, что при разумной избыточности он способен исправлять *любое* мыслимое количество ошибок. Подобная универсальность не проходит даром, и конструкция такого декодера усложнятся более чем в сто (!) раз. Самостоятельное проектирование декодеров Рида-Соломона требует глубоких знаний высшей математики в целом и природы корректирующих кодов в частности, поэтому не смущайтесь, если данная глава поначалу вам покажется непонятной. Это действительно сложные вещи, не допускающие простого объяснения.

С другой стороны, для практического использования корректирующих кодов можно и не вникать в их сущность, просто откомпилировав исходные тексты кодера/декодера Рида-Соломона, приведенные в данной главе. Также вы можете воспользоваться любой законченной библиотекой, поставляемой сторонними разработчиками. В качестве альтернативного примера в заключении этой главы будет кратко описан интерфейс библиотеки *ElByECC.DLL*, разработанной компанией "Elaborate Bytes" и распространяемой вместе с популярным копировщиком CloneCD. Известнейший "прожигатель" дисков всех времен и народов Ahead Nero Burning ROM имеет аналогичную библиотеку, размещенную в файле NEWTRF.DLL.

Легенда

Напомним читателю основные условные обозначения, используемые в этой главе. Количество символов кодируемого сообщения (называемого также информационным словом) по общепринятому соглашению обозначается

букой k; полная длина кодового слова, включающего в себя кодируемые данные и символы четности, — n. Отсюда количество символов четности равно: $n - k$. За максимальным количеством исправляемых ошибок "закреплена" буква t. Поскольку для исправления одной ошибки требуется два символа четности, общее количество символов четности равно $2t$. Выражение RS(n, k) описывает определенную разновидность корректирующих кодов Рида-Соломона, оперирующую с n-символьными блоками, k символов из которых представляют полезные данные, а все остальные задействованы под символы четности.

Полином, порожденный на основе примитивного члена α, называется *порожденным* или *сгенерированным* (generate) полиномом.

Кодировщик (encoder)

Существует, по меньшей мере, два типа кодеров Рида-Соломона: *несистематические* и *систематические* кодировщики.

Вычисление несистематических корректирующих кодов Рида-Соломона осуществляется *умножением* информационного слова на порожденный полином, в результате чего образуется кодовое слово, полностью отличающееся от исходного информационного слова, а потому для непосредственного употребления категорически непригодное. Для приведения полученных данных в исходный вид мы должны в обязательном порядке выполнить ресурсоемкую операцию декодирования, даже если данные не искажены и не требуют восстановления!

При систематическом кодировании, напротив, исходное информационное слово останется неизменным, а корректирующие коды (часто называемые *символами четности*) добавляются в его конец, благодаря чему к операции декодирования приходится прибегать лишь в случае действительного разрушения данных. Вычисление несистематических корректирующих кодов Рида-Соломона осуществляется *делением* информационного слова на порожденный полином. При этом все символы информационного слова сдвигаются на $n - k$ байт влево, а на освободившееся место записывается $2t$ байт остатка (рис. 2.1).

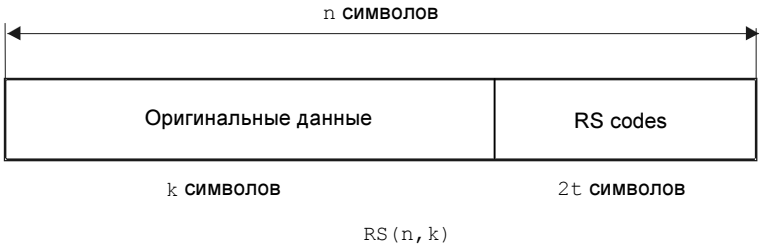


Рис. 2.1. Устройство кодового слова

Поскольку рассмотрение обоих типов кодировщиков заняло бы слишком много места, сосредоточим свое внимание на одних лишь систематических кодировщиках, как на наиболее популярных.

Архитектурно, кодировщик представляет собой совокупность *сдвиговых регистров (shift registers)*, объединенных посредством *сумматоров* и *умножителей*, функционирующих по правилам арифметики Галуа. Сдвиговый регистр (иначе называемый регистром сдвига) представляет последовательность ячеек памяти, называемых *разрядами*, каждый из которых содержит один элемент поля Галуа GF(q). Содержащийся в разряде символ, покидая этот разряд, поступает на выходную линию. Одновременно с этим, разряд фиксирует символ, находящийся на его входной линии. Замещение символов происходит дискретно, в строго определенные промежутки времени, называемые *тактами*.

При аппаратной реализации сдвигового регистра его элементы могут быть объедены как последовательно, так и параллельно. При последовательном объединении пересылка одного m-разрядного символа потребует *m* тактов, в то время как при параллельном она осуществляется всего за один такт.

Низкая эффективность программных реализаций кодировщиков Рида-Соломона объясняется тем, что разработчик не может осуществлять параллельное объединение элементов сдвигового регистра и вынужден работать с той шириной разрядности, которую "навязывает" архитектура данной машины. Однако создать 4-элементный 8-битный регистр сдвига параллельного типа на процессорах семейства IA-32 вполне реально.

Цепи, основанные на регистрах сдвига, обычно называют *фильтрами*. Блок-схема фильтра, осуществляющего деление полинома на константу, приведена на рис. 2.2. Пусть вас не смущает тот факт, что деление реализуется посредством умножения и сложения. Данный прием базируется на вычислении системы двух рекуррентных равенств с помощью формулы деления полинома на константу посредством умножения и сложения:

$$Q^{(r)}(x) = Q^{(r-1)}(x) + R_{n-r}^{(r-1)} x^{k-r}$$

$$R^{(r)}(x) = R^{(r-1)}(x) - R_{n-r}^{(r-1)} x^{k-r} g(x) \qquad (2.1)$$

Здесь: $Q^{(r)}(x)$ и $R^{(r)}(x)$ — соответственно, частное и остаток на *r*-шаге рекурсии. Поскольку сложение и вычисление, выполняемое по модулю два, тождественны друг другу, для реализации делителя нам достаточно иметь всего два устройства — устройство сложения и устройство умножения, а без устройства вычитания можно обойтись.

После *n*-сдвигов на выходе регистра появляется *частное*, а в самом регистре окажется *остаток*, который и представляет собой рассчитанные символы четности (они же — коды Рида-Соломона), а коэффициенты умножения с $g0$ по $g(2t - 1)$ напрямую соответствуют коэффициентам умножения порожденного полинома.

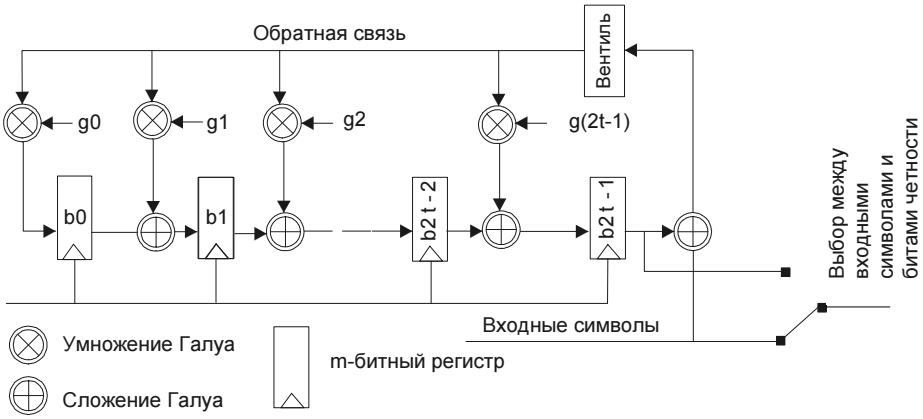


Рис. 2.2. Устройство простейшего кодировщика Рида-Соломона

Простейший пример программной реализации такого фильтра приведен в листинге 2.9. Это законченный кодировщик Рида-Соломона, вполне пригодный для практического использования. Конечно, при желании его можно было бы и улучшить, но тогда неизбежно пострадала бы наглядность и компактность листинга.

Листинг 2.9. Исходный текст простейшего кодировщика Рида-Соломона

```
/*---------------------------------------------------------------------------
 *
 *            кодировщик Рида-Соломона
 *            ========================
 *
 *     кодируемые  данные  передаются  через массив  data[i], где i=0..(k-1),
 * а сгенерированные  символы  четности  заносятся в  массив b[0]..b[2*t-1].
 * Исходные и результирующие данные должны быть представлены в полиномиальной
 * форме (т. е. в обычной форме машинного представления данных).
 *     кодирование производится с использованием сдвигового feedback-регистра,
 * заполненного  соответствующими  элементами  массива  g[]  с  порожденным
 * полиномом  внутри,  процедура  генерации  которого  уже  обсуждалась  в
 * предыдущей главе.
 *     сгенерированное кодовое слово описывается следующей формулой:
 * c(x) = data(x)*x^(n-k) + b(x), где ^ означает возведение в степень
 *
 *                                  на основе исходных текстов
 *                                  Simon'a Rockliff'a, от 26.06.1991
```

```
 *                                    распространяемых по лицензии GNU
-----------------------------------------------------------------------------*/
encode_rs()
{
    int i, j;
    int feedback;

    // инициализируем поле бит четности нулями
    for (i = 0; i < n - k; i++) b[i] = 0;

    // обрабатываем все символы
    // исходных данных справа налево
    for (i = k - 1; i >= 0; i--)
    {
         // готовим (data[i] + b[n - k -1]) к умножению на g[i]
         // т. е. складываем очередной "захваченный" символ исходных
         // данных с младшим символом битов четности (соответствующего
         // "регистру" b2t-1, см. рис.2.2) и переводим его в индексную
         // форму, сохраняя результат в регистре feedback
         // как мы уже говорили, сумма двух индексов есть произведение
         // полиномов
         feedback = index_of[data[i] ^ b[n - k - 1]];

         // есть еще символы для обработки?
         if (feedback != -1)
         {
              // осуществляем сдвиг цепи bx-регистров
              for (j=n-k-1; j>0; j--)
                    // если текущий коэффициент g - это действительный
                    // (т.е. ненулевой коэффициент, то
                    // умножаем feedback на соответствующий g-коэффициент
                    // и складываем его со следующим элементов цепочки
                    if (g[j]!=-1) b[j]=b[j-1]^alpha_to[(g[j]+feedback)%n];
              else
                    // если текущий коэффициент g - это нулевой коэффициент,
                    // выполняем один лишь сдвиг без умножения, перемещая
                    // символ из одного m-регистра в другой
                    b[j] = b[j-1];

              // закольцовываем выходящий символ в крайний левый b0-регистр
```

```
                b[0] = alpha_to[(g[0]+feedback)%n];
        }
        else

        {       // деление завершено,
                // осуществляем последний сдвиг регистра,
                // на выходе регистра будет частное, которое теряется,
                // а в самом регистре - искомый остаток
                for (j = n-k-1; j>0; j--) b[j] = b[j-1] ; b[0] = 0;
        }
    }
}
```

Декодер (decoder)

Декодирование кодов Рида-Соломона представляет собой довольно сложную задачу, решение которой выливается в громоздкий, запутанный и чрезвычайно ненаглядный программный код, требующий от разработчика обширных знаний во многих областях высшей математики. Типовая схема декодирования, получившая название *авторегрессионного спектрального метода декодирования*, состоит из следующих шагов:

1. Вычисления синдрома ошибки (синдромный декодер).
2. Построения полинома ошибки, осуществляемого либо посредством высокоэффективного, но сложно реализуемого алгоритма Берлекэмпа-Месси, либо посредством простого, но тормозного Евклидового алгоритма.
3. Нахождения корней данного полинома, обычно решающегося лобовым перебором (алгоритм Ченя).
4. Определения характера ошибки, сводящегося к построению битовой маски, вычисляемой на основе обращения алгоритма Форни или любого другого алгоритма обращения матрицы.
5. Наконец, исправления ошибочных символов путем наложения битовой маски на информационное слово и последовательного инвертирования всех искаженных битов посредством операции `XOR` .

Следует отметить, что данная схема (рис. 2.3) декодирования не единственная и вероятно, даже не самая лучшая, но зато универсальная. Всего же существует около десятка различных схем декодирования абсолютно непохожих друг на друга и выбираемых в зависимости от того, какая часть декодера реализуется программно, а какая аппаратно.

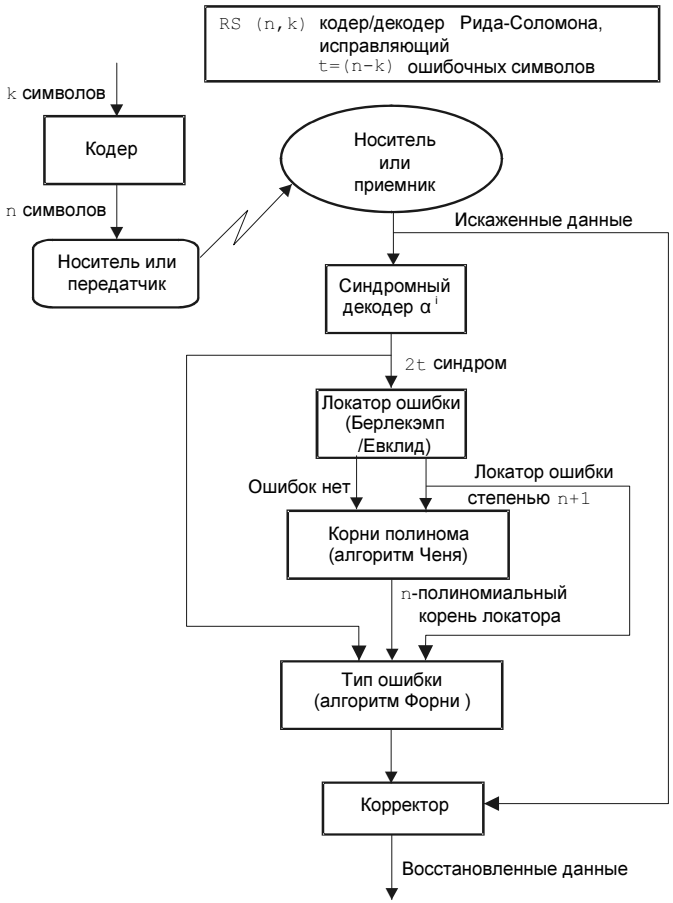


Рис. 2.3. Схема авторегрессионного спектрального декодера корректирующих кодов Рида-Соломона

Синдромный декодер

Грубо говоря, *синдром есть остаток деления декодируемого кодового слова c(x) на порожденный полином g(x)*, и, если этот остаток равен нулю, кодовое слово считается неискаженным. Ненулевой остаток свидетельствует о наличии по меньшей мере одной ошибки. Остаток от деления дает многочлен, не зависящий от исходного сообщения и определяемый исключительно характером ошибки.

Примечание

Syndrome (синдром) — греческое слово, обозначающее совокупность признаков и/или симптомов, характеризующих заболевание.

Принятое кодовое слово $\boldsymbol{v}$ с компонентами $\boldsymbol{v_i = c_i + e_i}$, где $i = 0, \quad n - 1$, представляет собой сумму кодового слова $\boldsymbol{c}$ и вектора ошибок $\boldsymbol{e}$. Цель декодирования состоит в очистке кодового слова от вектора ошибки, описываемым полиномом синдрома и вычисляемым по формуле $\boldsymbol{S_j = v(\alpha^{j + j0 - 1})}$, где j изменяется от 1 до $2t$, а α представляет собой примитивный член "альфа", который мы уже обсуждали в предыдущем разделе. Да, мы снова выражаем функцию деления через умножение, поскольку деление — крайне неэффективная в смысле производительности операция.

Блок-схема устройства, осуществляющего вычисление синдрома, приведена на рис. 2.4. Как видно, она представляет собой типичный фильтр (сравните ее со схемой рис. 2.2), а потому ни в каких дополнительных пояснениях не нуждается.

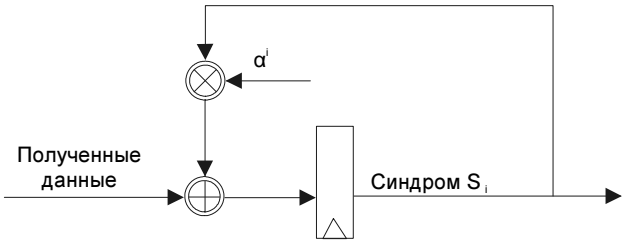


Рис. 2.4. Блок-схема цепи вычисления синдрома

Вычисление синдрома ошибки происходит итеративно, так что вычисление результирующего полинома (также называемого ответом от английского "answer") завершается непосредственно в момент прохождения последнего символа четности через фильтр. Всего требуется $2t$ циклов "прогона" декодируемых данных через фильтр, — по одному прогону на каждый символ результирующего полинома.

Пример простой программной реализации синдромного декодера содержится в листинге 2.20, и он намного нагляднее его словесного описания.

Полином локатора ошибки

Полученный синдром описывает конфигурацию ошибки, но еще не говорит нам, какие именно символы полученного сообщения были искажены. Действительно, степень синдромного полинома, равная $2t$, много меньше степени полинома сообщения, равной n, и между их коэффициентами нет прямого соответствия. Полином, коэффициенты которого напрямую соответствуют коэффициентам искаженных символов, называется *полиномом локатора ошибки* и по общепринятому соглашению обозначается греческой буквой Λ (лямбда).

Если количество искаженных символов не превышает t, между синдромом и локатором ошибки существует следующее однозначное соответствие,

выражаемое следующей формулой: НОД $[x^{n-1}, E(x)] = \Lambda(x)$ и вычисление локатора сводится к задаче нахождения наименьшего общего делителя, успешно решенной еще Евклидом и элементарно реализуемой как на программном, так и на аппаратном уровне. Правда, за простоту реализации нам

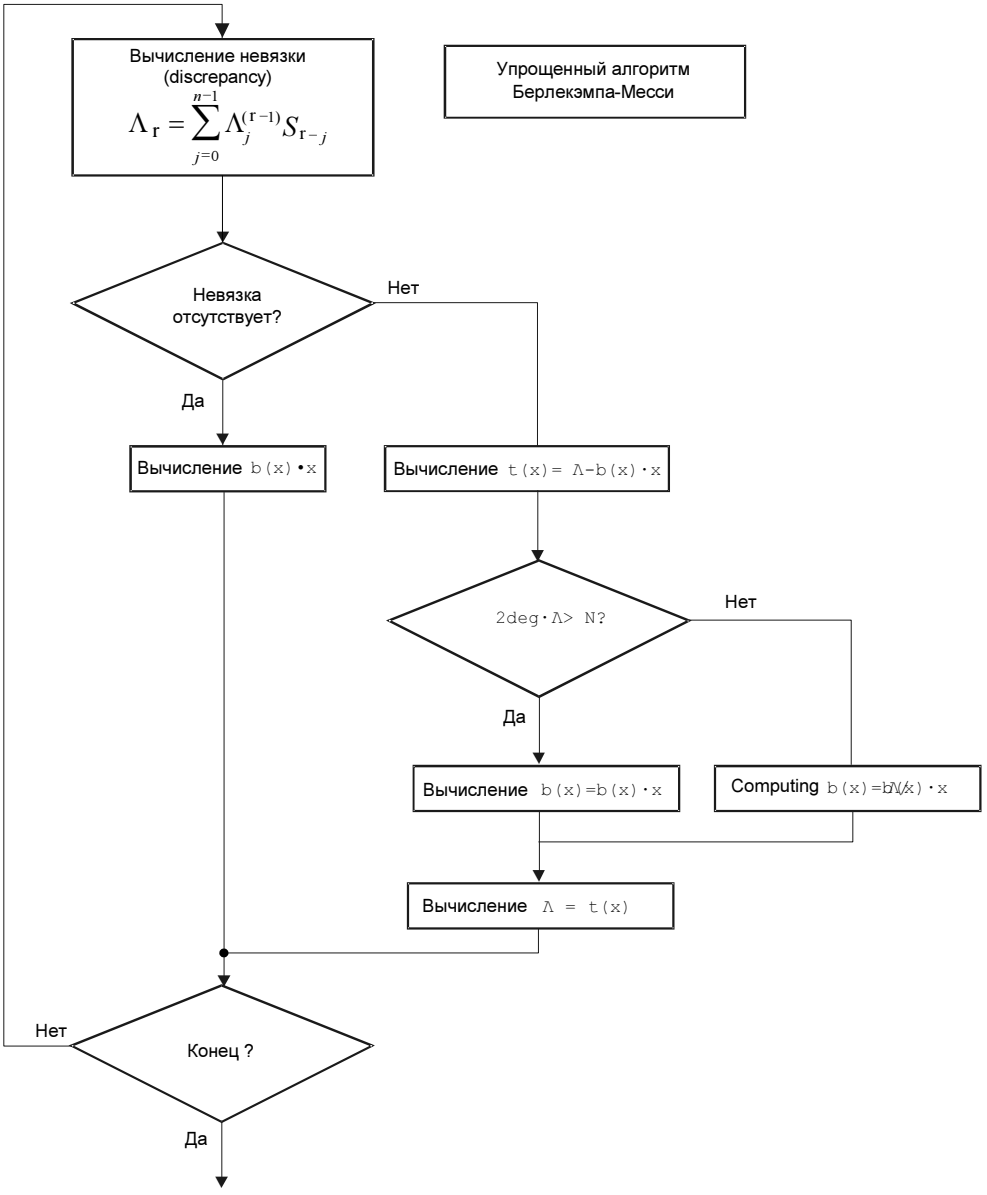


Рис. 2.5. Структурная схема-алгоритм Берлекэмпа-Месси

приходится расплачиваться производительностью, точнее *непроизводительностью* данного алгоритма, и на практике обычно применяют более эффективный, но и более сложный для понимания алгоритм *Берлекэмпа-Месси* (*Berlekamp-Massy*), подробно описанный Кнутом во втором томе *"Искусства программирования"* (см. также *"Теория и практика кодов, контролирующих ошибки"* Блейхута) и сводящейся к задаче построения цепи регистров сдвига с линейной обратной связью и по сути своей являющегося разновидностью авторегрессионого фильтра, множители в векторах которого и задают полином Λ.

Декодер, построенный по такому алгоритму, требует не более $3t$ операций умножения в каждой из итераций, количество которых не превышает $2t$. Таким образом, решение поставленной задачи укладывается всего в $6t^2$ операций умножения. Фактически, поиск локатора сводится к решению системы из $2t$ уравнений — по одному уравнению на каждый символ синдрома — с t-неизвестными. Неизвестные члены и есть позиции искаженных символов в кодовом слове v. Легко увидеть, что если количество ошибок превышает t, то система уравнений становится неразрешима и восстановить разрушенную информацию в этом случае не представляется возможным.

Блок-схема алгоритма Берлекэмпа-Месси приведена на рис. 2.5, а его законченная программа реализация содержится в листинге 1.2.

Корни полинома

Коль скоро полином локатора ошибки нам известен, то его корни определяют местоположение искаженных символов в принятом кодовом слове. Остается эти корни найти. Чаще всего для этого используется *процедура Ченя* (*Chien search*), аналогичная по своей природе обратному преобразованию Фурье и фактически сводящаяся к тупому перебору (brute force, exhaustive search) всех возможных вариантов. Все 2^m возможных символов один за другим подставляются в полином локатора в порядке "социалистической" очереди, и затем выполняется расчет полинома. Если результат обращается в ноль, то считается, что искомые корни найдены.

Восстановление данных

Итак, мы знаем, какие символы кодового слова искажены, но пока еще не готовы ответить на вопрос: *как именно* они искажены. Используя полином синдрома и корни полинома локатора, мы можем определить характер разрушений каждого из искаженных символов. Обычно для этой цели используется алгоритм *Форни* (*Forney*), состоящий из двух стадий: сначала путем свертки полинома синдрома полиномом локатора Λ мы получаем некоторый *промежуточный полином*, условно обозначаемый греческой буквой Ω (омега). Затем, на основе Ω-полинома, вычисляется *нулевая позиция ошибки* (*zero error location*), которая в свою очередь делится на производную от Λ-полинома. В результате получается битовая маска, каждый из установленных

битов которой соответствует искаженному биту, и для восстановления кодового слова в исходный вид все искаженные биты должны быть проинвертированы, что осуществляется посредством логической операции XOR.

На этом процедура декодирования принятого кодового слова считается законченной. Остается отсечь $n - k$ символов четности и полученное информационное слово будет готово к употреблению.

Исходный текст декодера

Далее в листинге 2.20 приводится исходный текст полноценного декодера Рида-Соломона, снабженный минимально разумным количеством комментариев. При возникновении трудностей в анализе этого листинга обращайтесь к блок-схемам, приведенным на рис. 2.3, 2.4 и 2.5 — они помогут.

Листинг 2.20. Исходный текст простейшего декодера Рида-Соломона

```
/*---------------------------------------------------------------------------
 *
 *                декодер Рида-Соломона
 *                =====================
 *
 *    процедура декодирования кодов Рида-Соломона состоит из нескольких шагов
 * сначала мы вычисляем 2t-символьный синдром путем постановки alpha**i в
 * recd(x), где recd - полученное кодовое слово, предварительно переведенное
 * в индексную форму. По факту вычисления recd(x) мы записываем очередной
 * символ синдрома в s[i], где i принимает значение от 1 до 2t, оставляя
 * s[0] равным нулю.
 *    затем, используя итеративный алгоритм Берлекэмпа (Berlekamp), мы
 * находим полином локатора ошибки - elp[i]. Если степень elp превышает
 * собой величину t, мы бессильны скорректировать все ошибки и ограничиваемся
 * выводом сообщения о неустранимой ошибке, после чего совершаем аварийный
 * выход из декодера. Если же степень elp не превышает t, мы подставляем
 * alpha**i, где i = 1..n в elp для вычисления корней полинома. Обращение
 * найденных корней дает нам позиции искаженных символов. Если количество
 * определенных позиций искаженных символов меньше степени elp, искажению
 * подверглось более чем t символов и мы не можем восстановить их.
 *     во всех остальных случаях восстановление оригинального содержимого
 * искаженных символов вполне возможно.
 *    в случае, когда количество ошибок заведомо велико, для их исправления
 * декодируемые символы проходят сквозь декодер без каких-либо изменений
 *
 *                                    на основе исходных текстов
```

```
 *                                    Simon'a Rockliff'a, от 26.06.1991
 *                                    распространяемых по лицензии GNU
-----------------------------------------------------------------------------*/
decode_rs()
{
    int i, j, u, q;
    int s[n-k+1];                     // полином синдрома ошибки
    int elp[n - k + 2][n - k];        // полином локатора ошибки лямбда
    int d[n-k+2];
    int l[n-k+2];
    int u_lu[n-k+2],

    int count=0, syn_error=0, root[t], loc[t], z[t+1], err[n], reg[t+1];

    // переводим полученное кодовое слово в индексную форму
    // для упрощения вычислений
    for (i = 0; i < n; i++) recd[i] = index_of[recd[i]];

    // вычисляем синдром
    //-----------------------------------------------------------------------------
    for (i = 1; i <= n - k; i++)
    {
         s[i] = 0;        // инициализация s-регистра (на его вход
                          // по умолчанию поступает ноль)

         // выполняем s[i] += recd[j]*ij
         // т.е. берем очередной символ декодируемых данных,
         // умножаем его на порядковый номер данного символа,
         // умноженный на номер очередного оборота, и складываем
         // полученный результат с содержимым s-регистра
         // по факту исчерпания всех декодируемых символов,
         // мы повторяем весь цикл вычислений опять - по одному
         // разу для каждого символа четности
         for (j=0; j<n; j++) if (recd[j]!=-1) s[i]^= alpha_to[(recd[j]+i*j)%n];

         if (s[i]!=0) syn_error=1;      // если синдром не равен нулю, взводим
                                        // флаг ошибки

         // преобразуем синдром из полиномиальной формы в индексную
         s[i] = index_of[s[i]];
    }
```

```
// коррекция ошибок
//-----------------------------------------------------------------------------
if (syn_error)            // если есть ошибки, пытаемся их скорректировать
{

     // вычисление полинома локатора лямбда
     //-------------------------------------------------------------------
     // вычисляем полином локатора ошибки через итеративный алгоритм
     // Берлекэмпа. Следуя терминологии Lin and Costello (см. "Error
     // Control Coding: Fundamentals and Applications" Prentice Hall 1983
     // ISBN 013283796) d[u] представляет собой μ ("мю"), выражающую
     // расхождение (discrepancy), где u = μ + 1 и μ есть номер шага
     // из диапазона от -1 до 2t. У Блейхута та же самая величина
     // обозначается Δ(x) ("дельта") и называется невязка.
     // l[u]    представляет собой степень elp для данного шага итерации,
     // u_l[u] представляет собой разницу между номером шага и степенью elp

     // инициализируем элементы таблицы
     d[0] = 0;                    // индексная форма
     d[1] = s[1];                 // индексная форма
     elp[0][0] = 0;               // индексная форма
     elp[1][0] = 1;               // полиномиальная форма

     for (i = 1; i < n - k; i++)
     {
           elp[0][i] = -1;        // индексная форма
           elp[1][i] = 0;         // полиномиальная форма
     }

     l[0] = 0; l[1] = 0; u_lu[0] = -1; u_lu[1] = 0; u = 0;

     do
     {
           u++;
           if (d[u] == -1)
           {
                 l[u + 1] = l[u];
                 for (i = 0; i <= l[u]; i++)
                 {
                        elp[u+1][i] = elp[u][i];
                        elp[u][i]   = index_of[elp[u][i]];
```

```
            }
        }
            else
        {
            // поиск слов с наибольшим u_lu[q], таких что d[q]!=0
            q = u - 1;
            while ((d[q] == -1) && (q>0)) q--;

            // найден первый ненулевой d[q]
            if (q > 0)
            {
                j=q ;
                do
            {
                j-- ;
                if ((d[j]!=-1) && (u_lu[q]<u_lu[j]))
                q = j ;
            } while (j>0);
        };

        // как только мы найдем q, такой что d[u]!=0
        // и u_lu[q] есть максимум
        // запишем степень нового elp-полинома
        if (l[u] > l[q]+u-q) l[u+1] = l[u]; else  l[u+1] = l[q]+u-q;


        // формируем новый elp(x)
        for (i = 0; i < n - k; i++) elp[u+1][i] = 0;
        for (i = 0; i <= l[q]; i++)
            if (elp[q][i]!=-1)
                elp[u+1][i+u-q]=alpha_to[(d[u]+n-d[q]+elp[q][i])%n];

        for (i=0; i<=l[u]; i++)
        {
            elp[u+1][i] ^= elp[u][i];

            // преобразуем старый elp
            // в индексную форму
            elp[u][i] = index_of[elp[u][i]];
        }
    }
    u_lu[u+1] = u-l[u+1];
```

```
      // формируем (u + 1)'ю невязку
      //-----------------------------------------------------------------------
      if (u < n-k)            // на последней итерации расхождение
      {                       // не было обнаружено

            if (s[u + 1]!=-1) d[u+1] = alpha_to[s[u+1]]; else d[u + 1] = 0;

            for (i = 1; i <= l[u + 1]; i++)
                  if ((s[u + 1 - i] != -1) && (elp[u + 1][i]!=0))
                  d[u+1] ^= alpha_to[(s[u+1-i]+index_of[elp[u+1][i]])%n];

            // переводим d[u+1] в индексную форму
            d[u+1] = index_of[d[u+1]];
      }
} while ((u < n-k) && (l[u+1]<=t));

// расчет локатора завершен
//-----------------------------------------------------------------------
u++ ;
if (l[u] <= t)
{                           // коррекция ошибок возможна

      // переводим elp в индексную форму
      for (i = 0; i <= l[u]; i++) elp[u][i] = index_of[elp[u][i]];


      // нахождение корней полинома локатора ошибки
      //-----------------------------------------------------------------------
      for (i = 1; i <= l[u]; i++) reg[i] = elp[u][i]; count = 0;

      for (i = 1; i <= n; i++)
      {
            q = 1 ;
            for (j = 1; j <= l[u]; j++)
            if (reg[j] != -1)
            {
                  reg[j] = (reg[j]+j)%n;
                  q ^= alpha_to[reg[j]];
            }

            if (!q)
```

```
        {       // записываем корень и индекс позиции ошибки
                root[count] = i;
                loc[count] = n-i;
                count++;
        }
}

if (count == l[u])
{               // нет корней - степень elp < t ошибок

        // формируем полином z(x)
        for (i = 1; i <= l[u]; i++)         // Z[0] всегда равно 1
        {
                if ((s[i]!=-1) && (elp[u][i]!=-1))
                        z[i] = alpha_to[s[i]] ^ alpha_to[elp[u][i]];
                else
                        if ((s[i]!=-1) && (elp[u][i]==-1))
                                z[i] = alpha_to[s[i]];
                        else
                                if ((s[i]==-1) && (elp[u][i]!=-1))
                                        z[i] = alpha_to[elp[u][i]];
                                else
                                                z[i] = 0 ;
        for (j=1; j<i; j++)
                if ((s[j]!=-1) && (elp[u][i-j]!=-1))
                        z[i] ^= alpha_to[(elp[u][i-j] + s[j])%n];

        // переводим z[i] в индексную форму
        z[i] = index_of[z[i]];
}

// вычисление значения ошибок в позициях loc[i]
//-----------------------------------------------------------------------
for (i = 0; i<n; i++)
{
        err[i] = 0;

        // переводим recd[] в полиномиальную форму
        if (recd[i]!=-1) recd[i] = alpha_to[recd[i]]; else recd[i] = 0;
}

// сначала вычисляем числитель ошибки
```

```
        for (i = 0; i < l[u]; i++)
        {
              err[loc[i]] = 1;
              for (j=1; j<=l[u]; j++)
                    if (z[j]!=-1)
                          err[loc[i]] ^= alpha_to[(z[j]+j*root[i])%n];

              if (err[loc[i]]!=0)
              {
                    err[loc[i]] = index_of[err[loc[i]]];
                    q = 0 ;       // формируем знаменатель коэффициента ошибки
                    for (j=0; j<l[u]; j++)
                          if (j!=i)
                                q+=index_of[1^alpha_to[(loc[j]+root[i])%n]];

                    q = q % n; err[loc[i]] = alpha_to[(err[loc[i]]-q+n)%n];

                    // recd[i] должен быть в полиномиальной форме
                    recd[loc[i]] ^= err[loc[i]];
              }
        }
    }
        else        // нет корней,
                    // решение системы уравнений невозможно, т.к. степень elp >= t
    {
        // переводим recd[] в полиномиальную форму
        for (i=0; i<n; i++)
              if (recd[i]!=-1) recd[i] = alpha_to[recd[i]];
        else
              recd[i] = 0;        // выводим информационное слово как есть
    }
        else                      // степень elp > t, решение невозможно
    {
    // переводим recd[] в полиномиальную форму
    for (i=0; i<n; i++)
        if (recd[i]!=-1)
              recd[i] = alpha_to[recd[i]] ;
        else
              recd[i] = 0 ;       // выводим информационное слово как есть
    }
        else                      // ошибок не обнаружено
    for (i=0;i<n;i++) if(recd[i]!=-1)recd[i]=alpha_to[recd[i]];else recd[i]=0;
}
```

Интерфейс с библиотечкой ElByECC.DLL

Программная реализация кодера/декодера Рида-Соломона, приведенная в листингах 2.1—2.2, достаточно наглядна, но крайне непроизводительна и нуждается в оптимизации. Как альтернативный вариант можно использовать готовые библиотеки от сторонних разработчиков, входящие с состав программных комплексов, так или иначе связанных с обработкой корректирующих кодов Рида-Соломона. Это и утилиты прожига/копирования/восстановления лазерных дисков, и драйвера ленточных накопителей (от стримера до Арвида), и различные телекоммуникационные комплексы и т. д.

Как правило, все эти библиотеки являются неотъемлемой частью самого программного комплекса и потому никак не документируются. Причем восстановление прототипов интерфейсных функций представляет весьма нетривиальную задачу, требующую от исследователя не только навыков дизассемблирования, но и знаний высшей математики, иначе смысл всех битовых манипуляций останется совершенно непонятным.

Насколько законно подобное дизассемблирование? Да, дизассемблирование сторонних программных продуктов действительно запрещено, но тем не менее оно *законно*. Здесь уместно провести аналогию со вскрытием пломб вашего телевизора, влекущее потерю гарантии, но отнюдь не приводящее к уголовному преследованию. Также никто не запрещает вызывать функции чужой библиотеки из своей программы. Нелегально распространять эту библиотеку в составе вашего программного обеспечения, действительно, нельзя, но что мешает вам попросить пользователя установить данную библиотеку самостоятельно?

Ниже приводится описание важнейших функций библиотеки ElByECC.DLL, входящей в состав известного копировщика защищенных лазерных дисков Clone CD, условно-бесплатную копию которого можно скачать с сайта по адресу: **http://www.elby.ch/**. Сам CloneCD проработает всего лишь 21 день, а затем потребует регистрации, однако на продолжительность использования библиотеки ElByECC.DLL не наложено никаких ограничений.

Усилиями хакера по имени МЫЩЪХ был создан h-файл, содержащий прототипы основных функций библиотеки ElByECC.DLL, специальная редакция которого была любезно предоставлена им для настоящей книги.

Несмотря на то, что библиотека ElByECC.DLL ориентирована на работу с секторами лазерных дисков, она может быть приспособлена и для других целей, например, построения отказоустойчивых дисковых массивов, о которых говорилось в предыдущем разделе.

Краткое описание основных функций библиотеки приводится далее.

Подключение библиотеки ElByECC.DLL к своей программе

Существует по меньшей мере два способа подключения динамических библиотек к вашим программам. При динамической компоновке, адреса требуемых функций определяются посредством вызова `GetProcAddress`, причем сама библиотека ElByECC.DLL должна быть предварительно загружена с помощью `LoadLibray`. Это может выглядеть например так, как показано в листинге 2.21 (обработка ошибок для простоты опущена).

Листинг 2.21. Динамическая загрузка библиотеки ElByECC.DLL

```
HANDLE h;
int (__cdecl *CheckECCAndEDC_Mode1) (char *userdata, char *header, char *sector);

h=LoadLibrary("ElbyECC.dll");
CheckECCAndEDC_Mode1 = GetProcAddress(h, "CheckECCAndEDC_Mode1");
```

Статическая компоновка предполагает наличие специального lib-файла, который может быть автоматически сгенерирован утилитой implib из пакета Borland C++ любой подходящей версии, представляющей собой утилиту командной строки, вызываемую так: "`implib.exe -a ElByECC.lib ElByECC.lib`".

Функция *GenECCAndEDC_Mode1*

Функция `GenECCAndEDC_Mode1` осуществляет генерацию корректирующих кодов на основе 2048-байтового блока пользовательских данных и имеет следующий прототип (листинг 2.22).

Листинг 2.22. Прототип функции GenECCAndEDC_Mode1

```
GenECCAndEDC_Mode1(char *userdata_src,      // указатель на массив из 2048 байт
            char *header_src,               // указатель на заголовок
            struct RAW_SECTOR_MODE1 *raw_sector_mode1_dst)
```

Где:

- `userdata_src` — указатель на 2048-байтовый блок пользовательских данных, для которых необходимо выполнить расчет корректирующих кодов. Сами пользовательские данные в процессе выполнения функции остаются неизменными и автоматически копируются в буфер целевого сектора, где к ним добавляется 104 + 172 байт четности и 4 байта контрольной суммы;

- `header_src` — указатель на 4-байтовый блок, содержащий заголовок сектора. Первые три байта занимает абсолютный адрес, записанный в BCD-форме, а четвертый байт отвечает за тип сектора, которому необходимо присвоить значение 1, и соответствующий режиму "корректирующие коды задействованы";
- `raw_sector_mode1_dst` — указатель на 2352-байтовый блок, в который будет записан сгенерированный сектор, содержащий 2048-байт пользовательских данных и 104 + 172 байт корректирующих кодов вместе с 4 байтами контрольной суммы и представленный структурой, приведенной в листинге 2.23.

Листинг 2.23. Структура "сырого" сектора

```
struct RAW_SECTOR_MODE1
{
    BYTE        SYNC[12];           // синхрогруппа
    BYTE        ADDR[3];            // абс. адрес сектора
    BYTE        MODE;               // тип сектора
    BYTE        USER_DATA[2048];    // пользовательские данные
    BYTE        EDC[4];             // контрольная сумма
    BYTE        ZERO[8];            // нули (не используется)
    BYTE        P[172];             // P-байты четности
    BYTE        Q[104];             // Q-байты четности
    };
```

При успешном завершении функция возвращает ненулевое значение и ноль в противном случае.

Функция *CheckSector*

Функция `CheckSector` (листинг 2.24) осуществляет проверку целостности сектора по контрольной сумме и при необходимости выполняет его восстановление по избыточным кодам Рида-Соломона.

Листинг 2.24. Прототип функции CheckSector

```
CheckSector(struct RAW_SECTOR *sector,     // указатель на секторный буфер
        int DO);                           // только проверка/лечение
```

Где:

- `sector` — указатель на 2352-байтовый блок данных, содержащий подопытный сектор. Лечение сектора осуществляется в "живую", т. е. непо-

средственно по месту возникновения ошибки. Если количество разрушенных байт превышает корректирующие способности кодов Рида-Соломона, исходные данные остаются неизменными;

- `DO` — флаг, нулевое значение которого указывает на запрет модификации сектора. Другими словами, соответствует режиму TEST ONLY. Ненулевое значение разрешает восстановление данных, если они действительно подверглись разрушению.

При успешном завершении функция возвращает ненулевое значение и ноль, если сектор содержит ошибку (в режиме TEST ONLY) или если данные восстановить не удалось (при вызове функции в режиме лечения). Для предотвращения возможной неоднозначности рекомендуется вызывать данную функцию в два приема. Первый раз — в режиме тестирования для проверки целостности данных, и второй раз — в режиме лечения (если это необходимо).

Финал

В листинге 2.25 приведен законченный пример использования корректирующих кодов на практике, пригодный для решения реальных практических задач.

Листинг 2.25. Пример вызова функций библиотеки ElByECC.DLL из своей программы

```
/*-----------------------------------------------------------------------------
 *
 *                демонстрация ElByECC.DLL
 *                ========================
 *
 *    данная программа  демонстрирует  работу  с  библиотекой  ElByECC.DLL,
 * генерируя избыточные коды Рида-Соломона на основе пользовательских данных,
 * затем умышленно искажает их и вновь  восстанавливает.
 *    количество разрушаемых байтов передается в первом параметре командной
 * строки (по умолчанию - 6)
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include "ElByECC.h"                          // декомпилировано МЫЩЬХем

#define _DEF_DMG    6                         // рушить по умолчанию
#define N_BYTES_DAMAGE    ((argc>1)?atol(argv[1]):_DEF_DMG)    // сколько байт
                                                               // рушить?
```

```
main(int argc, char **argv)
{
    int a;
    char stub_head[HEADER_SIZE];                // заголовок сектора
    char user_data[USER_DATA_SIZE];             // область польз. данных

    struct RAW_SECTOR_MODE1 raw_sector_for_damage;       // сектор для искажений
    struct RAW_SECTOR_MODE1 raw_sector_for_compre;       // контрольная копия сект.

    // TITLE
    //-----------------------------------------------------------------------
    printf("= ElByECC.DLL usage demo example by KK\n");

    // инициализация пользовательских данных
    //-----------------------------------------------------------------------
    printf("user data initialize...............");
    for (a = 0; a < USER_DATA_SIZE; a++) user_data[a] = a;   // user_data  init
    memset(stub_head, 0, HEADER_SIZE); stub_head[3] = 1;     // src header init
    printf("+OK\n");

    // генерация кодов Рида-Соломона на основе пользовательских данных
    //-----------------------------------------------------------------------
    printf("RS-code generate...................");
    a = GenECCAndEDC_Mode1(user_data, stub_head, &raw_sector_for_damage);
    if (a == ElBy_SECTOR_ERROR) { printf("-ERROR!\x7\n"); return -1;}
    memcpy(&raw_sector_for_compre, &raw_sector_for_damage, RAW_SECTOR_SIZE);
    printf("+OK\n");

    // умышленное искажение пользовательских данных
    //-----------------------------------------------------------------------
    printf("user-data %04d bytes damage........", N_BYTES_DAMAGE);
    for (a=0;a<N_BYTES_DAMAGE;a++) raw_sector_for_damage.USER_DATA[a]^=0xFF;
    if(!memcmp(&raw_sector_for_damage,&raw_sector_for_compre,RAW_SECTOR_SIZE))
         printf("-ERR: NOT DAMAGE YET\n"); else printf("+OK\n");

    // проверка целостности пользовательских данных
    //-----------------------------------------------------------------------
    printf("user-data check....................");
    a = CheckSector((struct RAW_SECTOR*)&raw_sector_for_damage,ElBy_TEST_ONLY);
    if (a==ElBy_SECTOR_OK){
         printf("-ERR:data not damage\x7\n");return -1;}printf(".data damge\n");
```

```
    // восстановление пользовательских данных
    //-----------------------------------------------------------------------
    printf("user-data recorver.................");
    a = CheckSector((struct RAW_SECTOR*)&raw_sector_for_damage, ElBy_REPAIR);
    if (a == ElBy_SECTOR_ERROR) {
         printf("-ERR: NOT RECORVER YET\x7\n"); return -1; } printf("+OK\n");

    // проверка успешности восстановления
    //-----------------------------------------------------------------------
    printf("user-data recorver check...........");
    if(memcmp(&raw_sector_for_damage,&raw_sector_for_compre,RAW_SECTOR_SIZE))
         printf("-ERR: NOT RECORVER YET\x7\n"); else printf("+OK\n");

    printf("+OK\n");
    return 1;
}
```

Часть II

МЕТОДЫ НИЗКОУРОВНЕВОГО УПРАВЛЕНИЯ ПРИВОДАМИ

Глава 3. Практические советы по восстановлению системы в боевых условиях

Глава 4. Интерфейсы взаимодействия с оборудованием

Глава 5. Способы разоблачения защитных механизмов

"Как правильно уложить парашют".

Пособие. Издание 2-е, исправленное

Секторный уровень взаимодействия всегда привлекал как создателей защитных механизмов, так и разработчиков утилит, предназначенных для копирования защищенных дисков. Еще бо́льшие перспективы открывает чтение/запись "сырых" (RAW) секторов — это наиболее низкий уровень общения с диском, какой только штатные приводы способны поддерживать. Большинство защитных механизмов именно так, собственно, и работает. Одни из них прячут ключевую информацию в каналы подкода, другие тем или иным образом искажают ECC/EDC-коды, третьи используют нестандартную разметку и т. д.

Существует множество способов для работы с диском на секторном уровне, и далее описан с добрый десяток из них. Бо́льшая часть рассматриваемых здесь методик рассчитана исключительно на Windows NT/2000/XP и не работает в операционных системах Windows 9x, которым, по-видимому, придется разделить судьбу мамонтов, а потому интерес к ним стремительно тает как со стороны пользователей, так и со стороны программистов. Конечно, какое-то время они еще продержатся "на плаву", но в долгосрочной перспективе я бы не стал на них рассчитывать, особенно учитывая тот факт, что Windows 9x не в состоянии поддерживать многопроцессорные системы, а победоносное шествие Hyper-Threading уже не за горами.

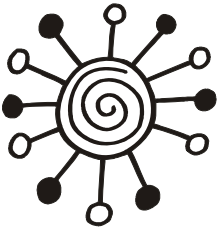
В силу того, что секторный уровень доступа к диску изначально ориентирован на ломателей защитных механизмов, данный раздел выкрашен ярко-хакерской краской и рассказывает не только о самих методиках низкоуровневого управления устройствами, но и описывает технику взлома каждого из них. Забегая вперед, отметим, что сломать можно все!

Замечание

На самом деле, это утверждение не совсем верно. Некоторые из защит от копирования на бытовом оборудовании не могут быть взломаны в принципе. В частности, защиты аудиодисков, основанные на искажении TOC, приводят к нечитабельности такого диска компьютерными приводами CD-ROM, но на аудиоплеерах, не слишком дотошно анализирующих TOC, такой диск воспроизводится вполне нормально. Единственный способ скопировать такой диск в цифровом виде — изменить код прошивки привода CD-ROM, убрав из него ряд "лишних" проверок, либо же развинтить привод для осуществления "горячей" замены диска.

Так что не стоит, право же, переоценивать стойкость механизмов, препятствующих несанкционированному копированию лазерных дисков. Если кому-то особо приспичит, вашу программу все равно взломают! Как? Вот об этом и рассказано далее. Как говорится: кто предупрежден — тот вооружен. Ну, а коль уж совсем невмоготу, то используйте прямой доступ к портам ввода/вывода с прикладного уровня. Нет, вы не ослышались — в Windows NT это действительно возможно, и далее рассказано, как это сделать.

Глава 3



Практические советы по восстановлению системы в боевых условиях

1. Во время исполнения ошибки имеют наивысший приоритет. Прервать исполнение ошибки может только другая, более активная ошибка.
2. Запросы операционной системы к ошибкам ошибками могут игнорироваться.
3. Запросы ошибок к операционной системе игнорироваться не могут.
4. При работе с файлами ошибки могут пользоваться файловой системой базовой ОС и ее ошибками.
5. На ЭВМ с параллельной архитектурой может выполняться несколько ошибок одновременно.

"Теория ошибок". В. Тихонов

Приложения, недопустимые операции и все, все, все…

Низкоуровневая работа с оборудованием требует чрезвычайной собранности и внимания. Малейшая ошибка приводит к появлению "голубого экрана смерти" *(BSOD — Blue Screen Of Death)* или аварийному завершению одного

или нескольких приложений. У разработчиков драйверов и саперов есть много общего — ни та, ни другая профессия беспечности не прощает. Интерфейсы ASPI и SPTI, несмотря на свои высококровные оболочки (в просторечии именуемые *"врапперами"*) настроены столь же агрессивно и роняют систему по поводу и без. Пройдет немало времени, прежде чем вы научитесь писать стабильный и неприхотливый код, а до той поры залогом вашего выживания будет умение бороться с последствиями критических ошибок и всякого рода сбоев.

Различные операционные системы по разному реагируют на критические ошибки. Так, например, Windows NT резервирует два региона своего адресного пространства для выявления некорректных указателей. Один находится на самом "дне" карты памяти и предназначен для отлавливания нулевых указателей. Другой расположен между "кучей" и областью памяти, закрепленной за операционной системой. Он контролирует выход за пределы пользовательской области памяти и, вопреки расхожему мнению, никак не связан с функцией `WriteProcessMemory` (см. техническую заметку ID: Q92764 в MSDN). Оба региона занимают по 64 Кбайт, и всякая попытка доступа к ним расценивается системой как критическая ошибка. В Windows 9x имеется всего лишь один регион в 4 Кбайта, следящий за нулевыми указателями, поэтому по своим контролирующим способностям она значительно уступает Windows NT.

В Windows NT экран критической ошибки (рис. 3.1) содержит следующую информацию:

- адрес машинной инструкции, возбудившей исключение;
- словесное описание категории исключения (или его код, если категория исключения неизвестна);
- параметры исключения (адрес недействительной ячейки памяти, род операции и т. д.).

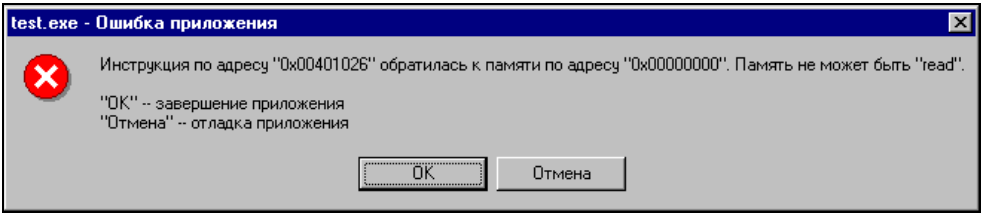


Рис. 3.1. Сообщение о критической ошибке, выдаваемое операционной системой Windows 2000

Операционные системы семейства Windows 9x в этом отношении намного более информативны (рис. 3.2) и помимо категории исключения выводят содержимое регистров ЦП на момент сбоя, состояние стека и байты памяти

по адресу CS:EIP (т. е. текущему адресу исполнения). Впрочем, наличие "Доктора Ватсона" (о нем — далее) стирает различие между двумя системами, и потому можно говорить лишь об удобстве и эргономике Windows 9x, сразу предоставляющей весь минимум необходимых сведений, в то время как в Windows NT отчет об ошибке создается отдельной утилитой.

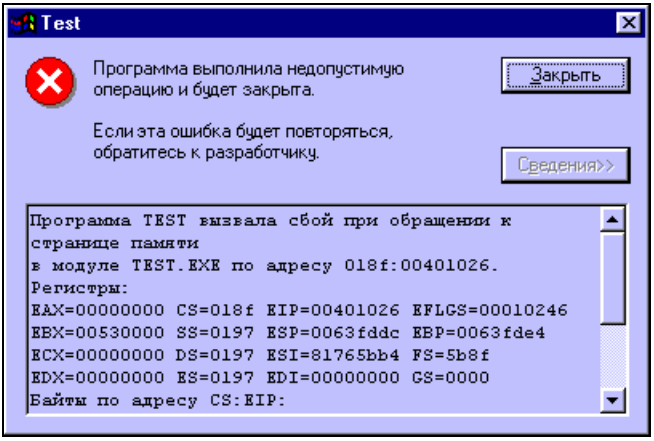


Рис. 3.2. Сообщение о критической ошибке, выдаваемое операционной системой Windows 98

Если никакой из отладчиков в системе не установлен, то окно о критической ошибке имеет всего лишь одну кнопку — кнопку **OK**, нажатие которой приводит к аварийному закрытию "политнекорректного" приложения. При желании окно критической ошибки можно оснастить кнопкой **Отмена** (Cancel), запускающей отладчик или иную утилиту анализа ситуации. Важно понять, что кнопка **Отмена** отнюдь не отменяет автоматическое закрытие приложения, но при некоторой сноровке вы можете устранить "пробоину" вручную, продолжив нормальную работу.

Запустите "Редактор реестра" (Regedit) и перейдите в раздел `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug`. Если такого раздела нет, то создайте его самостоятельно. Строковой параметр `Debugger` задает путь к файлу отладчика со всеми необходимыми ключами; строковой параметр `Auto` указывает, должен ли отладчик запускаться автоматически (значение `1`) или предлагать пользователю свободу выбора (`0`). Наконец, двойное слово параметра `UserDebuggerHotKey` специфицирует скэн-код "горячей" клавиши для принудительного вызова отладчика.

Доктор Ватсон

"Доктор Ватсон" является штатным обработчиком критических ошибок, входящим в базовый пакет поставки всех операционных систем семейства Windows. По своей природе он представляет собой статическое средство

сбора релевантной информации. Предоставляя исчерпывающий отчет о причинах сбоя, "Доктор Ватсон" в то же самое время лишен активных средств воздействия на некорректно работающие программы. Утихомирить разбушевавшееся приложение, заставив его продолжить свою работу с помощью одного "Доктора Ватсона", вы не сможете, и для этого вам придется прибегать к интерактивным отладчикам, одним из которых является Microsoft Visual Studio Debugger, входящий в состав одноименной среды разработки и рассматриваемый далее.

Считается, что "Доктор Ватсон" предпочтительнее использовать на рабочих станциях (точнее — на автоматизированных рабочих местах), а интерактивные средства отладки — на серверах. Дескать, во всех премудростях ассемблера пользователи все равно не разбираются, а вот на сервере продвинутый отладчик будет как нельзя кстати. Отчасти это действительно так, но не стоит игнорировать то обстоятельство, что далеко не все источники ошибок обнаруживаются статическими средствами анализа, к тому же интерактивные инструменты значительно упрощают процедуру анализа. С другой стороны, "Доктор Ватсон" достается нам даром, а все остальные программные пакеты приходится приобретать за дополнительную плату. Так что предпочтительный обработчик критических ошибок вы должны выбирать сами.

Для установки "Доктора Ватсона" отладчиком по умолчанию добавьте в реестр следующую запись (листинг 3.1) или запустите файл Drwtsn32.exe с ключом "−i" (для выполнения обоих действий вы должны иметь права администратора).

Листинг 3.1. Установка "Доктора Ватсона" отладчиком по умолчанию

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug]
"Auto"="1"
"Debugger"="drwtsn32 -p %ld -e %ld -g"
"UserDebuggerHotKey"=dword:00000000
```

Теперь возникновение критических ошибок программы станет сопровождаться генерацией отчета, составляемого "Доктором Ватсоном" и содержащим более или менее подробные сведения о характере ее происхождения (рис. 3.3).

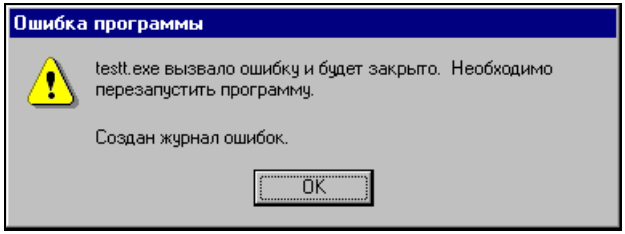


Рис. 3.3. Реакция "Доктора Ватсона" на критическую ошибку

Образец *дампа*, созданный "Доктором Ватсоном", приведен в листинге 3.2. Комментарии, добавленные автором, выделены серым цветом.

Листинг 3.2. Образец отчета "Доктора Ватсона" с комментариями автора

```
Исключение в приложении:
     Прил.:  (pid=612)
     ; pid процесса, в котором произошло исключение

     Время: 14.11.2003 @ 22:51:40.674
     ; время, когда произошло исключение

     Номер: c0000005 (нарушение прав доступа)
     ; код категории исключения
     ; расшифровку кодов исключений можно найти в WINNT.H
     ; входящим в состав SDK, прилагаемом к любому Windows-компилятору
     ; подробное описание всех исключений содержится в документации
     ; по процессорам Intel и AMD, бесплатно распространяемой их производителями
     ; (внимание: для перевода кода исключения операционной системы в
     ; вектор прерывания ЦП, вы должны обнулить старшее слово)
     ; в данном случае это 0x5 – попытка доступа к памяти по запрещенному адресу


*----> Сведения о системе <----*
     Имя компьютера: KPNC
     Имя пользователя: Kris Kaspersky
     Число процессоров: 1
     Тип процессора: x86 Family 6 Model 8 Stepping 6
     Версия Windows 2000: 5.0
     Текущая сборка: 2195
     Пакет обновления: None
     Текущий тип: Uniprocessor Free
     Зарегистрированная организация:
     Зарегистрированный пользователь: Kris Kaspersky
     ; краткие сведения о системе


*----> Список задач <----*
   0 Idle.exe
   8 System.exe
 232 smss.exe
```

```
…
1244 os2srv.exe
1164 os2ss.exe
1284 windbg.exe
1180 MSDEV.exe
1312 cmd.exe
 612 test.exe
1404 drwtsn32.exe
   0 _Total.exe

(00400000 - 00406000)
(77F80000 - 77FFA000)
(77E80000 - 77F37000)
; перечень загруженных DLL
; согласно документации, справа от адресов должны быть перечислены имена
; соответствующих модулей, однако практически все они так хорошо "замаскировались",
; что стали совершенно не видны. вытащить их имена из файла протокола все-таки
можно,
; но придется немного пошаманить (см. ниже "таблицу символов")

Копия памяти для потока 0x188
; ниже идет копия памяти потока, вызывавшего исключение

eax=00000064 ebx=7ffdf000 ecx=00000000 edx=00000064 esi=00000000 edi=00000000
eip=00401014 esp=0012ff70 ebp=0012ffc0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000202
; содержимое регистров и флагов

функция: <nosymbols>
; распечатка окрестной точки сбоя

     00400ffc 0000          add     [eax],al          ds:00000064=??
     ; записываем в ячейку, на которую ссылает EAX значение AL
     ; значение адреса ячейки, вычисленной Доктором Ватсоном, равно 64h
     ; что, очевидно, не соответствует действительности;
     ; Доктор Ватсон подставляет в выражение значение регистра EAX
     ; на момент возникновения сбоя, и это совсем не то значение, которое
     ; было в момент исполнения! к сожалению, чему был равен EAX в момент
     ; исполнения, ни нам, ни Доктору Ватсону не известно.

     00400ffe 0000         add     [eax],al               ds:00000064=??
     ; записываем в ячейку, на которую ссылает EAX, значение AL
```

```
     ; как? опять? что это  за бред?! вообще-то так кодируется
     ; последовательность 00 00 00 00, по всей видимости, являющаяся
     ; осколком некоторой машинной команды, неправильно интерпретированной
     ; дизассемблерным движком Доктора Ватсона;

     00401000 8b542408     mov     edx,[esp+0x8]          ss:00f8d547=????????
     ; загружаем в EDX аргумент функции
     ; какой именно аргумент - сказать невозможно, т.к. мы не знаем адрес
     ; стекового фрейма;

     00401004 33c9         xor     ecx,ecx
     ; обнуляем ECX

     00401006 85d2         test    edx,edx
     00401008 7e18         jle     00409b22
     ; если EDX == 0, прыгаем на адрес 409B22h

     0040100a 8b442408     mov     eax,[esp+0x8]          ss:00f8d547=????????
     ; загружаем уже упомянутый аргумент в регистр EAX

     0040100e 56           push    esi
     ; сохраняем ESI в стеке, перемещая тем самым указатель вершины стека
     ; на 4 байта вверх (в область младших адресов)

     0040100f 8b742408     mov     esi, [esp+0x8]         ss:00f8d547=????????
     ; загружаем в ESI очередной аргумент
     ; поскольку ESP был только что изменен, это совсем не тот аргумент,
     ; с которым мы имели дело ранее

     00401013 57           push    edi
     ; сохраняем регистр EDI в стеке

СБОЙ -> 00401014 0fbe3c31  movsx   edi,byte ptr [ecx+esi]  ds:00000000=??
     ; вот мы и добрались до инструкции, возбудившей исключение доступа
     ; она обращается к ячейке памяти, на которую указывает сумма
     ; регистров ECX и ESI
     ; а чему равно их значение? прокручиваем экран немного вверх и находим,
     ; что ECX и ESI равны 0, о чем Доктор Ватсон нам и сообщает: "ds:000000"
     ; отметим, что этой информации можно верить, поскольку подстановка
     ; эффективного адреса осуществлялась непосредственно в момент исполнения
     ; теперь вспомним, что ESI содержит копию переданного функции аргумента
     ; и что ECX был обнулен явно, следовательно, в выражении [ECX+ESI]
```

```
     ; регистр ESI - указатель, а ECX - индекс.
     ; раз ESI равен нулю, то нашей функции передали указатель на невыделенную
     ; область памяти. обычно это происходит либо вследствие алгоритмической
     ; ошибки в программе, либо вследствие исчерпания виртуальной памяти.
     ; к сожалению, Доктор Ватсон не осуществляет дизассемблирование
     ; материнской функции, и какой из двух предполагаемых вариантов правильный,
     ; нам остается лишь гадать… правда, можно дизассемблировать дамп памяти
     ; процесса (если, конечно, он был сохранен), но это уже не то…

     00401018 03c7         add     eax, edi
     ; сложить содержимое регистра EAX с регистром EDI и записать результат в EAX

     0040101a 41           inc     ecx
     ; увеличить ECX на единицу

     0040101b 3bca         cmp     ecx,edx
     0040101d 7cf5         jl      00407014
     ; до тех пор пока ECX < EDX, прыгать на адрес 407014
     ; (очевидно, мы имеем дело с циклом, управляемым счетчиком ECX)
     ; при интерактивной отладке мы могли бы принудительно выйти
     ; из функции, возвратив флаг ошибки, чтобы материнская функция
     ; (а с ней и вся программа целиком) могла продолжить свое выполнение
     ; и в этом случае потерянной окажется лишь последняя операция, но все
     ; остальные данные окажутся неискаженными;

     0040101f 5f           pop     edi
     00401020 5e           pop     esi
     00401021 c3           ret
     ; выходим из функции

*----> Обратная трассировка стека <----*
; содержимое стека на момент возникновения сбоя
; распечатывает адреса и параметры предыдущих выполняемых функций,
; при интерактивной отладке мы могли бы просто передать управление
; на одну из вышележащих функций, что эквивалентно возвращению в прошлое
; это только в реальной жизни разбитую чашку восстановить нельзя,
; в компьютерной вселенной возможно все!
FramePtr ReturnAd Param#1  Param#2  Param#3  Param#4  Function Name
; FramePtr:     указывает на значение фрейма стека,
;           выше (т.е. в более младших адресах) содержатся аргументы функции
;           ниже - ее локальные переменные
;
```

```
; ReturnAd:     бережно хранит адрес возврата в материнскую функцию
;          если здесь содержится мусор и обратная трассировка стека
;          начинает характерно шуметь, с высокой степенью вероятности
;          можно предположить, что мы имеем дело с ошибкой "срыва стека",
;          а возможно, и с попыткой атаки вашего компьютера
;
; Param#:     четыре первых параметра функции - именно столько параметров
;          Доктор Ватсон отображает на экране; это достаточно жесткое
;          ограничение - многие функции имеют десятки параметров и
;          четыре параметра еще ни о чем не говорят; однако недостающие
;          параметры легко вытащить из копии необработанного стека вручную,
;          достаточно лишь перейти по указанному в поле FramePtr адресу
;
; Func Name:     имя функции (если только его возможно определить); реально
;          отображает лишь имена функций, импортируемые из других DLL,
;          поскольку встретить коммерческую программу, откомпилированную
;          вместе с отладочной информацией, практически нереально
;
0012FFC0 77E87903 00000000 00000000 7FFDF000 C0000005 !<nosymbols>
0012FFF0 00000000 00401040 00000000 000000C8 00000100 kernel32!SetUnhandledExceptionFilter
; функции перечисляются в порядке их исполнения; самой последней исполнялась
; kernel32!SetUnhandledExceptionFilter функция, обрабатывающая данное исключение

*----> Копия необработанного стека <----*
; копия необработанного стека содержит стек таким, какой он есть
; очень помогает при обнаружении buffer overfull атак - весь shell-код,
; переданный злоумышленником, будет распечатан Доктором Ватсоном, и вам
; останется всего лишь опознать его (подробнее об этом рассказывается
; в моей книге "Техника сетевых атак")
0012ff70  00 00 00 00 00 00 00 00 - 39 10 40 00 00 00 00 00  ........9.@.....
0012ff80  64 00 00 00 f4 10 40 00 - 01 00 00 00 d0 0e 30 00  d.....@.......0.
…
00130090  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
001300a0  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

*----> Таблица символов <----*
; таблица символов содержит имена всех загруженных DLL вместе с именами
; импортируемых функций. используя эти адреса в качестве отправной точки,
; мы без труда сможем восстановить «перечень загруженных DLL»

ntdll.dll
```

```
77F81106 00000000   ZwAccessCheckByType
…
77FCEFB0 00000000   fltused

kernel32.dll
77E81765 0000003d   IsDebuggerPresent
…
77EDBF7A 00000000   VerSetConditionMask
;
; итак, возвращаемся к таблице загруженных DLL
; (00400000 - 00406000)  - это, очевидно, область памяти, занятая самой программой
; (77F80000 - 77FFA000)  - это KERNEL32.DLL
; (77E80000 - 77F37000)  - это NTDDL.DLL
```

Microsoft Visual Studio Debugger

При установке среды разработки Microsoft Visual Studio она регистрирует свой отладчик основным отладчиком критических ошибок по умолчанию. Это простой в использовании, но функционально ущербный отладчик, не поддерживающий даже такой банальной операции, как поиск hex-последовательности в оперативной памяти. Единственная "вкусность", отличающая его от продвинутого во всех отношениях Microsoft Kernel Debugger, — это возможность трассировки "упавших" процессов, выбросивших критическое исключение.

В опытных руках отладчик Microsoft Visual Studio Debugger способен творить настоящие чудеса, и одно из таких чудес — это возобновление работы приложений, совершивших недопустимую операцию и при нормальном течении событий аварийно завершаемых операционной системой без сохранения данных. В любом случае, интерактивный отладчик (коим Microsoft Visual Studio Debugger и является) предоставляет намного более подробную информацию о сбое и значительно упрощает процесс выявления источников его возникновения. К сожалению, тесные рамки данной главы (и без того далеко отошедшей от основной темы книги!) не позволяют изложить всю методику поиска неисправностей целиком и приходится ограничиваться лишь узким кругом наиболее интересных (и наименее известных!) вопросов *(см. разд. "Обитатели сумеречной зоны, или из морга в реанимацию" этой главы)*.

Для ручной установки Microsoft Visual Studio Debugger основным отладчиком критических ошибок добавьте в реестр данные, показанные в листинге 3.3.

Листинг 3.3. Установка Microsoft Visual Studio Debugger основным отладчиком критических ошибок

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug]
"Auto"="1"
"Debugger"="\"C:\\Prg Files\\MS VS\\Common\\MSDev98\\Bin\\msdev.exe\" -p %ld -e %ld"
"UserDebuggerHotKey"=dword:00000000
```

Листинг 3.4. Демонстрационная программа, вызывающая сообщение о критической ошибке

```
// функция возвращает сумму n символов типа char
// если ей передать null-pointer, она "упадет",
// хотя источник ошибки не в ней, а в аргументах,
// переданных материнской функцией
test(char *buf, int n)
{
     int a, sum;
     for (a = 0; a < n; a++) sum += buf[a];     // здесь возбуждается исключение
     return sum;
}

main()
{
     #define N     100
     char *buf = 0;                // инициализируем указатель на буфер

     /* buf = malloc(100); */      // "забываем" выделить память, здесь ошибка
     test(buf, N);                 // передаем null-pointer некоторой функции
}
```

Обитатели "сумеречной зоны", или из "морга в реанимацию"

Хотите узнать, как заставить приложение продолжить нормальную работу после появления сообщения о критической ошибке? Это действительно очень актуально. Представьте, что рухнуло приложение, содержащее уникальные и еще не сохраненные данные. По минимуму их придется набивать заново, по максимуму — они потеряны для вас навсегда. На рынке имеется некоторое количество утилит, нацеленных на эту задачу (взять те же Norton Utilities), но их интеллектуальность оставляет желать лучшего, и в среднем

они срабатывают один раз из десяти. В то же самое время, ручная реанимация программы воскрешает ее в 75—90% случаев.

Строго говоря, гарантированно восстановить работоспособность обрушившейся программы нельзя, равно как и невозможно выполнить откат тех действий, что предшествовали ее обрушению. В лучшем случае вам удастся сохранить свои данные на диск до того, как программа полностью потеряет нить управления и пойдет вразнос. Но и это неплохо!

Существует по меньшей мере три различных способа реанимации:

- *принудительный выход из функции, возбудившей исключение;*
- *"раскрутка" стека с передачей управления назад;*
- *передача управления на функцию обработки сообщений.*

Рассмотрим каждый из этих способов на примере приложения testt.exe.

Забегая вперед, отметим, что реанимации поддаются лишь те сбои, которые вызваны алгоритмическими, а не аппаратными ошибками (т. е. сбоем оборудования). Если информация, хранящаяся в оперативной памяти, оказалась искажена в результате физического дефекта последней, то восстановить работоспособность упавшего приложения, скорее всего, уже не удастся, хотя если сбой не затронул жизненно важные структуры данных, некоторая надежда на благополучный исход все-таки есть.

Принудительный выход из функции

Запускаем тестовую программу, набиваем в одном или нескольких окнах какой-нибудь текст, затем в меню **Help** выбираем пункт **About TestCEdit** и в появившемся диалоговом окне щелкаем по кнопке **make error**. Опля! Программа выбрасывает критическую ошибку, и, если мы нажмем на **OK**, все несохраненные данные необратимо погибнут, что никак не входит в наши планы. Однако при наличии предварительно установленного отладчика мы еще можем кое-что предпринять. Пусть для определенности это будет Microsoft Visual Studio Debugger.

Нажимаем кнопку **Отмена**, и отладчик немедленно дизассемблирует функцию, возбудившую исключение (листинг 3.5).

Листинг 3.5. Отладчик Microsoft Visual Studio Debugger дизассемблировал функцию, возбудившую исключение

```
0040135C    push    esi
0040135D    mov     esi,dword ptr [esp+8]
00401361    push    edi
00401362    movsx   edi,byte ptr [ecx+esi]
00401366    add     eax,edi
```

```
00401368     inc     ecx
00401369     cmp     ecx,edx
0040136B     jl      00401362
0040136D     pop     edi
0040136E     pop     esi
0040136F     ret     8
```

Проанализировав причину возникновения исключения (функции передан указатель на невыделенную память), мы приходим к выводу, что заставить функцию продолжить свою работу невозможно, поскольку структура передаваемых данных нам неизвестна. Приходится прибегать к принудительному возврату в материнскую функцию, не забыв при этом установить флаг ошибки, сигнализируя программе, что текущая операция не была выполнена. К сожалению, никаких общепринятых флагов ошибок не существует, и различные функции используют различные соглашения. Чтобы выяснить, как обстоят дела в данном конкретном случае, мы должны дизассемблировать материнскую функцию и определить, какой именно код ошибки она ожидает.

Переместив курсор в окно дампа, набьем в строке адреса название регистра указателя вершины стека — "ESP" и нажмем клавишу <Enter>. Содержимое стека тут же предстанет перед нашими глазами (листинг 3.6).

Листинг 3.6. Поиск адреса возврата из текущей функции (выделен полужирным шрифтом)

```
0012F488  0012FA64  0012FA64  004012FF
0012F494  00000000  00000064  00403458
0012F4A0  FFFFFFFF  0012F4C4  6C291CEA
0012F4AC  00000019  00000000  6C32FAF0
0012F4B8  0012F4C0  0012FA64  01100059
0012F4C4  006403C2  002F5788  00000000
0012F4D0  00640301  77E16383  004C1E20
```

Первые два двойных слова соответствуют машинным командам `POP EDI`/`POP ESI` и не представляют для нас совершенно никакого интереса. А вот следующее двойное слово содержит адрес выхода в материнскую процедуру (в приведенном листинге 3.6 оно выделено полужирным шрифтом). Как раз его-то нам и надо!

Нажимаем <Ctrl>+<D> и затем `0x4012FF`, отладчик послушно отображает следующий дизассемблерный текст (листинг 3.7).

Листинг 3.7. Дизассемблерный текст материнской функции

```
004012FA     call    00401350
004012FF     cmp     eax,0FFh
00401302     je      0040132D
00401304     push    eax
00401305     lea     eax, [esp+8]
00401309     push    405054h
0040130E     push    eax
0040130F     call    dword ptr ds:[4033B4h]
00401315     add     esp, 0Ch
00401318     lea     ecx, [esp+4]
0040131C     push    0
0040131E     push    0
00401320     push    ecx
00401321     mov     ecx, esi
00401323     call    00401BC4
00401328     pop     esi
00401329     add     esp,64h
0040132C     ret
0040132C
0040132D     push    0
0040132D     ; эта ветка получает управление, если Функция 401350h вернет FFh
0040132F     push    0
00401331     push    405048h
00401336     mov     ecx,esi
00401338     call    00401BC4
0040133D     pop     esi
0040133E     add     esp,64h
00401341     ret
```

Смотрите: если регистр EAX равен `FFh`, то материнская функция передает управление на ветку `40132Dh` и спустя несколько машинных команд завершает свою работу, передавая бразды правления функции более высокого уровня. Напротив, если EAX != `FFh`, то его значение передается функции `4033B4h`. Следовательно, мы можем предположить, что `FFh` — это флаг ошибки и есть. Возвращаемся в подопытную функцию, нажав <Ctrl>+<G> и "EIP", переходим в окно **Registers** и меняем значение регистра EAX на `FFh`.

Теперь необходимо найти подходящую точку возврата из функции. Просто перейти к машинной команде `RET` нельзя, поскольку перед выходом из функции следует в обязательном порядке сбалансировать стек или нас "выбросит" неизвестно куда и программа "обрушится" окончательно.

В общем случае число PUSH-команд должно в точности соответствовать количеству POP (также учитывайте, что PUSH DWORD X эквивалентен SUB ESP, 4, а POP DWORD X — ADD ESP, 4). Проанализировав дизассемблерный листинг функции, мы приходим к выводу, что для достижения гармонии добра и зла мы должны стащить с вершины стека два двойных слова, соответствующие машинным командам 40135C:PUSH ESI и 401361:PUSH EDI. Это достигается передачей управления по адресу 40136Dh, где "живут две добродушные" команды POP, приводящие стек в равновесное состояние. Подводим сюда курсор и уверенным щелчком правой клавиши мыши вызываем контекстное меню, среди пунктов которого выбираем **Set Next Statement**. Как вариант можно перейти в окно регистров и изменить значение регистра EIP с 401362h на 40136Dh.

Нажатием клавиши <F5> мы заставляем процессор продолжить выполнение программы и о чудо! Она действительно продолжает свою работу (незлобное ругательство на ошибку последней операции — не в счет!). Несохраненные данные спасены!

"Раскрутка" стека

Далеко не во всех случаях принудительный выход из функции оказывается возможным. Ряд критических сбоев затрагивает не одну, а сразу несколько вложенных функций, и тогда для реанимации программы мы должны совершить глубокий откат назад, продолжив выполнение программы с того места, где бы ее работоспособности ничто не угрожало. Точная глубина отката подбирается экспериментально и обычно составляет три — пять ступеней. Имейте в виду, что если вложенные функции модифицируют глобальные данные (например данные кучи), то попытка отката может привести к полному краху отлаживаемой программы, поэтому требуемую глубину отката желательно угадать с первого раза, придерживаясь правила: "лучше перебрать, чем недобрать". С другой стороны, чрезмерно глубокий откат ведет к потере всех несохраненных данных.

Процедура отката состоит из трех шагов:

- *построения дерева вызовов*;
- *определения координат стекового фрейма для каждого из них*;
- *восстановления регистрового контекста материнской функции.*

Хороший отладчик все это сделает за нас, и вам останется лишь записать в регистры EIP и ESP соответствующие значения. К сожалению, отладчик Microsoft Visual Studio Debugger к хорошим отладчикам не относится. Он довольно посредственно трассирует стек, пропуская FPO-функции (*Frame Point Omission* — функции с оптимизированным фреймом), и не сообщает координат стекового фрейма, "благодаря" чему самую трудоемкую часть работы нам приходится выполнять самостоятельно.

Впрочем, даже такой стек вызовов все же лучше, чем совсем ничего. Раскручивая его вручную, мы будем отталкиваться от того, что координаты

фрейма естественным образом определяются по адресу возврата. Допустим, содержимое окна **Call Stacks** выглядит так, как это показано в листинге 3.8.

Листинг 3.8. Содержимое окна Call Stacks отладчика Microsoft Visual Studio Debugger

```
TESTCEDIT! 00401362()
MFC42! 6c2922ae()
MFC42! 6c298fc5()
MFC42! 6c292976()
MFC42! 6c291dcc()
MFC42! 6c291cea()
MFC42! 6c291c73()
MFC42! 6c291bfb()
MFC42! 6c291bba()
```

Попробуем найти в стеке адреса `6C2922AEh` и `6C298FC5h`, соответствующие двум последним ступеням исполнения. Нажимаем <ATL>+<6> для перехода в окно дампа и, воспользовавшись "горячей" комбинацией клавиш <Ctr>+<G> в качестве базового адреса отображения, выбираем "ESP". Прокручивая окно дампа вниз, мы обнаруживаем оба адреса возврата (в приведенном далее листинге 3.9 они выделены рамкой).

Листинг 3.9. Содержимое стека после "раскрутки"

```
0012F488  0012FA64  0012FA64  004012FF ← 0040136F:ret 8 первый адрес возврата
0012F494  00000000  00000064  00403458 ← 00401328:pop esi
0012F4A0  FFFFFFFF  0012F4C4  6C291CEA
0012F4AC  00000019  00000000  6C32FAF0
0012F4B8  0012F4C0  0012FA64  01100059
0012F4C4  00320774  002F5788  00000000
0012F4D0  00320701  77E16383  004C1E20
0012F4DC  00320774  002F5788  00000000
0012F4E8  000003E8  0012FA64  004F8CD8
0012F4F4  0012F4DC  002F5788  0012F560
0012F500  77E61D49  6C2923D8  00403458 ← 0040132C:ret;
0012F50C  00000111  0012F540  6C2922AE ←6C29237E:pop ebx/pop ebp/ret 1Ch
0012F518  0012FA64  000003E8  00000000
0012F518  0012FA64  000003E8  00000000
0012F524  004012F0  00000000  0000000C
0012F530  00000000  00000000  0012FA64
```

```
0012F53C  000003E8  0012F564  6C298FC5
0012F548  000003E8  00000000  00000000
0012F554  00000000  000003E8  0012FA64
```

Ячейки памяти, лежащие выше адресов возврата, представляют собой значения регистров, сохраненные в стеке при входе в функцию и восстанавливаемые при ее завершении. Ячейки памяти, лежащие ниже адресов возврата, "оккупированы" аргументами функции (если, конечно, у функции есть аргументы) или же принадлежат локальным переменным материнской функции, если дочерняя функция не принимает никаких аргументов.

Возвращаясь к листингу 3.5, отметим, что два двойных слова, лежащие на верхушке стека, соответствуют машинным командам `POP EDI` и `POP ESI`, а следующий за ними адрес — `4012FFh` — это тот самый адрес, управление которому передается командой `40136Fh:RET 8`. Для продолжения раскрутки стека мы должны дизассемблировать код по этому адресу (листинг 3.10).

Листинг 3.10. Дизассемблерный листинг праматеринской функции ("бабушки")

```
004012FA    call    00401350
004012FF    cmp     eax,0FFh
00401302    je      0040132D
00401304    push    eax
00401305    lea     eax,[esp+8]
00401309    push    405054h
0040130E    push    eax
0040130F    call    dword ptr ds:[4033B4h]
00401315    add     esp,0Ch
00401318    lea     ecx,[esp+4]
0040131C    push    0
0040131E    push    0
00401320    push    ecx
00401321    mov     ecx,esi
00401323    call    00401BC4
00401328    pop     esi
00401329    add     esp,64h
0040132C    ret                     ; SS:[ESP] = 6C2923D8
```

Прокручивая экран вниз, мы замечаем инструкцию `ADD ESP, 64`, закрывающую текущий кадр стека. Еще восемь байт снимает инструкция `40136Fh:RET 8` и четыре байта оттягивает на себя `401328:POP ESI`. Таким образом, позиция адреса возврата в стеке равна: `current_ESP + 64h + 8 + 4 == 70h`. Спускаемся на 70h байт ниже и видим адрес возврата из праматеринской функции (листинг 3.11).

Листинг 3.11. Адрес возврата из праматеринской функции

```
0012F500  77E61D49  6C2923D8  00403458 ← 00401328:POP ESI/ret;
```

Первое двойное слово — это значение регистра ESI, который нам предстоит вручную восстановить; второе — адрес возврата из функции. Нажатием <Ctrl>+<G>, "0x6C2923D8" мы продолжаем "раскручивать" стек (листинг 3.12).

Листинг 3.12. Дизассемблерный листинг прапраматеринской функции

```
6C2923D8     jmp     6C29237B
…
6C29237B     mov     eax,ebx
6C29237D     pop     esi
6C29237E     pop     ebx
6C29237F     pop     ebp
6C292380     ret     1Ch
```

Вот мы и добрались до восстановления регистров! Сместившись на одно двойное слово вправо (оно только что было вытолкнуто из стека командой `RET`), переходим в окно **Registers** и восстанавливаем регистры ESI, EBX, EBP, извлекая сохраненные значения из стека (листинг 3.13).

Листинг 3.13. Содержимое регистров, ранее сохраненных в стеке вместе с адресом возврата

```
0012F500  77E61D49  6C2923D8  00403458 ← 6C29237D:pop esi
0012F50C  00000111  0012F540  6C2922AE ←6C29237E:pop ebx/pop ebp/ret 1Ch
```

Как вариант можно переместить регистр EIP на адрес `6C29237Dh`, а регистр ESP на адрес `12F508h`, после чего нажать на клавишу <F5> для продолжения выполнения программы. И этот прием действительно срабатывает! Причем реанимированная программа уже не ругается на ошибку последней операции (как это было при восстановлении путем принудительного выхода из функции), а просто ее не выполняет. Красота!

Передача управления на функцию обработки сообщений

Двум предыдущим способам "реанимации" приложений присущи серьезные ограничения и недостатки. При тяжелых разрушениях стека, вызванных атаками типа buffer overfull или же просто алгоритмическими ошибками, содержимое важнейших регистров процессора окажется искажено, и мы уже

не сможем ни совершить откат (стек утерян), ни выйти из текущей функции (EIP "смотрит" в "космос"). В консольных приложениях в такой ситуации действительно очень мало что можно сделать. Вот GUI — другое дело! Концепция событийно ориентированной архитектуры наделяет всякое оконное приложение определенными серверными функциями. Даже если текущий контекст выполнения необратимо утерян, мы можем передать управление на цикл извлечения и диспетчеризации сообщений, заставляя программу продолжить обработку действий пользователя.

Классический цикл обработки сообщений выглядит так, как это показано в листинге 3.14.

Листинг 3.14. Классический цикл обработки сообщений

```
while (GetMessage(&msg, NULL, 0, 0))
{
     TranslateMessage(&msg);
     DispatchMessage(&msg);
}
```

Все, что нам нужно — это передать управление на цикл `while`, даже не заботясь о настойке кадра стека, поскольку оптимизированные программы (а таковых большинство) адресуют свои локальные переменные не через EBP, а непосредственно через сам ESP. Конечно, при обращении к переменной `msg`, функция "угробит" содержимое стека, лежащее ниже его вершины, но это уже не важно.

Правда, при выходе из приложения оно "упадет" окончательно (ведь вместо адреса возврата из функции обработки сообщений машинная команда `RET` обнаружит на вершине стека неизвестно что), но это произойдет *после* сохранения всех данных, и потому никакой угрозы не несет. Исключение составляют приложения, "забывающие" закрыть все открытые файлы и перекладывающие эту работу на плечи функции `ExitProcess`. Что ж! Можно так подправить адрес возврата, чтобы он указывал на функцию ExitProcess!

Давайте создадим простейшее Windows-приложение и поэкспериментируем с ним. Запустив Microsoft Visual Studio, выберем **New → Project → Win32 Application** и там — **Typical Hello**, **World application**. Добавим новый пункт меню, а в нем: `char *p; *p = 0`; и откомпилируем этот проект с отладочной информацией.

"Роняем приложение на пол" и, запустив отладчик, подгоняем мышь к первой строке цикла обработки сообщений и в появившемся контекстном меню находим пункт **Set Next Statement**. Нажимаем клавишу <F5> для возобновления работы программы, и она действительно возобновляет свою работу!

А теперь откомпилируем наш проект в чистовом варианте (т. е. без отладочной информации) и попробуем реанимировать приложение в голом машинном коде. Пользуясь тем обстоятельством, что Windows — это действительно многозадачная среда, в которой крушение одного процесса не мешает работе всех остальных, запустим свой любимый дизассемблер (например, IDA Pro) и проанализируем таблицу импорта отлаживаемой программы (вообще-то это может сделать и бесплатно распространяемый утилитой dumpbin, но его отчет не так нагляден).

Целью нашего поиска будут функции `TranslateMessage`/`DispatchMessage` и перекрестные ссылки, ведущие к циклу выборки сообщений (листинг 3.15).

Листинг 3.15. Поиск функций TranslateMessage/DispatchMessage в таблице импорта

```
.idata:004040E0 ; BOOL __stdcall TranslateMessage(const MSG *lpMsg)
.idata:004040E0     extrn TranslateMessage:dword     ; DATA XREF: _WinMain@16+71↑r
.idata:004040E0                                      ; _WinMain@16+8D↑r
.idata:004040E4 ; LONG __stdcall DispatchMessageA(const MSG *lpMsg)
.idata:004040E4     extrn DispatchMessageA:dword     ; DATA XREF: _WinMain@16+94↑r
.idata:004040E8
```

С функцией `DispatchMessage` связана всего лишь одна перекрестная ссылка, со всей очевидностью ведущая к искомому циклу обработки сообщений, дизассемблерный код которого выглядит так, как показано в листинге 3.16.

Листинг 3.16. Дизассемблерный код функции обработки сообщений

```
.text:00401050          mov     edi, ds:GetMessageA
.text:00401050 ; первый вызов GetMessageA (это еще не цикл, это только его преддверье)
.text:00401050
.text:00401056          push    0               ; wMsgFilterMax
.text:00401058          push    0               ; wMsgFilterMin
.text:0040105A          lea     ecx, [esp+2Ch+Msg]
.text:0040105A ; ECX указывает на область памяти, через которую GetMessageA
.text:0040105A ; станет возвращать сообщение. текущее значение ESP может быть
.text:0040105A ; любым, главное, чтобы оно указывало на действительную область
.text:0040105A ; памяти (см. карту памяти, если значение ESP оказалось искажено
.text:0040105A ; настолько, что вывело его в "космос")
.text:0040105A ;
.text:0040105E          push    0               ; hWnd
.text:00401060          push    ecx             ; lpMsg
.text:00401061          mov     esi, eax
```

```
.text:00401063          call    edi             ; GetMessageA
.text:00401063 ; вызываем GetMessageA
.text:00401063
.text:00401065          test    eax, eax
.text:00401067          jz      short loc_4010AD
.text:00401067 ; проверка на наличие необработанных сообщений в очереди
.text:00401067
…
.text:00401077 loc_401077:                      ; CODE XREF: _WinMain@16+A9↓j
.text:00401077 ; начало цикла обработки сообщений
.text:00401077
.text:00401077          mov     eax, [esp+2Ch+Msg.hwnd]
.text:0040107B          lea     edx, [esp+2Ch+Msg]
.text:0040107B ; EDX указывает на область памяти, используемую для передачи сообщений
.text:0040107B
.text:0040107F          push    edx             ; lpMsg
.text:00401080          push    esi             ; hAccTable
.text:00401081          push    eax             ; hWnd
.text:00401082          call    ebx             ; TranslateAcceleratorA
.text:00401082 ; вызываем функцию TranslateAcceleratorA
.text:00401082
.text:00401084          test    eax, eax
.text:00401086          jnz     short loc_40109A
.text:00401086 ; проверка на наличие в очереди необработанных сообщений
.text:00401086
.text:00401088          lea     ecx, [esp+2Ch+Msg]
.text:0040108C          push    ecx             ; lpMsg
.text:0040108D          call    ebp             ; TranslateMessage
.text:0040108D ; вызываем функцию TranslateMessage, если есть что транслировать
.text:0040108D
.text:0040108F          lea     edx, [esp+2Ch+Msg]
.text:00401093          push    edx             ; lpMsg
.text:00401094          call    ds:DispatchMessageA
.text:00401094 ; диспетчеризуем сообщение
.text:0040109A
.text:0040109A loc_40109A:                      ; CODE XREF: _WinMain@16+86↑j
.text:0040109A          push    0               ; wMsgFilterMax
.text:0040109C          push    0               ; wMsgFilterMin
.text:0040109E          lea     eax, [esp+34h+Msg]
.text:004010A2          push    0               ; hWnd
.text:004010A4          push    eax             ; lpMsg
```

```
.text:004010A5          call    edi             ; GetMessageA
.text:004010A5 ; читаем очередное сообщений из очереди
.text:004010A5
.text:004010A7          test    eax, eax
.text:004010A9          jnz     short loc_401077
.text:004010A9 ; вращаем цикл обработки сообщений
.text:004010A9
.text:004010AB          pop     ebp
.text:004010AC          pop     ebx
.text:004010AD
.text:004010AD loc_4010AD:                      ; CODE XREF: _WinMain@16+67↑j
.text:004010AD          mov     eax, [esp+24h+Msg.wParam]
.text:004010B1          pop     edi
.text:004010B2          pop     esi
.text:004010B3          add     esp, 1Ch
.text:004010B6          retn    10h
.text:004010B6 _WinMain@16      endp
```

Мы видим, что цикл обработки сообщений начинается с адреса `401050h` и именно на этот адрес следует передать управление, чтобы возобновить работу "упавшей" программы. Пробуем сделать это, и программа работает!

Разумеется, настоящее приложение оживить намного сложнее, поскольку цикл обработки сообщений в нем рассредоточен по большому количеству функций, отождествить которые при беглом дизассемблировании невозможно. Тем не менее, приложения, построенные на основе общедоступных библиотек, например, MFC, OWL, обладают вполне предсказуемой архитектурой, и реанимировать их вполне возможно.

Рассмотрим, как устроен цикл обработки сообщений в MFC. Большую часть своего времени исполнения MFC-приложения проводят внутри функции `CWinThread::Run(void)`, которая периодически опрашивает очередь на предмет поступления свежих сообщений и рассылает их соответствующим обработчикам. Если один из обработчиков "споткнулся" и довел систему до критической ошибки, выполнение программы может быть продолжено в функции `Run`. В этом-то и заключается ее главная прелесть!

Функция не имеет явных аргументов, но принимает скрытый аргумент `this`, указывающей на экземпляр класса CWinThread или производный от него класс, без которого функция просто не сможет работать. К счастью, таблицы виртуальных методов класса CWinThread содержат достаточное количество "родимых пятен", чтобы указатель this можно было воссоздать вручную.

Загрузим функцию `Run` в дизассемблер и отметим все обращения к таблице виртуальных методов, адресуемой через регистр ECX (листинг 3.17).

Листинг 3.17. Дизассемблерный листинг функции Run (фрагмент)

```
.text:6C29919D n2k_Trasnlate_main:                  ; CODE XREF: MFC42_5715+1F↑j
.text:6C29919D                                      ; MFC42_5715+67↓j ...
.text:6C29919D        mov     eax, [esi]
.text:6C29919F        mov     ecx, esi
.text:6C2991A1        call    dword ptr [eax+64h]   ; CWinThread::PumpMessage(void)
.text:6C2991A4        test    eax, eax
.text:6C2991A6        jz      short loc_6C2991DA
.text:6C2991A8        mov     eax, [esi]
.text:6C2991AA        lea     ebp, [esi+34h]
.text:6C2991AD        push    ebp
.text:6C2991AE        mov     ecx, esi
.text:6C2991B0        call    dword ptr [eax+6Ch] ; CWinThread::IsIdleMessage(MSG*)
.text:6C2991B3        test    eax, eax
.text:6C2991B5        jz      short loc_6C2991BE
.text:6C2991B7        push    1
.text:6C2991B9        mov     [esp+14h], ebx
.text:6C2991BD        pop     edi
.text:6C2991BE
.text:6C2991BE loc_6C2991BE:                        ; CODE XREF: MFC42_5715+51↑j
.text:6C2991BE       push     ebx                   ; wRemoveMsg
.text:6C2991BF       push     ebx                   ; wMsgFilterMax
.text:6C2991C0       push     ebx                   ; wMsgFilterMin
.text:6C2991C1       push     ebx                   ; hWnd
.text:6C2991C2       push     ebp                   ; lpMsg
.text:6C2991C3       call     ds:PeekMessageA
.text:6C2991C9       test     eax, eax
.text:6C2991CB       jnz      short n2k_Trasnlate_main
.text:6C2991CD
```

Таким образом, функция `Run` ожидает получить указатель на двойное слово, указывающее на таблицу виртуальных методов, 0x19 и 0x1B элементы которой представляют собой функции `PumpMessage` и `IsIdleMessage` соответственно (или переходники к ним). Адреса импортируемых функций, если только динамическая библиотека не была перемещена, можно узнать в том же дизассемблере; в противном случае, следует отталкиваться от базового адреса модуля, отображаемого отладчиком по команде `Modules`. При условии, что эти две функции не были перекрыты программистом, поиск нужной нам виртуальной таблицы не составит никакого труда.

По непонятным причинам библиотека MFC42.DLL не экспортирует символьных имен функций, и эту информацию нам приходится добывать

самостоятельно. Обработав библиотеку MFC42.LIB утилитой dumpbin, запущенной с ключом "/ARCH", мы определим ординалы обеих функций (ординал PumpMessage — 5307, а IsIdleMessage — 4079). Остается найти эти значения в экспорте библиотеки MFC42.DLL (`dumpbin /EXPORTS mfc42.dll > mfc42.txt`), из чего мы узнаем, что адрес функции PumpMessage, `6C291194h`, а IsIdleMessage — `6C292583h`.

Теперь мы должны найти указатели на функции PumpMessage/IsIdleMessage в памяти, а точнее — в секции данных, базовый адрес которой содержится в заголовке PE-файла, только помните, что в процессорах x86 наименее значимый байт располагается по меньшему адресу, т. е. все числа записываются в обратном порядке. К сожалению, отладчик Microsoft Visual Studio Debugger не поддерживает операцию поиска в памяти, и нам приходится действовать обходным путем — копировать содержимое дампа в буфер обмена, вставлять его в текстовой файл и, нажав клавишу <F7>, искать адреса уже там.

Долго ли, коротко ли, но интересующие нас указатели обнаруживаются по адресам `403044h/40304Ch` (естественно, у вас эти адреса могут быть и другими). Причем обратите внимание: расстояние между указателями в точности равно расстоянию между указателями на `[EAX + 64h]` и `[EAX + 6Ch]`, а очередность их размещения в памяти обратна порядку объявления виртуальных методов. Это — хороший признак, и мы, скорее всего, находимся на правильном пути (листинг 3.18).

Листинг 3.18. Адреса функций IsIdleMessage/PumpMessage, найденные в секции данных

```
00403044     6C2911D4 6C292583 6C291194 ; IsIdleMessage/PumpMessage
00403050     6C2913D0 6C299144 6C297129
0040305C     6C297129 6C297129 6C291A47
```

Указатели, указывающие на адреса `403048h/40304Ch`, очевидно, и будут кандидатами в члены искомой таблицы виртуальных методов класса CWinThread. Расширив сферу поиска всем адресным пространством отлаживаемого процесса, мы обнаруживаем два следующих переходника (листинг 3.19).

Листинг 3.19. Переходники к функциям IsIdleMessage/PumpMessage, найденные там же

```
00401A20     jmp     dword ptr ds:[403044h] ; IsIdleMessage
00401A26     jmp     dword ptr ds:[403048h] ;
00401A2C     jmp     dword ptr ds:[40304Ch] ; PumpMessage
```

Ага, уже теплее! Мы нашли не сами виртуальные функции, но переходники к ним. Раскручивая этот запутанный клубок, попробуем отыскать ссылки на `401A26h`/`401A2Ch`, которые передают управление на приведенный ранее код (листинг 3.20).

Листинг 3.20. Виртуальная таблица класса CWinThread

```
00403490    00401A9E 00401040 004015F0 ← 0x0,  0x1,  0x2  элементы
0040349C    00401390 004015F0 00401A98 ← 0x3,  0x4,  0x5  элементы
004034A8    00401A92 00401A8C 00401A86 ← 0x6,  0x7,  0x8  элементы
004034B4    00401A80 00401A7A 00401A74 ← 0x9,  0xA,  0xB  элементы
004034C0    00401010 00401A6E 00401A68 ← 0xC,  0xD,  0xE  элементы
004034CC    00401A62 00401A5C 00401A56 ← 0xF,  0x10, 0x11 элементы
004034D8    00401A50 00401A4A 00401A44 ← 0x12, 0x13, 0x14 элементы
004034E4    00401A3E 004010B0 00401A38 ← 0x15, 0x16, 0x17 элементы
004034F0    00401A32 00401A2C 00401A26 ← 0x18, 0x19, 0x1A элементы (PumpMessage)
004034FC    00401A20 00401A1A 00401A14 ← 0x1B, 0x1C, 0x1D элементы (IsIdleMessage)
```

Даже неопытный исследователь программ распознает в этой структуре данных таблицу виртуальных функций. Указатели на переходники к функциям `PumpMessage`/`IsIdleMessage` разделяются ровно одним элементом, как того и требуют условия задачи. Предположим, что эта виртуальная таблица, которая нам и нужна. Для проверки этого предположения отсчитаем 0x19 (25) элементов верх от `4034F4h` и попытаемся найти указатель, ссылающийся на ее начало. Если повезет и он окажется экземпляром класса CwinThread, тогда программа сможет корректно продолжить свою работу (листинг 3.21).

Листинг 3.21. Экземпляр класса CWinThread, вручную найденный нами в памяти

```
004050B8    00403490 00000001 00000000
004050C4    00000000 00000000 00000001
```

Действительно, в памяти обнаруживается нечто похожее. Записываем в регистр ECX значение `4050B8h`, находим в памяти функцию `Run` (как уже говорилось, если только она не была перекрыта, ее адрес — `6C299164h` — известен). Нажимаем комбинацию клавиш <Ctrl>+<G>, затем вводим "0x6C299164" и в контекстном меню, вызванном правой клавишей мыши, выбираем **Set Next Statement**. Программа, отделавшись легким "испугом", продолжает свое исполнение, ну а мы на радостях идем пить пиво (кофе, квас, чай — по вкусу).

Аналогичным путем можно вернуть к жизни и зависшие приложения, потерявшие нить управления и не реагирующие ни на мышь, ни на клавиатуру.

Как подключить дамп памяти

> ...в отделе программ весь пол был усеян дырочками от перфокарт и какие-то мужики ползали по раскатанной по полу 20-метровой распечатке аварийного дампа памяти с целью обнаружения ошибки в распределителе памяти ОС-360. К президенту подошел начальник отдела и сообщил, что есть надежда сделать это еще к обеду.
>
> *Ю. Антонов. "Юность Гейтса"*

Дамп памяти (memory dump, также называемый "корой" [от английского core — сердцевина], crash- или аварийным дампом), сброшенный системой при возникновении критической ошибки — не самое убедительное средство для выявления причин катастрофы, но ничего другого в руках администратора зачастую просто не бывает. Последний вздох операционной системы, похожий на дурно пахнущую навозную кучу, из которой высовывается чей-то наполовину разложившийся труп, мгновенным снимком запечатленный в момент неустранимого сбоя, — вот что такое дамп памяти во время крушения системы! Копание в нем вряд ли доставит вам удовольствие. Не исключено, что истинного виновника краха системы вообще не удастся найти. Допустим, некий некорректно работающий драйвер вторгся в область памяти, принадлежащую другому драйверу, и наглым образом затер критические структуры данных, сделав из чисел "винегрет". К тому моменту, когда драйвер-жертва пойдет вразнос, драйвер-хищник может быть вообще выгружен из системы, и определить его причастность к крушению системы по одному лишь дампу практически нереально.

Тем не менее, полностью игнорировать факт существования дампа, право же, не стоит. В конце концов, до возникновения инактивных отладчиков ошибки в программах приходилось искать именно так. Избалованность современных программистов визуальными средствами анализа, увы, не добавляет им уверенности в тех ситуациях, когда неумолимая энтропия оставляет их со своими проблемами один на один. Но довольно лирики. Переходим к делу, расписывая каждое действие по шагам.

Первым делом, необходимо войти в конфигурацию системы **Пуск → Настройка → Панель управления → Система** (Start → Settings → Control Panel → System) и убедиться, что настройки дампа соответствуют предъявляемым к ним требованиям **Дополнительно → Загрузка и восстановление → Отказ системы** (Startup/Shutdown → Recovery) в Windows 2000 RUS и Windows NT 4.0 ENG соответственно). Операционная система Windows 2000 поддерживает три раз-

новидности дампов памяти: *малый дамп памяти* (small memory dump), *дамп памяти ядра* (kernel memory dump) и *полный дамп памяти* (complete dump memory). Для изменения настроек дампа вы должны иметь права администратора.

Малый дамп памяти занимает всего лишь 64 Кбайта (а отнюдь не 2 Мбайта, как утверждает контекстная помощь) и включает в себя:

- копию "голубого экрана смерти";
- перечень загруженных драйверов;
- контекст обрушившегося процесса со всеми его потоками;
- первые 16 Кбайт содержимого ядерного стека "обрушившегося" потока.

Разочаровывающие малоинформативные сведения! Непосредственный анализ дампа дает нам лишь адрес возникновения ошибки и имя драйвера, к которому этот адрес принадлежит. При условии, что конфигурация системы не была изменена после возникновения сбоя, мы можем загрузить отладчик и дизассемблировать подозреваемый драйвер, но это мало что даст. Ведь содержимое сегмента данных на момент возникновения сбоя нам неизвестно, более того — мы не можем утверждать, что видим те же самые машинные команды, что вызвали сбой. Поэтому малый дамп памяти полезен лишь тем администраторам, которым достаточно одного лишь имени нестабильного драйвера. Как показывает практика, в подавляющем большинстве случаев этой информации оказывается вполне достаточно. Разработчикам драйвера отсылается гневный "бан-рапорт" (вместе с дампом!), а сам драйвер тем временем заменяется другим — более новым и надежным. По умолчанию малый дамп памяти записывается в каталог %SystemRoot%\Minidump, где ему присваивается имя Mini, дата записи дампа и порядковый номер сбоя на данный день. Например, *Mini110701-69.dmp* — 69 дамп системы от 07 ноября 2001 года (не пугайтесь! это просто я отлаживал драйвера).

Дамп памяти ядра содержит намного более полную информацию о сбое и включает в себя всю память, выделенную ядром и его компонентами (драйверами, уровнем абстракции от оборудования и т. д.), а также копию "голубого экрана смерти". Размер дампа памяти ядра зависит от количества установленных драйверов и варьируется от системы к системе. Контекстная помощь утверждает, что эта величина составляет от 50 до 800 Мбайт. Ну, на счет 800 Мбайт авторы явно "загнули", и объем в 50—100 Мбайт выглядит более вероятным (техническая документация на систему сообщает, что ориентировочный размер дампа ядра составляет треть объема физической оперативной памяти, установленной на системе). Это наилучший компромисс между накладными расходами на дисковое пространство, скорости сброса дампа и информативностью последнего. Весь "джентльменский" минимум информации — в вашем распоряжении. Практически все типовые ошибки драйверов и прочих ядерных компонентов могут быть локализованы с точ-

ностью до байта, включая и те, что вызваны физическим сбоем аппаратуры (правда, для этого вы должны иметь некоторый "патологоанатомический" опыт исследования "трупных" дампов системы). По умолчанию дамп памяти ядра записывается в файл *%SystemRoot%\Memory.dmp*, затирая или не затирая (в зависимости от текущих настроек системы) предыдущий дамп.

Полный дамп памяти включает в себя все содержимое физической памяти компьютера, занятое как прикладными, так и компонентами ядра системы. Полный дамп памяти оказывается особенно полезным при отладке ASPI/SPTI-приложений, которые в силу своей специфики могут "уронить" ядро даже с прикладного уровня. Несмотря на довольно большой размер, равный размеру оперативной памяти, полный дамп остается наиболее любимым дампом всех системных программистов (системные же администраторы в своей массе предпочитают малый дамп). Это не покажется удивительным, если вспомнить, что объемы жестких дисков давно перевалили за отметку 100 Гбайт, а оплата труда системных программистов за последние несколько лет даже несколько возросла. Лучше иметь невостребованный полный дамп под рукой, чем кусать локти при его отсутствии. По умолчанию полный дамп памяти записывается в файл *%SystemRoot%\Memory.dmp*, затирая или не затирая (в зависимости от текущих настроек системы) предыдущий дамп.

Выбрав предпочтительный тип дампа, давайте совершим учебное "падение" системы, отрабатывая методику его анализа в "полевых" условиях. Для этого нам понадобится:

- *комплект разработчика драйверов* (Driver Development Kit или сокращенно DDK), бесплатно распространяемый фирмой Microsoft и содержащий в себе подробную техническую документацию по ядру системы; несколько компиляторов Си/Си++ и ассемблера, а также достаточно "продвинутые" средства анализа дампа памяти;
- драйвер W2K_KILL.SYS или любой другой *драйвер-убийца* операционной системы, например BDOS.EXE от Марка Русиновича, позволяющий получить дамп в любое удобное для нас время, не дожидаясь возникновения критической ошибки (бесплатную копию программы можно скачать с адреса **http://www.sysinternals.com**);
- *файлы символьных идентификаторов (symbol files)*, необходимые отладчикам ядра для его нормального функционирования и делающие дизассемблерный код более наглядным. Файлы символьных идентификаторов входят в состав "зеленого" набора MSDN, но, в принципе, без них можно и обойтись, однако переменная окружения _NT_SYMBOL_PATH по любому должна быть определена, иначе отладчик i386kd.exe работать не будет;
- одна или несколько книжек, описывающих архитектуру ядра системы. Очень хороша в этом смысле книга *"Внутреннее устройство Windows 2000"* Марка Руссиновича и Дэвида Соломона, интересная как системным программистам, так и администраторам.

Итак, установив DDK на свой компьютер и завершив все приложения, запускаем драйвер-убийцу, и под "скрипящий" звук записывающегося дампа система немедленно выбрасывает "голубой экран смерти" (BSOD — *Blue Screen Of Death*), свидетельствующий о возникновении неустранимого сбоя системы с краткой информацией о нем (рис. 3.4).

```
*** STOP: 0x0000001E (0xC0000005, 0xBE80B000, 0x00000000, 0x00000000)
KMODE_EXEPTION_NOT_HALTED

***  Address  0xBE80B000  base  at  0xBE80A000,  Date  Stamp  389db915  -
w2k_kill.sys

Beginning dump of physical memory
Dumping physical memory to disk: 69
```

Рис. 3.4. "Голубой экран смерти"

Для большинства администраторов "голубой экран смерти" означает лишь одно — системе "поплохело" настолько, что она предпочла смерть позору неустойчивого функционирования. Что же до таинственных надписей, сопровождающих эти экраны, то они остаются сплошной загадкой. Но только не для настоящих профессионалов!

Мы начнем с левого верхнего угла экрана и, зигзагами спускаясь вниз, трассируем все надписи по порядку.

- `*** STOP:` — буквально означает "останов [системы]" и не несет в себе никакой дополнительной информации;
- `0x0000001E` — представляет собой код Bug Check, содержащий категорию сбоя. Расшифровку кодов Bug Check можно найти в DDK. В данном случае это `0x1E` — **KMODE_EXEPTION_NOT_HALTED**, о чем и свидетельствует символьное имя, расположенное строкой ниже. Краткое объяснение некоторых, наиболее популярных кодов Bug Check приведено в табл. 3.1. Определение рейтинга популярности кодов Bug Check осуществлялось путем подсчета упоминаний о них в конференциях Интернет (спасибо старику Googl'у). Полноту фирменной документации она, разумеется, не заменяет, но некоторое представление о целесообразности скачивания 70 Мбайт DDK все-таки дает;
- "арабская вязь" в круглых скобках — это четыре Bug Check-параметра, физический смысл которых зависит от конкретного кода Bug Check и вне его контекста теряет всякий смысл; применительно к KMODE_EXEPTION_NOT_HALTED — первый Bug Check-параметр содержит номер возбужденного исключения. Судя по таблице 3.1, это — STATUS_ACCESS_VIOLATION — доступ к запрещенному адресу памяти — и четвертый

Bug Check-параметр указывает какой именно. В данном случае он равен нулю, следовательно, некоторая машинная инструкция попыталась совершить обращение по null-pointer, соответствующему инициализированному указателю, ссылающемуся на невыделенный регион памяти. Ее адрес содержится во втором Bug Check-параметре. Третий Bug Check-параметр в данном конкретном случае не определен;

- `*** Address 0xBE80B00` — это и есть тот адрес, по которому произошел сбой. В данном случае он идентичен второму Bug Check-параметру, однако так бывает далеко не всегда (коды Bug Check собственно и не подряжались хранить чьи-либо адреса);
- `base at 0xBE80A00` — содержит базовый адрес загрузки модуля-нарушителя системного порядка, по которому легко установить "паспортные" данные самого этого модуля.

Внимание

Далеко не во всех случаях правильное определение базового адреса вообще возможно.

Воспользовавшись любым подходящим отладчиком (например, Soft-Ice от Нумега или i386kd от Microsoft), введем команду, распечатывающую перечень загруженных драйверов с их краткими характеристиками (в i386kd это осуществляется командой `!drivers`). Как одним из вариантов, можно воспользоваться утилитой drivers.exe, входящей в NTDDK. Но какой бы вы путь не избрали, результат будет приблизительно следующим:

```
kd>                                                                !drivers!drivers
Loaded                    System                    Driver                   Summary
Base         Code Size        Data Size       Driver Name              Creation Time
80400000  142dc0 (1291 kb)  4d680 (309 kb) ntoskrnl.exe   Wed Dec 08 02:41:11 1999
80062000    cc20 (  51 kb)   32c0 ( 12 kb)        hal.dll   Wed Nov 03 04:14:22 1999
f4010000    1760 (   5 kb)   1000 (  4 kb)   BOOTVID.DLL   Thu Nov 04 04:24:33 1999
bffd8000   21ee0 ( 135 kb)   59a0 ( 22 kb)      ACPI.sys   Thu Nov 11 04:06:04 1999
be193000   16f60 (  91 kb)   ccc0 ( 51 kb)    kmixer.sys   Wed Nov 10 09:52:30 1999
bddb4000   355e0 ( 213 kb) 10ac0 ( 66 kb)     ATMFD.DLL   Fri Nov 12 06:48:40 1999
be80a000     200 (    0 kb)    a00 (  2 kb) w2k_kill.sys   Mon Aug 28 02:40:12 2000
TOTAL:   835ca0 (8407 kb) 326180 (3224 kb) (    0 kb     0 kb)
```

Обратите внимание на выделенную полужирным черным цветом строку с именем `w2k_kill.sys`, найденную по ее базовому адресу `0xBE80A00`. Это и есть тот самый драйвер, который нам нужен! А впрочем, этого можно и не делать, поскольку имя "неправильного" драйвера и без того присутствует на "голубом экране смерти";

- две нижние строки отражают прогресс "сброса" дампа на диск, развлекая администратора чередой быстро меняющихся циферок на это время.

Таблица 3.1. *Физический смысл наиболее популярных кодов Bug Check с краткими пояснениями*

Категория		Описание
hex-код	**Символьное имя**	
0x0A	IRQL_NOT_LESS_OR_EQUAL	Драйвер попытался обратиться к странице памяти на уровне DISPATCH_LEVEL или более высоком, что и привело к краху, поскольку менеджер виртуальной памяти работает на более низком уровне; источником сбоя может быть и BIOS, и драйвер, и системный сервис (особенно этим грешат вирусные сканеры и FM-тюнеры); как вариант — проверьте кабельные терминаторы на SCSI-накопителях и Master/Slayer на IDE, отключите кэширование памяти в BIOS; если и это не поможет, обратитесь к четырем параметрам кода Bug Check, содержащим ссылку на память, к которой осуществлялся доступ, уровень IRQ *(Interrupt ReQuest)*, тип доступа (чтение/запись) и адрес машинной инструкции драйвера
0x1E	KMODE_EXCEPTION_NOT_HANDLED	Компонент ядра возбудил исключение и "забыл" его обработать; номер исключения содержится в первом Bug Check-параметре; обычно он принимает одно из следующих значений: • 0x80000003 (STATUS_BREAKPOINT): встретилась программная точка останова — отладочный рудимент, по небрежности не удаленный разработчиком драйвера; • (0xC0000005) STATUS_ACCESS_VIOLATION: доступ к запрещенному адресу (четвертый Bug Check-параметр уточняет к какому) — ошибка разработчика;

Таблица 3.1 *(продолжение)*

Категория		Описание
hex-код	**Символьное имя**	
0x1E	KMODE_EXCEPTION_NOT_HANDLED	• (0xC000021A) STATUS_SYSTEM_PROCESS_TERMINATED: сбой процессов CSRSS и/или Winlogon, источником которого могут быть как компоненты ядра, так и пользовательские приложения; обычно это происходит при заражении машины вирусом или нарушении целостности системных файлов; • (0xC0000221) STATUS_IMAGE_CHECSUM_MISMATCH: целостность одного из системных файлов оказалась нарушена; второй Bug Check-параметр содержит адрес машинной команды, возбудившей исключение
0x24	NTFS_FILE_SYSTEM	Проблема с драйвером NTFS.SYS, обычно возникающая вследствие физического разрушения диска, реже — при остром недостатке физической оперативной памяти
0x2E	DATA_BUS_ERROR	Драйвер обратился по несуществующему физическому адресу; если только это не ошибка драйвера; оперативная память и/или кэш-память процессора (видеопамять) неисправны или же работают на запредельных тактовых частотах
0x35	NO_MORE_IRP_STACK_LOCATIONS	Драйвер более высокого уровня обратился к драйверу более низкого уровня посредством IoCallDriver-интерфейса, однако свободного пространства в стеке IRP *(I/O Request Packet)* не оказалось и передать весь IRP-пакет целиком не удалось; это гибельная ситуация, не имеющая прямых решений; попытайтесь удалить один или несколько наименее нужных драйверов, быть может тогда система заработает

Таблица 3.1 *(продолжение)*

Категория		Описание
hex-код	**Символьное имя**	
0x3F	NO_MORE_SYSTEM_PTES	Результат сильной фрагментации таблицы страниц PTE *(Page Table Entry)*, приводящей к невозможности выделения затребованного драйвером блока памяти; обычно это характерно для аудио- или видеодрайверов, манипулирующих огромными блоками памяти и к тому же не всегда их вовремя освобождающих; для решения проблемы попробуйте увеличить количество PTE (до 50000 максимум) в следующей ветке реестра: HLLM\SYSTEM\CurrentControlSet\Control\SessionManager\Memory Management\SystemPages
0x50	PAGE_FAULT_IN_NONPAGED_AREA	Обращение к несуществующей странице памяти, вызванное либо неисправностью оборудования (как правило — оперативной, видео или кэш-памяти), либо некорректно спроектированным сервисом (этим грешат многие антивирусы, в том числе Касперский и Доктор Веб), либо разрушениями NTFS-тома (запустите программу chkdsk.exe с ключами /f и /r), также попробуйте запретить кэширование памяти в BIOS
0x58	FTDISK_INTERNAL_ERROR	Сбой RAID-массива, — при попытке загрузки с основного диска система обнаружила, что он поврежден, тогда она обратилась к его зеркалу, но таблицы разделов не оказалось и там
0x76	PROCESS_HAS_LOCKED_PAGES	Драйвер не смог освободить "залоченные" страницы после завершения операции ввода-вывода; для определения имени дефективного драйвера следует обратиться к ветке HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management и установить параметр TrackLockedPages типа DWORD в значение 1, потом перезагрузить систему, после чего та будет сохранять трассируемый стек, и, если нехороший драйвер вновь начнет чудить, возникнет BSOD с кодом Bug Check: 0xCB, позволяющим определить виновника

Таблица 3.1 *(продолжение)*

Категория		Описание
hex-код	**Символьное имя**	
0x77	KERNEL_STACK_INPAGE_ERROR	Страница данных памяти ядра по техническим причинам недоступна, если первый код Bug Check не равен нулю, то он может принимать одно из следующих значений: (0xC000009A) STATUS_INSUFFICIENT_RESOURCES — недостаточно системных ресурсов; (0xC000009C) STATUS_DEVICE_DATA_ERROR — ошибка чтения с диска (bad-сектор?); (0xC000009D) STATUS_DEVICE_NOT_CONNECTED — система не видит привод (неисправность контроллера, плохой контакт шлейфа); (0xC000016A) STATUS_DISK_OPERATION_FAILED — ошибка чтения диска (bad-сектор или неисправный контроллер); (0xC0000185) STATUS_IO_DEVICE_ERROR — неправильное "термирование" SCSI-привода или конфликт IRQ IDE-приводов; нулевое же значение первого кода Bug Check указывает на неизвестную аппаратную проблему; такое сообщение может появляться и при заражении системы вирусами, и при разрушении диска старыми "докторами", и при отказе RAM — войдите в консоль восстановления и запустите программу chkdsk.exe с ключом /r
0x7A	KERNEL_DATA_INPAGE_ERROR###	Страница данных памяти ядра по техническим причинам недоступна, второй Bug Check-параметр содержит статус обмена, четвертый — виртуальный страничный адрес, загрузить который не удалось; возможные причины сбоя — те же дефектные сектора, попавшие в файл-подкачки pagefile.sys, сбои дискового контроллера, ну и вирусы, наконец

Таблица 3.1 *(продолжение)*

Категория		Описание
hex-код	**Символьное имя**	
0x7B	INACCESSIBLE_BOOT_DEVICE	Загрузочное устройство недоступно, — таблица разделов повреждена или не соответствует файлу boot.ini; также такое сообщение появляется при замене материнской платы с интегрированным IDE-контроллером (или замене SCSI-контроллера), поскольку всякий контроллер требует "своих" драйверов и при подключении жесткого диска с установленной Windows NT на компьютер, оснащенный несовместимым оборудованием, операционная система просто откажется грузиться и ее необходимо будет переустановить (опытные администраторы могут переустановить непосредственно сами дисковые драйвера, загрузившись с консоли восстановления); также не помешает проверить общую исправность оборудования и наличие вирусов на диске
0x7F	UNEXPECTED_KERNEL_MODE_TRAP	Исключение процессора, необработанное операционной системой; обычно возникает вследствие неисправности оборудования (как правило — разгона CPU), его несовместимости с установленными драйверами или алгоритмическими ошибками в самих драйверах; проверьте исправность оборудования и удалите все посторонние драйвера; первый Bug Check-параметр содержит номер исключения и может принимать следующие значения: 0x00 — попытка деления на нуль; 0x01 — исключение системного отладчика; 0x03 — исключение точки останова; 0x04 — переполнение; 0x05 — генерируется инструкцией BOUND;

Таблица 3.1 *(окончание)*

Категория		Описание
hex-код	**Символьное имя**	
0x7F	UNEXPECTED_KERNEL_MODE_TRAP	0x06 — неверный опкод; 0x07 — двойной отказ (Double Fault); описание остальных исключений содержится в документации на процессоры Intel и AMD
0xC2	BAD_POOL_CALLER	Текущий поток вызвал некорректный pool-request, что обычно происходит по причине алгоритмической ошибки, допущенной разработчиком драйвера; однако, судя по всему, и сама система не остается без ошибок, поскольку для устранения этого "голубого экрана смерти" Microsoft рекомендует установить SP2
0xCB	DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS	После завершения процедуры ввода/вывода, драйвер не может освободить заблокированные страницы (см. PROCESS_HAS_LOCKED_PAGES); первый Bug Check-параметр содержит вызываемый, а второй Bug Check-параметр, вызывающий адрес; последний, четвертый параметр указывает на UNICODE-строку с именем драйвера
0xD1	DRIVER_IRQL_NOT_LESS_OR_EQUAL	То же самое, что и IRQL_NOT_LESS_OR_EQUAL
0xE2	MANUALLY_INITIATED_CRAS	Сбой системы, спровоцированный вручную, путем нажатия "горячей" комбинации клавиш <Ctrl>+<Scroll Loock>, при условии, что параметр CrashOnCtrlScroll реестра HKLM\System\CurrentControlSet\Services\i8042prt\Parameters содержит ненулевое значение
0x7A	KERNEL_DATA_INPAGE_ERROR	Страница данных памяти ядра по техническим причинам недоступна, второй Bug Check-параметр содержит статус обмена, четвертый — виртуальный страничный адрес, загрузить который не удалось; возможные причины сбоя — те же дефектные сектора, попавшие в файл pagefile.sys, сбои дискового контроллера, ну и вирусы, наконец

Восстановление системы после критического сбоя

> Неестественное, почти половое влечение к кнопке F8 появилось в Кролике совершенно не внезапно.
>
> *Щербаков Андрей. "14400 бод и 19200 юзеров, и те же самые все-все-все..."*

Операционные системы семейства NT достаточно безболезненно переносят критические сбои, даже если те произошли в самый неудобный момент (например, в период дефрагментации диска). Отказоустойчивый драйвер файловой системы все сделает сам (хотя запустить файл chkdsk.exe все же не помешает).

Если был выбран "полный дамп памяти" или "дамп памяти ядра", то при следующей успешной загрузке системы жесткий диск будет долго "молотить" головкой, даже если к нему и не происходит никаких обращений. Не пугайтесь! Просто Windows перемещает дамп из виртуальной памяти на место его постоянного "проживания". Запустив "Диспетчер Задач", вы увидите новый процесс в списке — SaveDump.exe, — вот он этим и занимается. Необходимость в подобной двухтактной схеме "сброса" дампа объясняется тем, что в момент возникновения критической ошибки работоспособность драйверов файловой системы уже не гарантируется и операционная система не может позволить себе их использовать, ограничиваясь временным размещением дампа в виртуальной памяти. Кстати, если имеющегося объема виртуальной памяти окажется недостаточно (Дополнительно → Параметры быстродействия → Виртуальная память), "сброс" дампа окажется невозможным.

Если же система от загрузки отказывается, упорно забрасывая вас "голубыми экранами смерти", вспомните о существовании клавиши <F8> и выберите пункт *"Загрузка последней удачной конфигурации"* (*"Last Known Good Configuration"*). Более радикальной мерой является запуск системы в безопасном (safe) режиме с минимумом загружаемых служб и драйверов. Переустановка системы — это крайняя мера, и без особой нужны к ней лучше не прибегать. Лучше войдите в "консоль восстановления" и переместите файл дампа на другую машину для его дальнейшего исследования.

Подключение дампа памяти

Для подключения дампа памяти к отладчику Windows Debugger (windbg.exe) в меню **File** выберите пункт **Crash Dump** или воспользуйтесь "горячей" комбинацией клавиш <Ctrl>+<D>. В отладчике i386kd.exe для той же цели служит ключ "−z" командной строки, за которым следует полный путь к файлу дампа, отделенный от ключа одним или несколькими пробелами, при этом переменная окружения `_NT_SYMBOL_PATH` должна быть определена и должна

содержать полный путь к файлам символьных идентификаторов, в противном случае отладчик аварийно завершит свою работу. Как один из вариантов, можно указать в командной строке ключ "–y", и тогда экран консоли будет выглядеть так: `i386kd -z C:\WINNT\memory.dmp -y C:\WINNT\Symbols`, причем отладчик следует вызывать из Checked Build Environment/Free Build Environment консоли, находящейся в папке Windows 2000 DDK, иначе у вас ничего не получится.

Хорошая идея — ассоциировать DMP-файлы с отладчиком i386kd.exe, запуская их одним нажатием клавиши <Enter> из менеджера FAR. Впрочем, выбор средства анализа — дело вкуса. Кому-то нравится KAnalyze, а кому-то достаточно и простенького DumpChk. Выбор аналитических инструментов чрезвычайно велик (один лишь DDK содержит четыре из них!), и, чтобы хоть как-то определиться с выбором, мы остановимся на i386kd.exe, также называемом Kernel Debugger.

Как только консоль отладчика появится на экране (а Kernel Debugger — это консольное приложение, горячо любимое всеми, кто провел свою молодость за текстовыми терминалами), курсор наскоро дизассемблирует текущую машинную инструкцию и своим тревожным мерцанием затягивает нас в пучину машинного кода. "Ну что, будем глазки строить или все-таки дизассемблировать?" — незлобно ворчим мы, выбивая на клавиатуре команду "u", заставляющую отладчик продолжить дизассемблирование.

Судя по символьным идентификаторам `PspUnhandledExceptionInSystemThread` и `KeBugCheckEx`, мы находимся глубоко в ядре, а точнее — в окрестностях того кода, что выводит BSOD на экран (листинг 3.22).

Листинг 3.22. Результат дизассемблирования подключенного дампа памяти с текущего адреса

```
8045249c 6a01          push     0x1
kd>u
_PspUnhandledExceptionInSystemThread@4:
80452484 8B442404      mov      eax, dword ptr [esp+4]
80452488 8B00          mov      eax, dword ptr [eax]
8045248A FF7018        push     dword ptr [eax+18h]
8045248D FF7014        push     dword ptr [eax+14h]
80452490 FF700C        push     dword ptr [eax+0Ch]
80452493 FF30          push     dword ptr [eax]
80452495 6A1E          push     1Eh
80452497 E8789AFDFF    call     _KeBugCheckEx@20
8045249C 6A01          push     1
8045249E 58            pop      eax
8045249F C20400        ret      4
```

В стеке ничего интересного также не содержится, вот судите сами (листинг 3.23) (просмотр содержимого стека осуществляется командной `kb`).

Листинг 3.23. Содержимое стека не дает никаких намеков на природу истинного виновника

```
kd> kb
ChildEBP RetAddr  Args to Child
f403f71c 8045251c f403f744 8045cc77 f403f74c ntoskrnl!PspUnhandledExceptionInSystemThread+0x18
f403fddc 80465b62 80418ada 00000001 00000000 ntoskrnl!PspSystemThreadStartup+0x5e
00000000 00000000 00000000 00000000 00000000 ntoskrnl!KiThreadStartup+0x16
```

Такой поворот событий ставит нас в тупик. Сколько бы мы ни дизассемблировали ядро, это ни на йоту не приблизит нас к источнику критической ошибки. Что ж, все вполне логично. Текущий адрес (`8045249Ch`) лежит далеко за пределами драйвера-убийцы (`0BE80A00h`). Хорошо, давайте развернемся и пойдем другим путем. Помните тот адрес, что высвечивал "голубой экран смерти"? Не помните — не беда! Если это только не запрещено настройками, копии всех голубых экранов сохраняются в Журнале системы. Откроем его: **Панель управления → Администрирование → Просмотр событий** (листинг 3.24).

Листинг 3.24. Копия "голубого экрана смерти", сохраненная в системном журнале

```
Компьютер был перезагружен после критической ошибки:
0x0000001e (0xc0000005, 0xbe80b000, 0x00000000, 0x00000000).
Microsoft Windows 2000 [v15.2195]
Копия памяти сохранена: C:\WINNT\MEMORY.DMP.
```

Отталкиваясь от категории критической ошибки (0x1E), мы без труда сможем определить адрес инструкции-убийцы — `0xBE80B000` (в приведенном листинге 3.24 он выделен полужирным шрифтом). Применяем команду `u BE80B000` для просмотра его содержимого и видим следующий результат (листинг 3.25).

Листинг 3.25. Результат дизассемблирования дампа памяти по адресу, сообщенному "голубым экраном смерти"

```
kd>u 0xBE80B000
be80b000 a100000000     mov     eax,[00000000]
be80b005 c20800     ret     0x8
be80b008 90          nop
```

```
be80b009 90          nop
be80b00a 90          nop
be80b00b 90          nop
be80b00c 90          nop
be80b00d 90          nop
```

Ага! Вот это уже больше похоже на истину! Инструкция, на которую указывает курсор (в тексте она выделена черным цветом), обращается к ячейке с нулевым адресом, возбуждая тем самым губительное для системы исключение. Теперь мы точно знаем, какая ветка программы вызвала сбой.

Хорошо, а как быть, если копии экрана смерти в нашем распоряжении нет? На самом деле, синий экран всегда с нами, надо только знать, где искать! Попробуйте открыть файл дампа в любом HEX-редакторе, и вы обнаружите следующие строки (листинг 3.26).

Листинг 3.26. Копия голубого экрана в заголовке дампа программы

```
00000000:  50 41 47 45 44 55 4D 50 | 0F 00 00 00 93 08 00 00   PAGEDUMP☼    У◘
00000010:  00 00 03 00 00 80 8B 81 | C0 A4 46 80 80 A1 46 80    ♥  АЛБ└дFААбFA
00000020:  4C 01 00 00 01 00 00 00 | 1E 00 00 00 05 00 00 C0   L☺  ☺   ▲   ♣  └
00000030:  00 B0 80 BE 00 00 00 00 | 00 00 00 00 00 41 47 45   ░A╝          AGE
```

С первого же взгляда удается опознать все основные Bug Check-параметры: `1E 00 00 00` — это код категории сбоя 0x1E (на процессорах x86 наименее значимый байт располагается по меньшему адресу, то есть все числа записываются в обратном порядке); `05 00 00 C0` — код исключения ACCESS VIOLATION; а `00 B0 80 BE` — и есть адрес машинной команды, породившей это исключение. В комбинации же `0F 00 00 00 93 08` легко узнается номер билда системы, стоит только записать его в десятичной нотации.

Для просмотра Bug Check-параметров в более удобочитаемом виде можно воспользоваться следующей командой отладчика: `dd KiBugCheckData` (листинг 3.27).

Листинг 3.27. Bug Check-параметры, отображаемые в удобочитаемом виде

```
kd> dd KiBugCheckData
dd KiBugCheckData
8047e6c0  0000001e c0000005 be80b000 00000000
8047e6d0  00000000 00000000 00000001 00000000
8047e6e0  00000000 00000000 00000000 00000000
8047e6f0  00000000 00000000 00000000 00000000
8047e700  00000000 00000000 00000000 00000000
```

```
8047e710  00000000 00000000 00000000 00000000
8047e720  00000000 00000000 00000000 00000000
8047e730  00000000 e0ffffff edffffff 00020000
```

Другие полезные команды: `!drivers` — выводящая список драйверов, загруженных на момент сбоя, `!arbiter` — показывающая всех арбитров вместе с диапазонами арбитража, `!filecache` — отображающая информацию о кэше файловой системы и PT, `!vm` — отчитывающаяся об использовании виртуальной памяти и т. д. — всех не перечислишь! (полный перечень команд вы найдете в руководстве по своему любимому отладчику).

Конечно, в реальной жизни определить истинного виновника краха системы намного сложнее, поскольку всякий нормальный драйвер состоит из множества сложно взаимодействующих функций, образующих запутанные иерархические комплексы, местами пересеченные туннелями глобальных переменных, превращающих драйвер в самый настоящий лабиринт. Приведем только один пример. Конструкция вида `mov eax, [ebx]`, где `ebx == 0`, работает вполне нормально, послушно возбуждая исключение, и пытаться "поговорить с ней по-мужски" — бессмысленно! Нужно найти тот код, который записывает в регистр EBX нулевое значение, и сделать это непросто. Можно, конечно, просто прокрутить экран вверх, надеясь, что на данном участке программный код выполнялся линейно, но никаких гарантий, что это действительно так, у нас нет, равно как нет и возможности обратной трассировки (back trace). Грубо говоря, адрес предшествующей машинной инструкции нам неизвестен, и "закладываться" на прокрутку экрана нельзя!

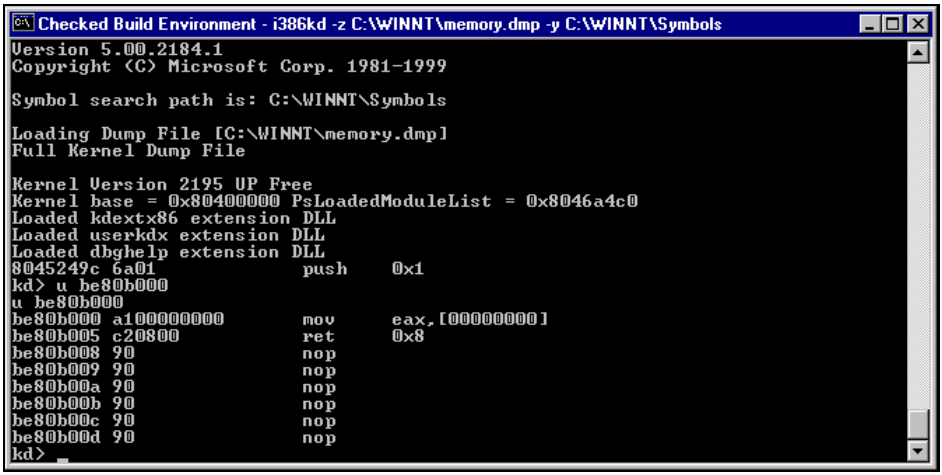


Рис. 3.5. Отладчик i386kd за работой (пиво, бутерброды прилагаются); несмотря на свою отталкивающую внешность, это чрезвычайно мощный и удобный в работе инструмент, позволяющий проворачивать умопомрачительные пассажи нажатием всего пары-тройки клавиш (одна из которых вызывает ваш собственный скрипт)

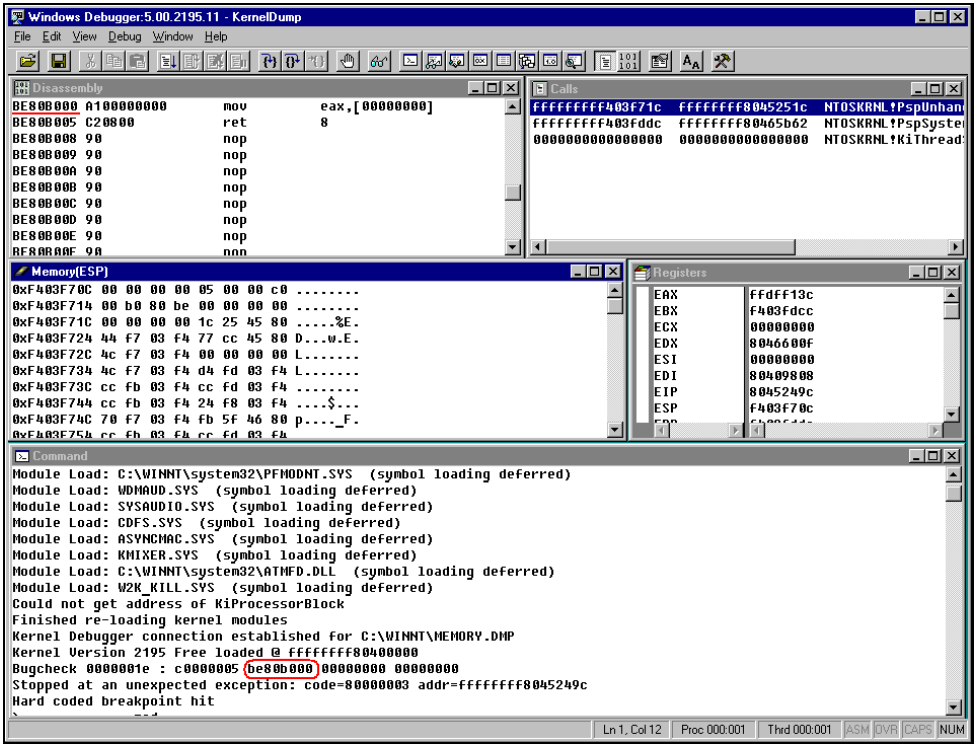
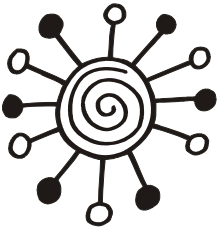


Рис. 3.6. Windows Debugger с загруженным дампом памяти. Обратите внимание: отладчик самостоятельно высвечивает Bug Check-коды, не ожидая, пока мы об этом его попросим, а при попытке дизассемблирования инструкции, возбудившей исключение, на экране выскакивает "Module Load: W2K_KILL.SYS", сообщающая нам имя драйвера-убийцы. Вроде бы мелочь, а как приятно!

Загрузив подопытный драйвер в любой интеллектуальный дизассемблер, автоматически восстанавливающий перекрестные ссылки (например, дизассемблер IDA Pro), мы получим более или менее полное представление о топологии управляющих ветвей программы. Конечно, дизассемблирование в силу своей статической природы, не гарантирует, что управление не перекинулось откуда-то еще, но, по крайней мере, сужает круг поиска. Вообще же, о дизассемблировании написано множество хороших книг (и "Фундаментальные основы хакерства" Криса Касперски в том числе), поэтому не будем останавливаться на этом вопросе, а просто пожелаем всем читателям удачи.

Глава 4



Интерфейсы взаимодействия с оборудованием

Стандарты — вещь хорошая. Всегда есть из чего выбрать.

Фольклор

Существует множество способов взаимодействия с оборудованием. В зависимости от специфики решаемой задачи и специфики самого оборудования, предпочтение отдается либо тем, либо иным интерфейсам управления. На самом высоком уровне интерфейсной иерархии располагается семейство API-функций операционной системы, реализующих типовые операции ввода/вывода (такие, например, как открыть файл, прочитать данные из файла). Для подавляющего большинства прикладных программ этого оказывается более, чем достаточно, однако даже простейший копировщик на этом наборе, увы, не напишешь, и приходится спускаться по меньшей мере на один уровень вглубь, обращаясь непосредственно к *драйверу* данного устройства.

Стандартные дисковые драйвера, входящие в состав операционных систем Windows 9x и Windows NT, поддерживают довольно ограниченное количество типовых команд (прочитать сектор, просмотреть TOC и т. д.), не позволяющих в должной мере реализовать все возможности современных приводов CD-ROM/R/RW, однако для написания простейших защитных механизмов их функционала вполне достаточно.

Подавляющее большинство защитных механизмов данного типа безупречно копируется штатными копировщиками, что, собственно, и неудивительно: ведь и копировщик, и защита "кормятся" из одной и той же "кормушки", простите, используют идентичный набор управляющих команд, работающих с устройством на логическом уровне.

Для создания устойчивой к взлому защиты мы должны опуститься "на самое дно колодца", заговорив с устройством на родном для него языке. Несмотря на то, что контроллеры оптических накопителей поддерживают высокоуровневый набор управляющих команд (намного более высокоуровневый,

чем приводы гибких дисков), несмотря на то, что интерфейс привода абстрагирован от конкретного физического оборудования и что диски CD-ROM/R/RW изначально не были ориентированы на защиту, создание практически не копируемых дисков на этом уровне все-таки возможно.

Вопреки расхожему мнению, для низкоуровневого управления накопителями совершенно необязательно прибегать к написанию своего собственного драйвера. Все необходимые драйвера давным-давно написаны до нас, и на выбор разработчика предоставляется несколько конкурирующих интерфейсов, обеспечивающих низкоуровневое взаимодействие со SCSI/ATAPI-устройствами с прикладного уровня. Это и ASPI *(Advanced SCSI Programming Interface)*, и SPTI *(SCSI Pass Through IOCTLs)*, и MSCDEX *(MS-DOS CD-ROM Extension)* (ныне практически забытый, но все же поддерживаемый операционными системами Windows 98 и ME). Каждый из интерфейсов имеет свои достоинства и свои недостатки, поэтому коммерческие программные пакеты вынуждены поддерживать их все.

Поскольку программирование оптических накопителей выходит далеко за рамки предмета защиты лазерных дисков (основного предмета данной книги!), то интерфейсы взаимодействия с устройствами будут рассмотрены максимально кратко и упрощенно. К слову сказать, ряд книг, посвященных непосредственно управлению SCSI/ATAPI-устройствами, значительно проигрывает настоящему разделу (взять, к примеру, *"Программирование устройств SCSI и IDE"* Всеволода Несвижского — СПб.: БХВ-Петербург, 2003 г., описывающего исключительно интерфейс ASPI и к тому же описывающего его *неправильно*).

Информации, приведенной далее, вполне достаточно для самостоятельного изучения всех вышеперечисленных интерфейсов с абсолютного нуля. Даже если вам никогда до этого не приходилось сталкиваться с программированием SCSI/ATAPI-устройств, вы вряд ли будете испытывать какие-либо затруднения по ходу чтения книги (не говоря уж о том, что данная книга научит вас основам шпионажа за чужими программами и взлому оных, но это строго между нами!).

Доступ через CD-ROM-драйвер

Управление драйверами устройств в операционных системах семейства Windows осуществляется посредством вызова функции `DeviceIoControl`, отвечающей за посылку специальных FSCTL/IOCTL-команд. Префикс *FS*- свидетельствует о принадлежности данной команды к файловой системе и в контексте настоящей публикации не представляет для нас никакого интереса. Команды с префиксом *IO*- относятся к устройству ввода/вывода, а точнее — к его драйверу.

Функция `DeviceIoControl` просто передает такую команду, как она есть, совершенно не задумываясь о ее "физическом смысле". Следовательно,

совершенно бессмысленно искать перечень доступных IOCTL-команд в описании функции `DeviceIoControl`. Их там нет! Точнее, здесь приводятся лишь стандартные IOCTL-команды, а вся остальная информация по этому вопросу содержится в DDK *(Device Driver Kit)*.

Там, в частности, мы найдем, что для чтения оглавления диска используется команда `IOCTL_CDROM_READ_TOC`, а для перечисления адресов сессий многосессионных дисков — `IOCTL_CDROM_GET_LAST_SESSION`. Также обратите свое внимание на команду `IOCTL_CDROM_READ_Q_CHANNEL`, обеспечивающую извлечение информации из Q-канала подкода (для извлечения ключевых меток — это актуально).

Чтение "сырых" секторов осуществляется командой `IOCTL_CDROM_RAW_READ`, возможности которой, к сожалению, ограничены только лишь дисками CD-DA *(Compact Disk Digital Audio)*. Посекторное чтение с дисков CD-DATA ни на сыром, ни на "сухом" уровнях *не поддерживается*. В соответствии с принятой политикой безопасности, никакое приложение не должно действовать в обход системы безопасности, в противном случае, злоумышленник сможет без труда добраться до конфиденциальных данных, просто прочитав диск на секторном уровне. Штатные драйвера, которыми укомплектованы операционные системы семейства Windows, всецело следуют этим требованиям, хотя сторонние разработчики могут, при желании, и нарушить этот запрет. В состав NT DDK входит исходный текст демонстрационного CD-ROM-драйвера (NTDDK\src\storage\class\cdrom\), который после небольшой обработки напильником согласится читать диски всех типов, не задавая при этом глупых вопросов. Найдите в теле файла cdrom.c следующую строку `if (rawReadInfo->TrackMode == CDDA) {` и перейдите к ветке, чей `OperationCode` равен `SCSIOP_READ`. А теперь модифицируйте код так, чтобы эта ветка получала управление и во всех остальных случаях.

Замечание

Команда `IRP_MJ_READ`, присутствующая в DDK, и по идее обеспечивающая возможность чтения отдельных логических блоков, является внутренней функцией драйвера, и доступ к последней с прикладного уровня закрыт; пытаться использовать ее в паре с функцией `DeviceIoControl` — бессмысленно.

В табл. 4.1 приведено описание IOCTL-команд штатного драйвера CD-ROM (за более подробной информацией обращайтесь к DDK).

Таблица 4.1. *Описание IOCTL-команд штатного драйвера CD-ROM*

IOCTL-команда	Описание
`IOCTL_CDROM_CHECK_VERIFY`, `IOCTL_STORAGE_CHECK_VERIFY` (0x24800h)	Определяет факт смены диска (открытия/закрытия лотка)

Таблица 4.1 *(окончание)*

IOCTL-команда	Описание
IOCTL_CDROM_CLOSE_DOOR* IOCTL_STORAGE_LOAD_MEDIA (0x2D480Ch)	Закрывает лоток привода
IOCTL_CDROM_FIND_NEW_DEVICES, IOCTL_STORAGE_FIND_NEW_DEVICES (0x24818h)	Перечисляет новые приводы, подключенные после загрузки системы или последнего вызова данной команды
IOCTL_CDROM_GET_CONTROL	Сообщает текущую позицию воспроизведения аудио
IOCTL_CDROM_GET_DRIVE_GEOMETRY (0x2404Ch)	Определяет тип лазерного диска и его геометрию (количество секторов на диске, размер одного сектора и т. д.)
IOCTL_CDROM_GET_LAST_SESSION (0x24038h)	Перечисляет стартовые адреса сессий и записывает их в буфер TOC, читаемый командой IOCTL_CDROM_READ_TOC
IOCTL_CDROM_GET_VOLUME (0x24014h)	Возвращает текущий уровень громкости с CD-ROM
IOCTL_CDROM_PAUSE_AUDIO (0x2400Ch)	Временно останавливает воспроизведение аудио
IOCTL_CDROM_PLAY_AUDIO_MSF (0x24018h)	Инициирует процесс воспроизведения аудио от сих до сих
IOCTL_CDROM_RAW_READ (0x2403Eh)	Выполняет "сырое" чтение секторов с аудиодисков
IOCTL_CDROM_READ_Q_CHANNEL (0x2402Ch)	Читает данные Q-канала подкода
IOCTL_CDROM_READ_TOC (0x24000h)	Читает оглавление диска
IOCTL_CDROM_RESUME_AUDIO (0x24010h)	Продолжает воспроизведение аудио
IOCTL_CDROM_SEEK_AUDIO_MSF (0x24004h)	Позиционирует оптическую головку
IOCTL_CDROM_SET_VOLUME (0x24028h)	Устанавливает уровень громкости с CD-ROM
IOCTL_CDROM_STOP_AUDIO (0x24008h)	Останавливает воспроизведение аудио

* — устарело и ныне удалено из DDK

Функции DeviceIoControl всегда предшествует вызов функции CreateFile, возвращающей дескриптор соответствующего устройства, задаваемого в виде \\.\X:, где X — буквенное обозначение того привода, с которым мы собрались работать, причем флаг dwCreationDisposition должен быть установлен

в состояние OPEN_EXISTING, иначе вы потерпите неудачу. Типовой пример вызова функции приведен далее в листингах 4.1—4.2.

Замечание

Windows NT регистрирует устройство с именем \\.\CdRom$_x$, где x — номер привода, считая от нуля, которое ссылается на тот же самый драйвер, что и буквенное обозначение диска, и обладает тем же самым набором функций.

Листинг 4.1. Пример открытия устройства

```
HANDLE hCD;                                          // дескриптор привода
hCD=CreateFile("\\\\.\\X:", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0,0);
if (hCD == INVALID_HANDLE_VALUE)                     // ошибка
```

Прототип самой же функции DeviceIoControl выглядит так, как это показано в листинге 4.2.

Листинг 4.2. Прототип функции DeviceIoControl

```
BOOL DeviceIoControl(
  HANDLE hDevice,               // дескриптор устройства
  DWORD dwIoControlCode,        // IOCTL-код команды для выполнения
  LPVOID lpInBuffer,            // указатель на входной буфер
                                // (Irp->AssociatedIrp.SystemBuffer)
  DWORD nInBufferSize,          // размер входного буфера в байтах
  LPVOID lpOutBuffer,           // указатель на выходной буфер
                                // (Irp->AssociatedIrp.SystemBuffer)
  DWORD nOutBufferSize,         // размер выходного буфера в байтах
  LPDWORD lpBytesReturned,      // указатель на счетчик кол-ва возвращенных байт
  LPOVERLAPPED lpOverlapped     // указатель на структуру для асинхронных операций
);
```

Здесь:

- hDevice — тот самый дескриптор, который был только что возвращен функцией CreateFile;
- dwIoControlCode — IOCTL-код нашей операции;
- lpInBuffer — указатель на буфер, содержащий данные, подготовленные для передачи устройству (как правило, аргументы команды). В процессе выполнения функции содержимое буфера копируется в Irp->AssociatedIrp.SystemBuffer. Это на тот случай, чтобы, увидев такую абракадабру в DDK, вы не хватались за сердце и не пытались "скормить" функции DeviceIoControl всю IRP-структуру;

- `nInBufferSize` — размер входного буфера в байтах. В процессе выполнения функции он копируется в структуру `Parameters.DeviceIoControl.InputBufferLength`;
- `lpOutBuffer` — указатель на выходной буфер, в который помещается содержимое `Irp->AssociatedIrp.SystemBuffer`;
- `nOutBuffersSize` — указатель на двойное слово, в которое будет записано количество байт, возвращенных драйвером через выходной буфер.

Если операция завершилась успешно, функция возвращает ненулевое значение, и нуль — в противном случае. За более подробной информацией об ошибке возвращайтесь к функции `GetLastError`.

Передача IOCTL-команд устройству не требует наличия прав администратора (за тем исключением, когда устройство открывается с флагом `GENETIC_WRITE`), что значительно увеличивает "эргономичность" защитных механизмов, базирующихся на ее основе *(Кстати, о защитных механизмах, точнее их стойкости ко взлому. Поскольку функция `DeviceIoControl` к числу популярных явно не относится, она демаскирует "штаб-квартиру" защитного механизма, и его становится очень легко "запеленговать". Достаточно поставить на функцию `DeviceIoControl` точку останова и дождаться, пока передаваемая ей IOCTL-команда не примет одно из вышеперечисленных значений. На функцию `CreateFile` точку останова лучше не ставить, т. к. это даст множество ложных срабатываний (`CreateFile` вызывается всякий раз при открытии/создании какого-либо файла). А вот попробовать поискать в теле программы текстовую строку "\\.\" все-таки стоит. И, если она действительно будет найдена, вам останется лишь подбежать курсором к перекрестной ссылке и нажать клавишу <Enter>. Все! Защитный код перед вами!).*

Для лучшего понимания данного способа взаимодействия между прикладной программой и драйвером далее в листинге 4.3 приведен ключевой фрагмент функции, как раз и осуществляющей такое взаимодействие (обработка ошибок по соображениям наглядности опущена).

Листинг 4.3. Функция, демонстрирующая технику чтения "сырых" секторов через CDFS-драйвер (только для дисков CD-DA!)

```
//--[ReadCDDA]-------------------------------------------------------------
//
//                  читает сектор в "сыром" виде с CDDA-дисков
//                  ==========================================
// ARG:
//      drive           - имя устройства, с которого читать (например "\\\\.\\X:")
//      start_sector    - номер первого читаемого сектора
//      n_sec           - сколько секторов читать
```

```
//
// RET:
//      == 0            - ошибка
//      != 0            - указатель на буфер, содержащий считанные сектора
//
// NOTE:
//      функция поддерживает только диски тех типов, что поддерживает драйвер
//      CDFS, который она и использует, а штатный драйвер Windows NT поддерживает
//      лишь CDDA-диски
//-----------------------------------------------------------------------------
char* ReadCDDA(char *drive, int start_sector, int n_sec)
{
      // поддерживаемые типы треков
      typedef enum _TRACK_MODE_TYPE {
            YellowMode2,                    // native MODE 2 (не CD-data)
            XAForm2,                        // XA MODE 2 Form 2 (VideoCD)
            CDDA                            // Audio-CD
      } TRACK_MODE_TYPE, *PTRACK_MODE_TYPE;

      // аргумент IOCTL-команды IOCTL_RAW_READ
      typedef struct __RAW_READ_INFO {
            LARGE_INTEGER      DiskOffset;    // смещение в байтах лог. блоков
            ULONG              SectorCount;   // кол-во секторов для чтения
            TRACK_MODE_TYPE    TrackMode;     // режим читаемого трека
      } RAW_READ_INFO, *PRAW_READ_INFO;

      #define CDROM_RAW_SECTOR_SIZE        2352
      #define CDROM_SECTOR_SIZE            2048

      int      a;
      HANDLE   hCD;
      DWORD    x_size;
      char     *szDrive;
      BOOL     fResult = 0;
      unsigned char      *buf;
      RAW_READ_INFO      rawRead;

      // ПОДГОТАВЛИВАЕМ СТРУКТУРУ RAW_READ_INFO, передаваемую драйверу CD-ROM'a
      rawRead.TrackMode             = CDDA;          // тип диска - Audio CD
      rawRead.SectorCount           = n_sec;           // кол-во читаемых секторов
      rawRead.DiskOffset.QuadPart   = start_sector * CDROM_SECTOR_SIZE;
      //                                             ^^^^^^^^^^^^^^^^^^
```

```
    // стартовый сектор задается отнюдь не своим логическим номером,
    // а номером своего первого байта. сквозная нумерация байтов от
    // первого до последнего байта диска теоритически обеспечивает
    // полное абстрагирование от конкретного оборудования
    // (размер одного сектора возвращается IOCTL-командой
    // IOCTRL_CDROM_GET_DRIVE_GEOMETRY), но практически архитекторами
    // драйвера допущен грубый ляп, "благодаря" которому драйвер
    // принимает вовсе не сквозные номера байт,
    // а start_address * CDROM_SECTOR_SIZE, где
    // SECTOR_SIZE - размер логического блока, который в данном случае равен
    // стандартному размеру сектора CDDATA-диска (2048 байт для справки),
    // в то время как размер сектора CDDA-дисков составляет 2352 байта
    // поэтому DiskOffset равен start_secor * CDROM_SECTOR_SIZE, а размер
    // буфера должен быть равен start_secor * CDROM_RAW_SECTOR_SIZE

    // ВЫДЕЛЯЕМ ПАМЯТЬ
    buf = malloc(CDROM_RAW_SECTOR_SIZE * n_sec);

    // ПОЛУЧАЕМ ДЕСКРИПТОР УСТРОЙСТВА
    hCD = CreateFile(drive,GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,0,0);
    if (hCD != INVALID_HANDLE_VALUE)

    // ПЕРЕДАЕМ ДРАЙВЕРУ ПРИВОДА КОМАНДУ IOCTL_CDROM_RAW_READ
    fResult = DeviceIoControl(      hCD, 0x2403E /* IOCTL_CDROM_RAW_READ */,
                                    &rawRead, sizeof(RAW_READ_INFO),
                                    buf, CDROM_RAW_SECTOR_SIZE*n_sec,
                                    &x_size, (LPOVERLAPPED) NULL);

    // ВЫВОДИМ РЕЗУЛЬТАТ (если есть, что выводить)
    if (fResult)
         for (a = 0; a <= x_size; ++a) printf("%02X%s",buf[a],(a%24)?" ":"\n");
    else
         printf("-ERROR"); printf("\n");

    // СВАЛИВАЕМ
    CloseHandle(hCD); return (fResult)?buf:0;
}
```

Еще один демонстрационный пример приведен в листинге 4.4, изучение которого бывает полезно при анализе некоторых защищенных дисков. Он иллюстрирует технику чтения TOC (*Table Of Content*) — своеобразный аналог таблицы разделов лазерных аудиодисков.

Листинг 4.4. Пример программы, взаимодействующей с CDFS-драйвером через IOCTL и читающей содержимое TOC (с расшифровкой)

```
/*-----------------------------------------------------------------------------
 *
 *                        ЧТЕНИЕ И РАСШИФРОВКА TOC
 *                        ========================
 *
 * build 0x001 @ 26.05.2003
-----------------------------------------------------------------------------*/
main(int argc, char **argv)
{
   int              a;
   HANDLE           hCD;
   unsigned char    *buf;
   WORD             TOC_SIZE;
   BYTE             n_track;
   DWORD            x_size,b;

   #define DEF_X      "\\\\.\\G:"                  // привод по умолчанию
   #define argCD      ((argc>1)?argv[1]:DEF_X)

   // ПРОВЕРКА АРГУМЕНТОВ
   if (argc < 2) {fprintf(stderr, "USAGE: IOCTL.read.TOC \\\\.\\X:\n"); return 0;}

   // TITLE
   fprintf(stderr,"simple TOC reader via IOCTL\n");

   // ВЫДЕЛЯЕМ ПАМЯТЬ
   buf = (char *) malloc(buf_len);

   // ОТКРЫВАЕМ УСТРОЙСТВО
   hCD=CreateFile(argv[1], GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);

   // ВЫХОДИМ, ЕСЛИ ОШИБКА
   if (hCD == INVALID_HANDLE_VALUE)
         {fprintf(stderr,"-ERR: %x\n", GetLastError()); return 0;}

   // ПЕРЕДАЕМ ДРАЙВЕРУ КОМАНДУ CDROM_READ_TOC
   if (DeviceIoControl(      hCD, 0x24000 /* IOCTL_READ_TOC */,
                             0, 0, buf, buf_len, &x_size, 0) != 0)
   {
```

```
        // ПОЛУЧАЕМ ДЛИНУ TOC'a (она записана в обратном порядке)
        TOC_SIZE = buf[0]*0x100L + buf[1];
        printf("TOC Data Length........%d\n",TOC_SIZE);

        // декодируем остальную информацию
        printf("First Session Number...%d\n",buf[2]);
        printf("Last Session Number....%d\n\n",(n_track=buf[3]));
        for (a = 1; a <= n_track; a++)
        {
              printf("track %d\n{\n",a);
              printf("\treserved.............%x\n",buf[a * 8 - 4]);
              printf("\tADR|control..........%d\n",buf[a * 8 - 3]);
              printf("\ttrack number.........%d\n",buf[a * 8 - 2]);
              printf("\treserved.............%d\n",buf[a * 8 - 1]);
              printf("\treserved.............%d\n",buf[a * 8 + 0]);
              printf("\tmin..................%d\n",buf[a * 8 + 1]);
              printf("\tsec..................%d\n",buf[a * 8 + 2]);
              printf("\tframe................%d\n",buf[a * 8 + 3]);
              printf("}\n\n");
        }

        // выводим содержимое TOC'a в "сыром" виде
        printf("\n\t\t\t* * * RAW * * *\n");
        for(a = 0; a < x_size; a++)
              printf("%02X%s",(unsigned char)buf[a],((a+1)%22)?" ":"\n");
        printf("\n\t\t\t* * *  *  * * *\n");
    }
}
```

Доступ через Cooked-Mode (режим блочного чтения)

Операционная система Windows NT выгодно отличается тем, что поддерживает режим блочного чтения с устройства, — так называемый *Cooked-Mode*, в котором все содержимое диска трактуется как один большой файл. По этому "файлу" можно перемещаться вызовом функции `SetFilePointer` и читать/писать отдельные сектора посредством вызовов функций `ReadFile`/`WriteFile` соответственно. Текущая позиция указателя задается в байтах (не секторах!), однако значение указателя обязательно должно быть кратным логической длине сектора (512 байт для гибких/жестких дисков и 2048 байт для CD-ROM), в противном случае произойдет ошибка. Количество байт,

читаемых (записываемых) за один раз, также должно укладываться в целое число секторов. Попытка прочитать сектор по "кусочкам" ни к чему не приведет.

Несмотря на всю изящность и простоту программной реализации, данному способу взаимодействия с приводом присущи серьезные недостатки. Во-первых, он не работает с файловыми системами, отличными от ISO 9660/Juliet и *High Sierra File System*. В переводе на нормальный человеческий язык это означает, что для чтения секторов с аудиодисков режим блочного чтения непригоден и подходит лишь для обработки дисков с данными. Во-вторых, чтение "сырых" секторов в Cooked-Mode невозможно, и нам придется довольствоваться лишь той их частью, что содержит пользовательские данные (User-Data). Такое положение дел значительно ослабляет стойкость защитного механизма и позволяет легко ввести его в заблуждение. Допустим, защита, основанная на привязке к физическим дефектам поверхности носителя, пытается прочесть ключевой сектор на предмет проверки его читабельности. Поскольку содержимое кодов коррекции защитному механизму недоступно, он не может отличить действительные физические дефекты от их грубой имитации (то есть умышленного искажения ECC/EDC-кодов копировщиком с целью эмуляции неустранимых ошибок чтения).

Проверить, использует ли защита данный способ доступа к диску или нет, можно следующим образом: просто установите точку останова на функцию `CreateFile`, заставив отладчик всплывать в том и только в том случае, если первые четыре символа имени открываемого файла равны `\\.\` (то есть функция открывает не файл, а устройство). Например, это может выглядеть так: `bpx CreateFileA if (*esp->4=='\\\\.\\')`, затем нам останется лишь убедиться в том, что за последней косой чертой следует буква именно того привода, который нам нужен (на компьютере автора это привод `\\.\G:`). Дождавшись выхода из функции `CreateFile` по команде `P RET` и подсмотрев возвращенный ей дескриптор устройства (который будет содержаться в регистре EAX), мы сможем перехватить все вызовы функций `SetFilePointer`/`ReadFile`, анализ окрестностей которых и "разоблачит" алгоритм работы защитного механизма.

Демонстрационный пример, приведенный в листинге 4.5, представляет собой вполне законченную утилиту для "грабежа" дисков с данными на секторном уровне с последующей записью всего "награбленного" в файл.

Листинг 4.5. Пример, демонстрирующий технику чтения секторов в Cooked-Mode

```
/*-----------------------------------------------------------------------------
 *
 *                    ЧИТАЕТ СЕКТОРА С CD-ROM В БЛОЧНОМ РЕЖИМЕ
```

```
 *                  ========================================
 *
 *      данная программа работает только под Windows NT, не требуя для себя
 *      прав администратора
 *
 * Build 0x001 @ 19.05.03
---------------------------------------------------------------------------- */
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>

// ПАРАМЕТРЫ ПО УМОЛЧАНИЮ
#define DEF_FN                          "sector"
#define DEF_TO                          0x666
#define DEF_FROM                        0x000
#define CDROM_SECTOR_SIZE      2048            // for MODE1/MODE2FORM1 only!

// АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ
#define argCD       (argv[1])
#define argFN       ((argc > 2)?argv[2]      :DEF_FN)
#define argFROM     ((argc > 3)?atol(argv[3]):DEF_FROM)
#define argTO       ((argc>4)?(atol(argv[4])>argFROM)?atol(argv[4]):argFROM:DEF_TO)

main(int argc, char **argv)
{
   int              a;
   FILE             *f;
   HANDLE           hCD;
   char             *buf;
   DWORD            x_read;
   char             buf_n[1024];

   // ПРОВЕРЯЕМ АРГУМЕНТЫ
   if (argc<2)
   {
        printf("USAGE: cooked.sector.read PhysCD [filename] [from] [to]\n");
        printf("\tPhysCD   - physical name of CD (\"\\\\.\\G:\")\n");
        printf("\tfilename - file name to store follow sector\n");
        printf("\tfrom     - start sector\n");
        printf("\tto       - end sector\n");
        return 0;
   }
```

```
    // TITLE
    fprintf(stderr,"cooked sector reader for NT\n");

    // ВЫДЕЛЯЕМ ПАМЯТЬ
    buf=malloc(CDROM_SECTOR_SIZE);if (!buf){printf("-ERR:low memory\n");return -1;}

    // ОТКРЫВАЕМ УСТРОЙСТВО
    hCD=CreateFile(argCD, GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
    if (hCD == INVALID_HANDLE_VALUE) {
         printf("-ERR: error CreateFile(%s,....)\n", argCD); return -1;
    }

    // INFO
    printf("read sector from %04d to %04d in %s file\n", argFROM, argTO, argFN);

    // ПОЗИЦИОНИРУЕМ УКАЗАТЕЛЬ НА ПЕРВЫЙ ЧИТАЕМЫЙ СЕКТОР
    SetFilePointer (hCD, CDROM_SECTOR_SIZE * argFROM, NULL, FILE_BEGIN);

    // ЧИТАЕМ СЕКТОРА ОДИН ЗА ДРУГИМ
    for (a = argFROM; a <= argTO; a++)
    {
         // читаем очередной сектор
         if (ReadFile(hCD, buf, CDROM_SECTOR_SIZE, &x_read, NULL) && x_read)
         {
                 // записываем только что считанный сектор в файл
                 sprintf(buf_n,"%s[%04d].dat",argFN, a);
                 if (f=fopen(buf_n,"wb")){fwrite(buf, 1, x_read, f); fclose(f);}
                 printf("sector [%04d.%04d] read\r",a, argTO);
         }
                 else
         {
                 printf("sector %04d read error\n",a);
         }
    }
}
```

Доступ через SPTI

Одно из интереснейших архитектурных особенностей операционной системы Windows NT заключается в ее умении взаимодействовать с IDE-устройствами через SCSI-интерфейс! К сожалению, данная технология чрезвычайно скудно

документирована — Platform SDK, MSDN, DDK содержат лишь обрывки информации, а имеющиеся примеры крайне ненаглядны и к тому же выполнены с большим количеством фактических ошибок, так что разобраться с ними под силу лишь профессионалу ну или очень настырному новичку.

Замечание

В общем-то ситуация вполне логична — ведь Microsoft не имеет к ATAPI /SCSI-интерфейсам ни малейшего отношения, и их стандартизацией занимаются совершенно иные комитеты. Однако в "приличных домах" так все-таки не поступают. Вместо того чтобы оставить программиста со своими проблемами наедине, составители документации могли бы по крайней мере нарисовать общую картину взаимодействия. Попробуйте выкачать из Сети тысячи страниц технической документации (большей частью ненужной, но кто ж это знает заранее!) и, проштудировав ее всю, попытаться свести эту разрозненную картину воедино.

И, судя по сообщениям в телеконференциях, многим программистам осилить технику управления устройствами через SCSI-интерфейс так и не удалось, поэтому имеет смысл рассмотреть эту проблему поподробнее.

Для решения поставленной задачи нам понадобятся:

1. **Описание SCSI-интерфейса** (см. документ *"SCSI Architecture Model — 3"*, описывающий общие концепции SCSI-архитектуры и *"SCSI Primary Commands — 3"*, определяющий базовый набор команд для всех SCSI-устройств; черновые версии обоих документов доступны в электронном виде по адресам **http://www.t10.org/ftp/t10/drafts/sam3/sam3r08.pdf** и **http://www.t10.org/ftp/t10/drafts/spc3/spc3r14.pdf** соответственно; в качестве пособия "быстрого старта" рекомендую *"The Linux SCSI programming HOWTO"*, который можно найти по адресу **http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/SCSI-Programming-HOWTO.pdf**).
2. **Описание SCSI-команд, специфичных для оптических накопителей** (см. документ *"Multimedia Commands — 4"*, описывающий принципы программирования CD-ROM/R/RW-накопителей, электронную версию которого можно найти, в частности, по адресу: **http://www.t10.org/ftp/t10/drafts/mmc4/mmc4r02b.pdf**).
3. **Описание ATAPI-интерфейса для CD-ROM/DVD-накопителей** (см. например, *"ATA Packet Interface for CD-ROMs"* и *"Specification for ATAPI DVD Devices"*, причем спецификации на DVD гораздо лучше и полнее описывают архитектуру CD-ROM, чем его родная документация; не самые свежие, но вполне подходящие ревизии можно найти по адресам: **www.stanford.edu/~csapuntz/specs/INF-8020.PDF** и **ftp.seagate.com/sff/INF-8090.PDF**; описания SCSI- и ATAPI-команд во многом дублируют друг друга, однако некоторые особо тонкие моменты лучше описываются то в одном, то в другом руководстве, поэтому профессиональные программисты должны иметь оба).

4. **Описание форматов хранения данных на лазерных дисках** (см. стандарт ECMA-130 *"Data interchange on read-only 120 mm optical data disks"*, известный также под именем "Желтой Книги", которую можно найти по адресу: **http://www.ecma-international.org/publications/files/ecma-st/Ecma-130.pdf**; это — базовый стандарт для накопителей CD-ROM).
5. Помимо этого годится любая литература, так или иначе затрагивающая вопросы программирования CD-ROM; нелишним будет почитать *"ATAPI(IDE) CD. Информация к размышлению"* от Константина Норватова и *"Особенности программирования CD-ROM'а на Спектруме"* от Влада Сотникова.

Итак, что же такое SCSI? Это — стандартизованный, платформенно-независимый интерфейс, обеспечивающий согласованное взаимодействие различных устройств и высокоуровневых приложений. Собственно, аббревиатура SCSI именно так и расшифровывается — *Small Computer System Interface (системный интерфейс малых компьютеров)*. Благодаря SCSI для низкоуровневого управления устройствами совершенно необязательно прибегать к написанию собственных драйверов (писать драйвер только для того, чтобы прорваться сквозь ограничения API — чистейший маразм), ведь эту задачу можно решить и на прикладном уровне, посылая устройству специальные CDB-блоки, содержащие стандартные или специфичные для данного устройства команды управления вместе со всеми необходимыми им параметрами. Собственно, CDB так и расшифровывается — *Command Descriptor Block*. Пример одного из таких блоков приведен в таблице 4.2.

Таблица 4.2. *Пример CDB-блока, который, будучи переданным SCSI-устройству, заставляет его прочитать 0x69-сектор*

Смещение, байт	Содержимое	
0x0	**0x28**	Код команды "read sector"
0x1	0x00	Зарезервировано
0x2	**0x00**	Номер сектора — 0x69
0x3	**0x00**	
0x4	**0x00**	
0x5	**0x69**	
0x6	**0x00**	Количество секторов
0x7	**0x01**	
0x8	0x00	Зарезервировано
0x9	0x00	Зарезервировано
0xA	0x00	Зарезервировано

Первый байт блока представляет собой *команду операции* (в нашем случае: 0x28 — чтение одного или нескольких секторов), а все остальные байты блока — *параметры* данной команды. Причем обратите внимание на тот факт, что младший байт слова располагается по большему адресу, — то есть все происходит не так, как в привычном нам IBM PC! Поэтому если передать в качестве номера первого сектора последовательность `0x69 0x00 0x00 0x00`, то прочитается `0x6900000` сектор, а вовсе не `0x00000069`, как можно было того ожидать!

Краткое описание стандартных SCSI-команд можно найти в том же *"The Linux SCSI programming HOWTO"*, однако для наших целей их вряд ли окажется достаточно, и команды, специфичные для дисков CD-ROM, мы рассмотрим отдельно. Однако это произойдет не раньше, чем мы разберемся, как CDB-блоки упаковываются в SRB-конверт *(SCSI Request Block)*, без которого операционная система просто не поймет, что же мы хотим сделать (как известно, машинная программа выполняет то, что ей приказали сделать, иногда это совпадает с тем, что от нее хотели, иногда нет).

Структура SRB-блока подробно описана в NT DDK, поэтому не будем подробно на ней останавливаться и пробежимся по основным полям лишь вкратце (листинг 4.6).

Листинг 4.6. Краткое описание структуры `SCSI_REQUEST_BLOCK`

```
typedef struct _SCSI_REQUEST_BLOCK {
    USHORT Length;          // длина структуры SCSI_REQUEST_BLOCK

    UCHAR Function;         // функция (обычно SRB_FUNCTION_EXECUTE_SCSI == 0, т.е.
                            // отправить устройству команду на выполнение)

    UCHAR SrbStatus;        // здесь устройство отображает прогресс выполнения
                            // команды, наиболее часто встречаются значения:
                            // SRB_STATUS_SUCCESS == 0x1 - команда завершена успешно
                            // SRB_STATUS_PENDING == 0x0 - команда еще выполняется
                            // SRB_STATUS_ERROR   == 0x4 - произошла ошибка
                            // также возможны и другие значения, перечисленные в DDK

    UCHAR ScsiStatus;       // здесь устройство возвращает статус завершения команды
                            // и, если не SUCCESS, значит, произошел ERROR

    UCHAR PathId            // SCSI-порт, на котором сидит контроллер устройства
                            // для "виртуальных" SCSI устройств всегда 0
```

```
    UCHAR TargetId;         // контроллер устройства на шине
                            // для IDE устройств обычно 0 - primary, 1 - secondary

    UCHAR Lun;              // логический номер устройства внутри контроллера
                            // для IDE устройств обычно 0 - master, 1 - slayer

    CHAR QueueTag;          // обычно не используется и должно быть равно нулю
    CHAR QueueAction;       // обычно не используется и должно быть равно нулю

    CHAR CdbLength;         // длина CDB-блока, для ATAPI-устройств всегда 12 (0Ch)
    CHAR SenseInfoBufferLength;      // длина SENSE-буфера (о нем — далее)

    LONG SrbFlags;          // флаги. обычно принимают два значения
                            // SRB_FLAGS_DATA_IN == 0x40 - перемещение данных от
                            //                   устройства к компьютеру (чтение)
                            // SRB_FLAGS_DATA_OUT == 0x80 - перемещение данных от
                            //                   компьютера к устройству (запись)

    ULONG DataTransferLength;        // длина блока читаемых/записываемых данных

    LONG TimeOutValue;      // время вылета по тайм-ауту в секундах

    PVOID DataBuffer;       // указатель на буфер с читаемыми/записываемыми данными

    PVOID SenseInfoBuffer; // указатель на SENSE буфер (о нем - далее)

    struct _SCSI_REQUEST_BLOCK *NextSrb; // указатель на след. SRB. Обычно не исп.

    PVOID OriginalRequest; // указатель на IRP. Практически не используется

    PVOID SrbExtension;     // обычно не используется и должно быть равно нулю

    UCHAR Cdb[16];          // собственно, сам CDB-блок
} SCSI_REQUEST_BLOCK, *PSCSI_REQUEST_BLOCK;
```

Заполнив поля структуры `SCSI_REQUEST_BLOCK` подобающим образом, мы можем передать SRB-блок выбранному нами устройству посредством функции `DeviceIoControl`, просто задав соответствующий код `IOCTL`. Вот, собственно, и все! Заглотнув наживку, операционная система передаст CDB-блок соответствующему устройству, и оно выполнит (или не выполнит) содержащуюся в нем (CDB-блоке) команду. Обратите внимание: *CDB-блок обраба-*

тывается не драйвером устройства, а самим устройством, и потому мы имеем практически неограниченные возможности по управлению последним. И все это — с прикладного уровня!

Теперь о грустном. Процедура управления устройствами довольно капризна, и одно-единственное неправильно заполненное поле может обернуться категорическим нежеланием устройства выполнять передаваемые ему команды. Вместо этого будет возвращаться код ошибки или вовсе не возвратится ничего. К тому же малейшая неаккуратность может запросто испортить данные на всех жестких дисках, а потому с выбором значений TargetID и lun вы должны быть особенно внимательными! *(Для автоматического определения физического адреса CD-ROM'а можно использовать SCSI-команду* `SCSI_INQUIRY` *— см. демонстрационный пример \NTDDK\src\win_me\block\wnaspi32 из DDK)*. Однако довольно говорить об опасностях (без них жизнь была бы слишком скучной), переходим к самому интересному — поиску того самого IOCTL-кода, который этот SRB-блок собственно и передает.

Оказывается, напрямую это сделать не так-то просто, точнее — легальными средствами невозможно вообще! Создатели Windows по ряду соображений решили предоставить полный доступ к полям структуры `SCSI_REQUEST_BLOCK` только писателям драйверов, а прикладных программистов оставили наедине со структурами `SCSI_PASS_THROUGH` и `SCSI_PASS_THROUGH_DIRECT`, — схожими по назначению с SRB, но несколько ограниченными в своей функциональности. К счастью, на содержимое CDB-блоков не было наложено никаких ограничений, а потому возможность низкоуровневого управления железом у нас все-таки осталась. Подробнее обо всем этом можно прочитать в разделе *"9.2 SCSI Port I/O Control Codes"* из NT DDK, а также из исходного текста демонстрационного примера \NTDDK\src\storage\class\spti из того же DDK (обратите внимание на файл spti.htm, лежащий в этом же каталоге, который достаточно подробно описывает суть управления устройством через SCSI-интерфейс).

Согласно наименованию каталога с демонстрационным примером, данный способ взаимодействия с устройством носит название *SPTI* и расшифровывается как *SCSI Pass Through IOCTLs* — т. е. *SCSI, проходящий через IOCTL*. Кратко перечислим основные особенности и ограничения SPTI-интерфейса.

- Во-первых, для передачи CDB-блоков устройству вы должны обладать привилегиями администратора, что не всегда удобно (зато безопасно!).
- Во-вторых, использование многоцелевых команд запрещено (т. е. мы не можем отдать команду копирования данных с устройства А на устройство Б в обход процессора, хотя такие команды у современных приводов есть, и было бы очень здорово копировать лазерные диски, совершенно не загружая процессор).
- В-третьих, реверсивное (то бишь двунаправленное) перемещение данных не поддерживается, и в каждый момент данные могут перемещаться либо

от устройства к компьютеру, либо от компьютера к устройству, но не то и другое одновременно!).

- В-четвертых, при установленном class-драйвере для целевого устройства, мы должны направлять CDB-блоки именно class-драйверу, а не самому SCSI-устройству. То есть для управления CD-ROM'ом вы должны взаимодействовать с ним через устройство `\\.\X:`, где `X` — буква привода, попытка же обращения к `\\.\Scsi0:` возвратит ошибку (и это, как показывает практика, основной камень преткновения неопытных программистов, начинающих программировать раньше, чем читать документацию).

Замечание

Как вариант — можно обращаться к устройству `\\.\CdRom0` или `\\.\CdRom1` без знака двоеточия на конце, где `0` и `1` — порядковый номер привода CD-ROM в системе. Вопреки распространенному заблуждению, гласящему, что устройство `\\.\CdRom0` расположено на более низком уровне, чем `\\.\X:`, с точки зрения операционной системы это синонимы, и, чтобы убедиться в этом, достаточно заглянуть в содержимое таблицы объектов (objdir "\DosDevice"), доказывающее, что `\\.\X:` представляет собой не что иное, как символическую ссылку на `\\.\CdRomN`.

- В-пятых, на максимальный размер пересылаемых данных *(MaximumTransferLength)* наложены жесткие ограничения, диктуемые спецификой используемого оборудования и обслуживающего его драйвера мини-порта. Ограничения касаются как предельно допустимого размера блока данных, так и количества занятых им физических страниц. Для определения конкретных характеристик следует послать устройству команду `IOCTL_SCSI_GET_CAPABILITIES`, которая возвратит структуру `IO_SCSI_CAPABILITIES` (ищите ее определение в NTDDSCSI.h), содержащую среди всего прочего значения `MaximumTransferLength` и MaximumPhysicalPages_in_bytes. Максимальный размер пересылаемых данных вычисляется по следующей формуле: `largest transfer = min (MaximumTransferLength, MaximumPhysicalPages_in_bytes)`. Как вариант можно ограничиться блоками по 64 Кбайт, гарантированно поддерживаемых всеми устройствами. Буфер также должен быть выровнен на величину, кратную величине `AlignmentMask`, возвращаемую в структуре `IO_SCSI_CAPABILITIES`. Степень выравнивания, обеспечиваемая функцией `malloc`, для этих целей оказывается вполне достаточной, и при ее использовании никаких проблем не возникает. Другое дело, если выделение памяти осуществляется конструкцией `char buf[BUF_SIZE]`, — в этом случае работоспособность вашей программы уже не гарантируется.

- В-шестых, сама структура `SCSI_PASS_THROUGH_DIRECT` (листинг 4.7) содержит значительно меньше полей, причем значения полей `PathId`, `TargetId` и `Lun` просто *игнорируются*! Физический адрес устройства на шине определяется непосредственно самой операционной системой по

символьному имени дескриптора устройства, которому, собственно, и посылается SCSI_PASS_THROUGH_DIRECT-запрос.

Листинг 4.7. Формат структуры SCSI_PASS_THROUGH_DIRECT (структура SCSI_PASS_THROUGH во всем похожа на нее, но не обеспечивает передачу данных через DMA)

```
typedef struct _SCSI_PASS_THROUGH_DIRECT {
    USHORT Length;                  // размер структуры SCSI_PASS_THROUGH_DIRECT
    UCHAR ScsiStatus;               // статус выполнения SCSI-команды устройством
    UCHAR PathId;                   // игнорируется
    UCHAR TargetId;                 // игнорируется
    UCHAR Lun;                      // игнорируется
    UCHAR CdbLength;                // длина CDB-пакета, посылаемая устройству, байты
    UCHAR SenseInfoLength;          // длина SENSE-буфера для возращения ошибки
    UCHAR *DataIn;                  // направление передачи данных
    ULONG DataTransferLength;       // размер буфера для обмена данными в байтах
    ULONG TimeOutValue;             // время вылета по тайм-ауту
    PVOID DataBuffer;               // указатель на буфер для обмена данными
    ULONG SenseInfoOffset;          // указатель на SENSE-буфер с информацией о error
    UCHAR Cdb[16];                  // буфер с CDB-пакетом (16 байт максимум)
}SCSI_PASS_THROUGH_DIRECT, *PSCSI_PASS_THROUGH_DIRECT;
```

К счастью, "цензура", в основном, коснулась тех полей, которые в реальной жизни все равно практически не используются, так что мы ровным счетом ничего не потеряли. Заполняем оставшиеся поля, и наша структура готова!

Естественно, прежде чем передать ее устройству, нам необходимо получить дескриптор этого самого устройства. Это можно сделать так, как показано в листинге 4.8.

Листинг 4.8. Открытие привода для получения дескриптора, использующегося для его управления

```
HANDLE hCD = CreateFile ("\\\\.\\X:", GENERIC_WRITE | GENERIC_READ,
                    FILE_SHARE_READ | FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0);
```

Убедившись, что `hCD` не равно `INVALID_HANDLE_VALUE`, передаем полученный дескриптор вместе с самой структурой `IOCTL_SCSI_PASS_THROUGHT_DIRECT` функции `DeviceIoControl`, вызывая ее, как это показано в листинге 4.9.

Листинг 4.9. Передача структуры IOCTL_SCSI_PASS_THROUGH

```
DeviceIoControl(hCD, 0x4D014h /* IOCTL_SCSI_PASS_THROUGH_DIRECT */, &srb,
            sizeof(SCSI_PASS_THROUGH_DIRECT), sense_buf, SENSE_SIZE, &returned, 0);
```

Где srb и есть заполненный экземпляр структуры IOCTRL_SCSI_PASS_THROUGHT_DIRECT, а returned — переменная, в которую будет записано количество байт, возвращенных устройством. В свою очередь, sense_buf — это тот самый буфер, в котором заполненный нами экземпляр структуры IOCTL_SCSI_PASS_THROUGHT_DIRECT возвращается назад, да не один, а вместе с Sense Info — кодом ошибки завершения операции. Если же операция завершилась без ошибок, то Sense Info не возвращается и sense_buf содержит только структуру IOCTL_SCSI_PASS_THROUGHT. Позиция размещения Sense Info в буфере определяется содержимым поля SenseInfoOffset, значение которого должно быть подобрано так, чтобы не "наступать на пятки" структуре IOCTRL_SCSI_PASS_THROUGHT, т. е., попросту говоря, минимально возможное смещение Sense Info равно: srb.SenseInfoOffset = sizeof(SCSI_PASS_THROUGH_DIRECT). Обратите внимание, SenseInfoOffset это не указатель на Sense Info, но индекс первого байта Sense Info в возвращаемом буфере!

Для определения факта наличия ошибки, необходимо проанализировать количество байт, возвращенных функцией DeviceIoControl в переменной returned. Если оно превышает размер структуры IOCTL_SCSI_PASS_THROUGHT, то в буфере находится Sense Info, а раз есть Sense Info, то есть и ошибка! Формат кода ошибки Sense Info приведен на рисунке 4.1.

Первый байт указывает на тип ошибки и обычно принимает значение 70h (*текущая ошибка* — current error) или 71h (*отсроченная ошибка* — deferred error). Коды ошибок с 72h по 7Eh зарезервированы, причем ошибки с кодом 7Eh указывают на нестандартный (vendor-specific) формат sense-info. Коды ошибок с 00h по 6Fh в спецификации CD-ROM ATAPI неопределенны, и потому их использование нежелательно (данное предостережение, разумеется, адресовано не программистам, а разработчикам аппаратуры).

Описание ошибки кодируется тройкой чисел: Sense Key, Additional Sense Code (дополнительный смысловой код, сокращенно ASC) и Additional Sense Code Qualifier (ASCQ). Вершину этой иерархической пирамиды возглавляет Sense Key, содержащий общую категорию ошибки (genetic categories), затем идет дополнительный смысловой код, более детально описывающий ошибку, и, наконец, в самом низу иерархии находится квалификатор дополнительного смыслового кода, уточняющий непосредственно сам дополнительный смысловой код. Если ошибка исчерпывающе описывается одним лишь кодом Sense Key и ASC, то ASCQ в таком случае отсутствует (точнее — находится в неопределенном состоянии).

Байт \ Бит	7	6	5	4	3	2	1	0
0	Бит Valid	Код ошибки (70h или 71h)						
1	Номер сегмента (зарезервирован)							
2	Зарезервировано		ILI	Резерв	Смысловой код			
3	Информация							
6								
7	Длина дополнительного смыслового кода (n - 7)							
8	Информация, зависящая от команды							
11								
12	Дополнительный смысловой код							
13	Квалификатор дополнительного смыслового кода (необязателен)							
14	Необязательный код полевого заменяемого устройства (Field Replaceable Unit Code)							
15	SKSV (опц.)	Информация, зависящая от смыслового кода (необязательна)						
17								
18	Дополнительные байты смыслового кода							
n								

Рис. 4.1. Формат кода `Sense Info`, возвращаемого устройством в случае возникновения ошибки

Расшифровка основных кодов ошибок описывается в двух таблицах (табл. 4.3 и 4.4). Стоит сказать, что для анализа ошибки значение Sense Key в общем-то некритично, т. к. гарантируется, что каждый код ASC принадлежит только одному Sense Key; напротив, один и тот же код ASCQ может принадлежать нескольким различным кодам ASC, и потому в отрыве от последнего он бессмыслен.

Таблица 4.3. *Основные Sense Key (категории ошибок) и их описания*

Sense Key	Описание
00h	**NO SENSE**. Нет дополнительной Sense Info. Операция выполнена успешно
01h	**RECOVERED ERROR** (восстановленная ошибка). Операция выполнена успешно, но в процессе ее выполнения возникли некоторые проблемы, устраненные непосредственно самим приводом. За дополнительной информацией обращайтесь к ключам ASC и ASCQ
02h	**NOT READY** (не готов). Устройство не готово

Таблица 4.3 *(окончание)*

Sense Key	Описание
03h	**MEDIUM ERROR** (ошибка носителя). В процессе выполнения операции произошла неустранимая ошибка, вызванная, по всей видимости, дефектами носителя или ошибкой записи данных. Данный Sense Key может возвращаться и в тех случаях, когда привод оказывается не в состоянии отличить дефект носителя от аппаратного сбоя самого привода
04h	**HARDWARE ERROR** (аппаратная ошибка). Неустранимая аппаратная ошибка (например, отказ контроллера)
05h	**ILLEGAL REQEST** (неверный запрос). Неверные параметры, переданные приводу в CDB-пакете (например, начальный адрес больше конечного)
06h	**UNIT ATTENTION** (модуль требует внимания) Носитель заменен или выполнен сброс контроллера привода
07h	**DATA PROTECT** (защищенные данные) Попытка чтения защищенных данных
8h—0Ah	Зарезервировано
0Bh	**ABORTED COMMAND** (команда прервана). По тем или иным причинам выполнение команды было прервано
0Eh	**MISCOMPARE** (ошибка сравнения) Исходные данные не соответствуют данным, прочитанным с носителя
0Fh	Зарезервировано

Таблица 4.4. *Основные ASC- и ASCQ-коды*

ASC	ASCQ	DROM	Описание
00	00	DROM	NO ADDITIONAL SENSE INFORMATION
00	11	R	PLAY OPERATION IN PROGRESS
00	12	R	PLAY OPERATION PAUSED
00	13	R	PLAY OPERATION SUCCESSFULLY COMPLETED
00	14	R	PLAY OPERATION STOPPED DUE TO ERROR
00	15	R	NO CURRENT AUDIO STATUS TO RETURN
01	00	R	MECHANICAL POSITIONING OR CHANGER ERROR
02	00	DROM	NO SEEK COMPLETE
04	00	DROM	LOGICAL DRIVE NOT READY - CAUSE NOT REPORTABLE

Таблица 4.4 *(продолжение)*

ASC	ASCQ	DROM	Описание
04	01	DROM	LOGICAL DRIVE NOT READY - IN PROGRESS OF BECOMING READY
04	02	DROM	LOGICAL DRIVE NOT READY - INITIALIZING COMMAND REQUIRED
04	03	DROM	LOGICAL DRIVE NOT READY - MANUAL INTERVENTION REQUIRED
05	01	DROM	MEDIA LOAD - EJECT FAILED
06	00	DROM	NO REFERENCE POSITION FOUND
09	00	DRO	TRACK FOLLOWING ERROR
09	01	RO	TRACKING SERVO FAILURE
09	02	RO	FOCUS SERVO FAILURE
09	03	RO	SPINDLE SERVO FAILURE
11	00	DRO	UNRECOVERED READ ERROR
11	06	RO	CIRC UNRECOVERED ERROR
15	00	DROM	RANDOM POSITIONING ERROR
15	01	DROM	MECHANICAL POSITIONING OR CHANGER ERROR
15	02	DRO	POSITIONING ERROR DETECTED BY READ OF MEDIUM
17	00	DRO	RECOVERED DATA WITH NO ERROR CORRECTION APPLIED
17	01	DRO	RECOVERED DATA WITH RETRIES
17	02	DRO	RECOVERED DATA WITH POSITIVE HEAD OFFSET
17	03	DRO	RECOVERED DATA WITH NEGATIVE HEAD OFFSET
17	04	RO	RECOVERED DATA WITH RETRIES AND/OR CIRC APPLIED
17	05	DRO	RECOVERED DATA USING PREVIOUS SECTOR ID
18	00	DRO	RECOVERED DATA WITH ERROR CORRECTION APPLIED
18	01	DRO	RECOVERED DATA WITH ERROR CORRECTION & RETRIES APPLIED
18	02	DRO	RECOVERED DATA - THE DATA WAS AUTO-REALLOCATED

Таблица 4.4 *(продолжение)*

ASC	ASCQ	DROM	Описание
18	03	R	RECOVERED DATA WITH CIRC
18	04	R	RECOVERED DATA WITH L-EC
1A	00	DROM	PARAMETER LIST LENGTH ERROR
20	00	DROM	INVALID COMMAND OPERATION CODE
21	00	DROM	LOGICAL BLOCK ADDRESS OUT OF RANGE
24	00	DROM	INVALID FIELD IN COMMAND PACKET
26	00	DROM	INVALID FIELD IN PARAMETER LIST
26	01	DROM	PARAMETER NOT SUPPORTED
26	02	DROM	PARAMETER VALUE INVALID
28	00	ROM	NOT READY TO READY TRANSITION, MEDIUM MAY HAVE CHANGED
29	00	ROM	POWER ON, RESET OR BUS DEVICE RESET OCCURRED
2A	00	ROM	PARAMETERS CHANGED
2A	01	ROM	MODE PARAMETERS CHANGED
30	00	ROM	INCOMPATIBLE MEDIUM INSTALLED
30	01	RO	CANNOT READ MEDIUM - UNKNOWN FORMAT
30	02	RO	CANNOT READ MEDIUM - INCOMPATIBLE FORMAT
39	00	ROM	SAVING PARAMETERS NOT SUPPORTED
3A	00	ROM	MEDIUM NOT PRESENT
3F	00	ROM	ATAPI CD-ROM DRIVE OPERATING CONDITIONS HAVE CHANGED
3F	01	ROM	MICROCODE HAS BEEN CHANGED
40	NN	ROM	DIAGNOSTIC FAILURE ON COMPONENT NN (80H-FFH)
44	00	ROM	INTERNAL ATAPI CD-ROM DRIVE FAILURE
4E	00	ROM	OVERLAPPED COMMANDS ATTEMPTED
53	00	ROM	MEDIA LOAD OR EJECT FAILED
53	02	ROM	MEDIUM REMOVAL PREVENTED
57	00	R	UNABLE TO RECOVER TABLE OF CONTENTS
5A	00	DROM	OPERATOR REQUEST OR STATE CHANGE INPUT (UNSPECIFIED)

Таблица 4.4 *(окончание)*

ASC	ASCQ	DROM	Описание
5A	01	DROM	OPERATOR MEDIUM REMOVAL REQUEST
63	00	R	END OF USER AREA ENCOUNTERED ON THIS TRACK
64	00	R	ILLEGAL MODE FOR THIS TRACK
B9	00	R	PLAY OPERATION OBORTED
BF	00	R	LOSS OF STREAMING

Как видите, — все просто! Единственное, с чем мы еще не разобрались, — это ATAPI. Поскольку мы не собираемся взаимодействовать с ATAPI-интерфейсом напрямую (этой возможности "благодаря" архитекторам Windows мы, увы, лишены), "промчимся галопом" лишь по ключевым аспектам и особенностям. Как пишет Михаил Гук в своей книге *"Интерфейсы персональных компьютеров"*: *"Для устройств, логически отличающихся от жестких дисков — оптических, магнитооптических, ленточных и любых других, — в 1996 г. была принята спецификация ATAPI. Это пакетное расширение интерфейса, которое позволяет передавать по шине ATA-устройству блоки командной информации, структура которых была позаимствована из SCSI"*. Теперь, по крайней мере, становится понятно, почему Windows так лихо "превращает" ATAPI-устройства в SCSI. Если отбросить аппаратные различия интерфейсов, которые с программного уровня все равно не видны, то ATAPI-интерфейс будет очень напоминать SCSI. Во всяком случае, управление ATAPI-устройствами осуществляется посредством тех самых CDB-блоков, которые мы уже рассмотрели ранее.

Естественно, чтобы управлять устройством, необходимо знать, какими именно командами оно управляется. Для получения этой информации нам понадобится документ *"ATAPI Packet Commands for CD-ROM devices"*. Откройте его на описании команды READ CD command (код `BEh`), и вы обнаружите таблицу формата этой команды (рис. 4.2).

Попробуем в ней разобраться. Первый байт (точнее байт 0), представляющий собой код выполняемой команды, никаких вопросов не вызывает, но вот дальше мы сталкиваемся с полем Expected Sector Type, задающим тип требуемого сектора. Перевернув несколько страниц вперед, мы найдем коды, соответствующие всем существующим типам секторов: CDDA, Mode 1, Mode 2, Mode 2 Form 1 и Mode 2 Form 2. Если же тип сектора заранее неизвестен, передавайте с этим полем 0x0, что обозначает "нас устроит любой тип сектора".

Следующие четыре байта занимает *адрес первого читаемого сектора* (Starting Logical Block Address), заданный в формате LBA *(Logical Block Address)*.

За этой "страшной" аббревиатурой скрывается элегантный способ сквозной нумерации секторов. Если вы когда-то программировали "древние" жесткие диски, то наверняка помните, какие громоздкие расчеты приходилось выполнять, чтобы определить, к какой головке, цилиндру, сектору каждый байт прилежит. Теперь же можно обойтись безо всех этих заморочек. Первый сектор имеет номер 0, затем идет 1, 2, 3 и так до последнего сектора диска. Только помните, что порядок байт в этом двойном слове обратный, — т. е. старший байт старшего слова идет первым.

Байты с шестого по восьмой "оккупировал" параметр, задающий *количество читаемых секторов* (Transfer Length in Blocks). Вот какая несправедливость — для адреса сектора выделяется четыре байта, а для количества читаемых секторов только три. Шутка! Вы же ведь не собираетесь читать весь диск за раз?! Порядок байт здесь также обратный, так что не ошибитесь, иначе при попытке считать один-единственный сектор вы запросите добрую половину диска целиком!

Девятый байт наиболее интересен, ибо он хранит *флаги, определяющие, какие части сектора мы хотим прочитать* (Flag Bits). Помимо пользовательских данных, мы можем запросить *синхробайты* (Synch Field), *заголовок* (Header(s) code), *EDC/ECC-коды* (EDC & ECC) и даже *флаги ошибок чтения* (Error Flag(s)) (для взлома некоторых защит это самое то! — правда, эту возможность поддерживают не все приводы).

Байт \ Бит	7	6	5	4	3	2	1	0
0	Код операции (BEh)							
1	Зарезервировано			Ожидаемый тип сектора			Зарезервировано	
2	MSB							
3	Начальный адрес логического блока (LBA)							
4								
5								LSB
6	MSB							
7	Длина передаваемой информации в блоках							
8								LSB
9	Флаги							
	Поле Synch	Код заголовка(ов)		Данные	EDC& ECC	Флаг(и) ошибки		Резерв
10	Зарезервировано					Биты выбора данных подканала		
11	Зарезервировано							

Рис. 4.2. Формат команды `READ CD`

Десятый байт (Sub-Channel Data Selection Bits) отвечает за извлечение данных из подканалов, однако поскольку эти же самые данные уже содержатся в заголовке, то без них можно, в принципе, и обойтись.

Наконец, последний, одиннадцатый, считая от нуля, байт никак не используется и зарезервирован на будущее, а потому для гарантии совместимости с новыми моделями приводов, он должен быть равен нулю.

Естественно, в зависимости от рода и количества запрашиваемых данных, длина возвращенного сектора может варьироваться в очень широких пределах (табл. 4.5).

Таблица 4.5. *Взаимосвязь рода запрошенных данных и длины возвращаемого сектора, стандартные режимы выделены серым*

Data to be transferred	Flag Bits	CD-DA	Mode 1	Mode 2 non XA	Mode 2 Form 1	Mode 2 Form 2
User Data	10h	2352	2048	2336	2048	2338
User Data + EDC/ECC	18h	(10h)	2336	(10h)	2336	(10h)
Header Only	20h	(10h)	4	4	4	4
Header Only + EDC/ECC	28h	(10h)	Illegal	Illegal	Illegal	Illegal
Header & User Data	30h	(10h)	2052	2340	Illegal	Illegal
Header & User Data + EDC/ECC	38h	(10h)	2344	(30h)	Illegal	Illegal
Sub Header Only	40h	(10h)	8	8	8	8
Sub Header On-ly + EDC/ECC	48h	(10h)	Illegal	Illegal	Illegal	Illegal
Sub Header & User Data	50h	(10h)	(10h)	(10h)	2056	2336
Sub Header & User Data + EDC/ECC	58h	(10h)	(10h)	(10h)	2344	(50h)
All Header Only	60h	(10h)	12	12	12	12
All Header Only + EDC/ECC	68h	(10h)	Illegal	Illegal	Illegal	Illegal
All Header & User Data	70h	(10h)	(30h)	(30h)	2060	2340
All Header & User Data + EDC/ECC	78h	(10h)	(30h)	(30h)	2340	2340
Sync & User Data	90h	(10h)	Illegal	Illegal	Illegal	Illegal
Sync & User Data + EDC/ECC	98h	(10h)	Illegal	Illegal	Illegal	Illegal

Таблица 4.5 *(окончание)*

Data to be transferred	Flag Bits	CD-DA	Mode 1	Mode 2 non XA	Mode 2 Form 1	Mode 2 Form 2
Sync & Header Only	A0h	(10h)	16	16	16	16
Sync & Header Only + EDC/ECC	A8h	(10h)	Illegal	Illegal	Illegal	Illegal
Sync & Header & User Data	B0h	(10h)	2064	2352	Illegal	Illegal
Sync & Header & User Data + EDC/ECC	B8h	(10h)	2344	(30h)	Illegal	Illegal
Sync & Sub Header Only	C0h	(10h)	Illegal	Illegal	Illegal	Illegal
Sync & Sub Header Only + EDC/ECC	C8h	(10h)	Illegal	Illegal	Illegal	Illegal
Sync & Sub Header & User Data	D0h	(10h)	(10h)	(10h)	Illegal	Illegal
Sync & Sub Header & User Data + EDC/ECC	D8h	(10h)	(10h)	(10h)	Illegal	Illegal
Sync & All Headers Only	E0h	(10h)	24	24	24	24
Sync & All Headers Only + EDC/ECC	E8h	(10h)	Illegal	Illegal	Illegal	Illegal
Sync & All Headers & User Data	F0h	(10h)	2064	2352	2072	2352
Sync & All Headers & User Data + EDC/ECC	F8h	(10h)	2352	(F0h)	2352	(F0h)
Repeat All Above and Add Error Flags	02h	294	294	294	294	294
Repeat All Above and Add Block & Error Flags	04h	296	296	296	296	296

IDE-устройства с прикладного уровня видятся как SCSI. Разумеется, на физическом уровне с приводом не происходит никаких изменений, и привод CD-ROM с IDE-интерфейсом так IDE-приводом и остается, со всеми присущими ему достоинствами и недостатками. Однако IRP-запросы *(I/O Request Packet)* к этому драйверу, проходя через Storage class driver, транслируются в блок SRB *(SCSI Request Block)*. Затем SRB-запросы попадают в Storage port driver (т. е. непосредственно в сам драйвер привода), где они заново транслируются в конкретные физические команды данного устройства (рис. 4.3). Подробности этого увлекательного процессора можно почерпнуть из набора NT DDK (*"4.1 Storage Driver Architecture"*), здесь же

достаточно указать на тот немаловажный факт, что, кроме команд семейства `IRP_MJ_xxx`, мы также можем посылать устройству и SRB-запросы, которые обладают значительно большей свободой и гибкостью. Однако такое взаимодействие невозможно осуществить непосредственно с прикладного уровня, поскольку IRP-команды относятся к числу *приватных* команд, в то время как API-функция `DeviceIoControl` передает лишь публичные команды, явно обрабатываемые драйвером в диспетчере `IRP_MJ_DEVICE_CONTROL`.

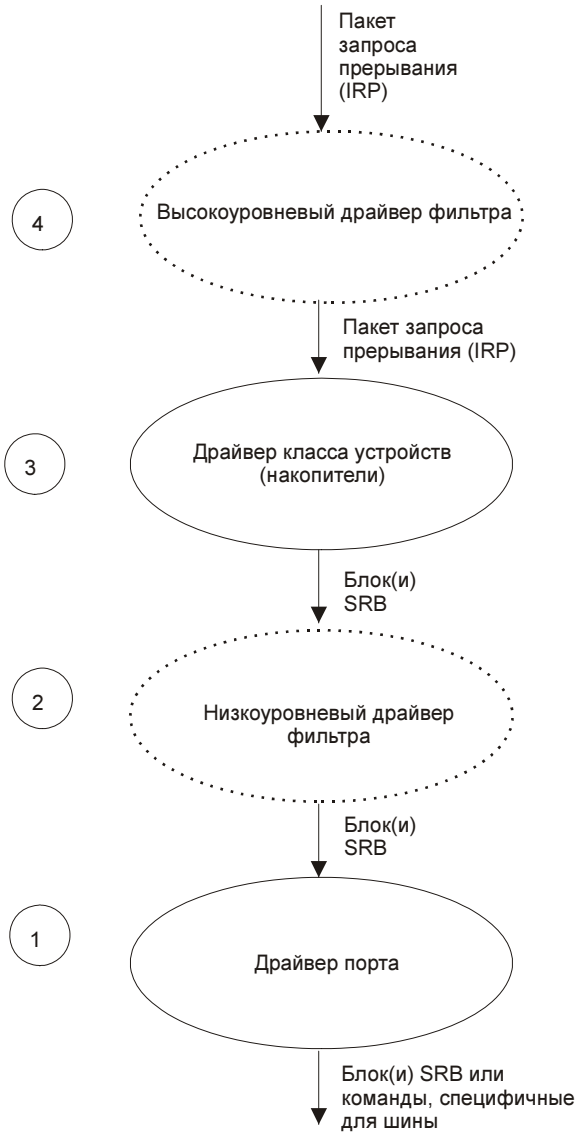


Рис. 4.3. Внутренний мир Windows NT

Давайте теперь, в порядке закрепления всего вышесказанного, попытаемся создать программу, которая бы читала сектора с лазерных дисков в "сыром" виде. Ее ключевой фрагмент (вместе со всеми необходимыми комментариями) приведен в листинге 4.10.

Листинг 4.10. Функция, читающая сектора в "сыром" виде посредством SPTI

```
#define RAW_READ_CMD            0xBE // ATAPI RAW READ
#define WHATS_READ              0xF8 // Sync & All Headers & User Data + EDC/ECC
#define PACKET_LEN              2352 // длина одного сектора
//#define WHATS_READ            0x10 // User Data
//#define PACKET_LEN            2048 // длина одного сектора

//-[SPTI_RAW_SECTOR_READ]-------------------------------------------------------
//      функция читает один или несколько секторов с CDROM в сыром (RAW) виде,
//      согласно переданным флагам
//
//      ARG:
//           CD            - что открывать (типа "\\\\.\\X:" или "\\\\.\\CdRom0")
//           buf           - буфер куда читать
//           buf_len       - размер буфера в байтах
//           StartSec      - с какого сектора читать, считая от нуля
//           N_SECTOR      - сколько секторов читать
//           flags         - что читать (см. спецификацию на SCSI/ATAPI)
//
//      RET:
//           !=0           - функция завершилась успешно
//           ==0           - функция завершилась с ошибкой
//
//      NOTE:
//           - работает только под NT/W2K/XP и требует прав администратора
//
//           - 64 Кб данных за раз максимум
//-------------------------------------------------------------------------------
SPTI_RAW_SECTOR_READ(char *CD,char *buf,int buf_len,int StartSec,int N_SEC,char flags)
{
      HANDLE                        hCD;
      SCSI_PASS_THROUGH_DIRECT      srb;
      DWORD                         returned, length, status;
```

```
// ОТКРЫВАЕМ УСТРОЙСТВО
hCD = CreateFile (      driver, GENERIC_WRITE|GENERIC_READ,

FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,0,0);
if (hCD == INVALID_HANDLE_VALUE) { printf("-ERR: open CD\n"); return 0;}

// ФОРМИРУЕМ SRB
memset(&srb,0,sizeof(SCSI_PASS_THROUGH_DIRECT));      // инициализация

srb.Length              = sizeof(SCSI_PASS_THROUGH_DIRECT);
srb.PathId              = 0;                    // SCSI controller ID
srb.TargetId            = 6;                    // игнорируется
srb.Lun                 = 9;                    // игнорируется
srb.CdbLength           = 12;                   // длина CDB-пакета
srb.SenseInfoLength     = 0;                    // нам не нужна SenseInfo
srb.DataIn              = SCSI_IOCTL_DATA_IN;   // мы будем читать
srb.DataTransferLength  = PACKET_LEN*N_SECTOR;  // сколько мы будем читать
srb.TimeOutValue        = 200;                  // время выхода по TimeOut
srb.DataBufferOffset    = buf;                  // указатель на буфер
srb.SenseInfoOffset     = 0;                    // SenseInfo не нужна

// CDB-пакет, содержащий команды ATAPI
srb.Cdb[0]              = RAW_READ_CMD;         // читать сырой сектор
srb.Cdb[1]              = 0x0;                  // формат диска - любой

// номер первого сектора для чтения, причем сначала передается старший
// байт старшего слова, а потом младший байт младшего слова
srb.Cdb[2]              = HIBYTE(HIWORD(StartSector));
srb.Cdb[3]              = LOBYTE(HIWORD(StartSector));
srb.Cdb[4]              = HIBYTE(LOWORD(StartSector));
srb.Cdb[5]              = LOBYTE(LOWORD(StartSector));

// количество секторов для чтения
srb.Cdb[6]              = LOBYTE(HIWORD(N_SECTOR));
srb.Cdb[7]              = HIBYTE(LOWORD(N_SECTOR));
srb.Cdb[8]              = LOBYTE(LOWORD(N_SECTOR));

srb.Cdb[9]              = flags;                // что читать
srb.Cdb[10]             = 0;                    // Sub-Channel Data Bits
srb.Cdb[11]             = 0;                    // reserverd
```

```
    // ОТПРАВЛЯЕМ SRB-блок ATAPI-устройству
    status = DeviceIoControl(hCD, IOCTL_SCSI_PASS_THROUGH_DIRECT,
              &srb, sizeof(SCSI_PASS_THROUGH_DIRECT), &srb, 0, &returned, 0);

    return 1;
}
```

Остается отметить, что защитные механизмы, взаимодействующие с диском посредством интерфейса SPTI, элементарно ломаются установкой точки останова на функции `CreateFile`/`DeviceIoControl`. Для предотвращения "лишних" всплытий отладчика фильтр точки останова должен реагировать только на те вызовы функции `CreateFile`, чей первый слева аргумент равен `\\.\X:` или `\\.\CdRomN`. Автоматически, второй слева аргумент функции `DeviceIoControl` должен представлять собой либо структуру `IOCTL_SCSI_PASS_THROUGHT`, либо `IOCTL_SCSI_PASS_THROUGHT_DIRECT`, шестнадцатеричные значения кодов которых равны `0x4D004` и `0x4D014` соответственно.

Доступ через ASPI

> Отлаженная программа — это такая программа, для которой еще не найдены условия, в которых она откажет.
>
> *Программистский фольклор*

Вот два основных недостатка интерфейса SPTI (только что описанного ранее): для взаимодействия с устройством он требует наличия прав администратора, и, что еще хуже, SPTI поддерживается только операционными системами семейства NT и отсутствует в Windows 9x/ME. Единственный легальный способ дотянуться до CD-ROM'a под Windows 9x — это воспользоваться 16-разрядным шлюзом, напрямую обращаясь к MS-DOS-драйверу MSCDEX, который обеспечивает значительно бо́льшую функциональность, нежели Windows-драйвер. Естественно, параллельная поддержка двух семейств операционных систем требует от программиста значительных усилий, что существенно повышает себестоимость программного продукта.

Для упрощения разработки кросс-платформенных приложений фирма Adaptec разработала специальный системно-независимый интерфейс, позволяющий управлять различными SCSI-устройствами с прикладного уровня, и назвала его ASPI — *Advanced SCSI Programming Interface* (хотя неофициально его расшифровывают как Adaptec SCSI Programming Interface, поскольку это больше соответствует истине).

Системонезависимость интерфейса ASPI обеспечивается двухуровневой моделью его организации: архитектурно он состоит из *низкоуровневого драйвера*

и прикладной *библиотеки-обертки*. ASPI-драйвер разрабатывается с учетом специфики конкретной операционной системы и отвечает за непосредственное управление SCSI-шиной (реальной или виртуальной — не суть важно). Поскольку интерфейс между операционной системой и драйвером меняется от одной операционной системы к другой, то для сокрытия всех этих различий используется специальная ASPI-библиотека, предоставляющая единый унифицированный интерфейс для всех операционных систем.

Рассмотрим, как осуществляется внедрение ASPI-интерфейса в операционную систему на примере Windows Me (см. рис. 4.2). На самом высоком уровне иерархии находятся прикладные библиотеки Wnaspi32.dll и Winaspi.dll для 32- и 16-разрядных приложений соответственно. Они экспортируют три базовых ASPI-функции: `GetASPI32DLLVersion`, `GetASPI32SupportInfo` и `SendASPI32Command` (причем последняя — самая важная) и три вспомогательных: `GetASPI32Buffer`, `FreeASPI32Buffer`, `TranslateASPI32Address` (последняя — только в 32-разрядной версии библиотеки).

Посредством функции `DeviceIoControl` они взаимодействуют с ASPI-драйвером, расположенным "ниже" и, в зависимости от версии операционной системы, называющимся либо APIX.VXD (Windows 9x), либо ASPI.SYS (Windows NT) и создающим в процессе своей инициализации устройство с непроизносимым названием *MbMmDp32*. Только не спрашивайте меня, как это абракадабра расшифровывается — ответ похоронен в застенках компании Adaptec.

Замечание

В 16-разрядных приложениях взаимодействие с драйвером осуществляется через функцию 1868h прерывания 2Fh. Подробности этого процесса можно узнать, дизассемблируя Winaspi.dll. Она, кстати, совсем крошечная — всего 5 Кбайт.

В принципе, ничего не мешает взаимодействовать с ASPI-драйвером и напрямую — в обход библиотеки Wnaspi32.dll. Собственно, многие разработчики защитных механизмов именно так и поступают. Достаточно лишь дизассемблировать Wnaspi32.dll и разобраться, каким ASPI-командам какие IOCTL-коды соответствуют (ASPI-протокол по понятным соображениям не документирован). Действительно, на функции `SendASPI32Command` очень легко поставить точку останова (breakpoint), и тогда хакер мгновенно локализует защитный код. С вызовами же функции `DeviceIoControl` в силу их многочисленности взломщикам справиться намного труднее. К тому же начинающие ломатели защит (а таких среди хакеров — большинство) весьма смутно представляют себе архитектуру ввода-вывода и уж тем более не разбираются в ASPI-протоколе. Впрочем, для опытных хакеров такая защита — не преграда (подробнее *см. главу 5*).

Сам же ASPI-драйвер "подключен" к SCSI- и IDE/ATAPI-портам (рис. 4.4), за счет чего он позволяет управлять всеми этими устройствами (и приводами CD-ROM в том числе).

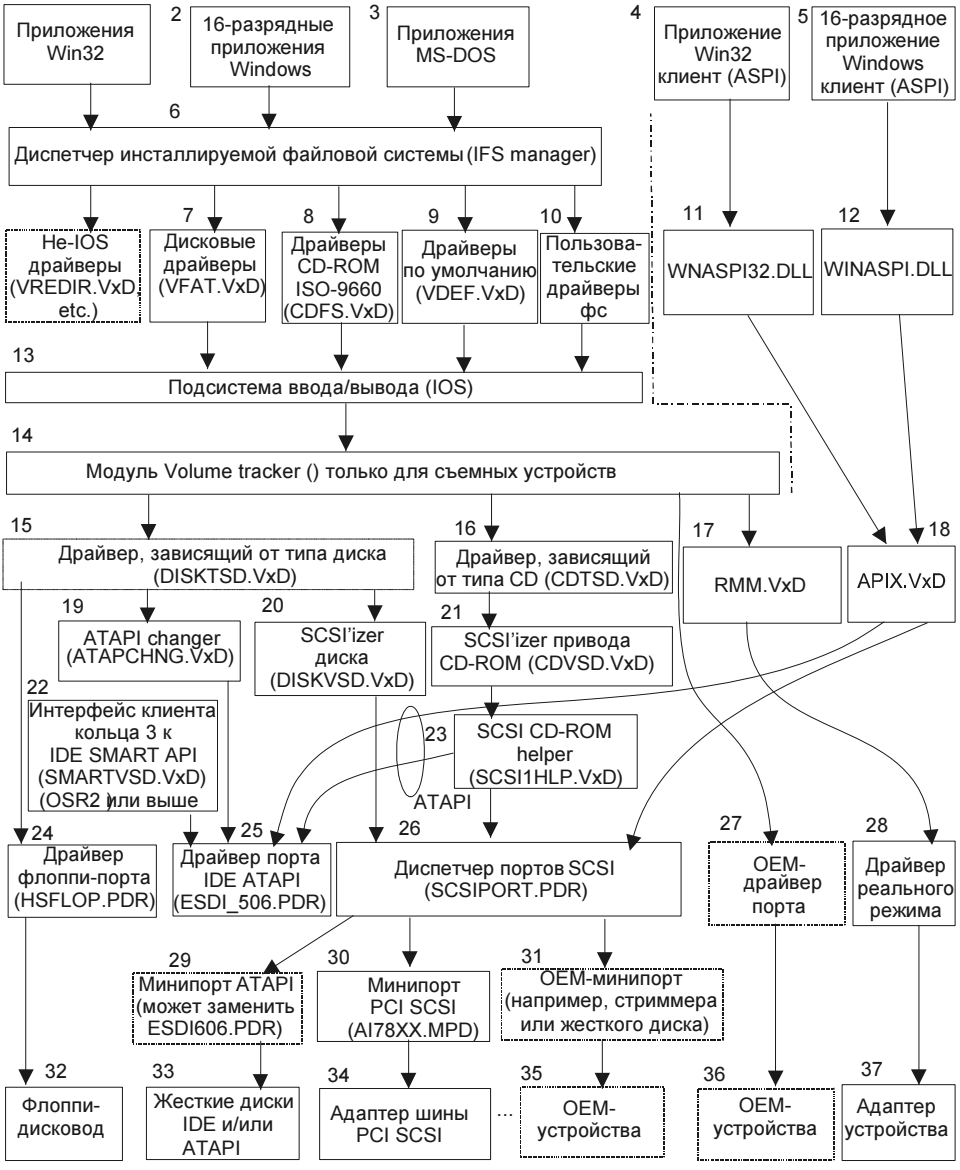


Рис. 4.4. Архитектура подсистемы ввода-вывода Windows 98

Клиентские модули (на данной схеме они обозначены цифрами 1, 2 и 3) посылают свои запросы драйверу файловой системы — Instable File System (обозначенному цифрой 6). В распоряжении клиентских модулей также имеются ASPI-библиотеки для 32- и 16-разрядных приложений соответственно (они обозначены цифрами 4 и 6). От всей системы они стоят особняком, поскольку разработаны независимой компанией Adaptec и представляют собой факультативные компоненты. Драйвер файловой системы перенаправляет полученный им запрос на один их следующих специализированных драйверов, среди которых присутствует и драйвер привода CD-ROM, — CDFS.VxD, обозначенный цифрой 8. В его задачи входит поддержка файловых систем лазерных дисков, таких как: ISO-9660, High Sierra или других файловых систем. Уровнем ниже лежит Volume Tracker (цифра 14), отслеживающий смену диска в накопителе, а еще ниже находится непосредственно сам драйвер, поддерживающий данную модель CD-ROM, — так называемый CD type specific driver, реализуемый драйвером CDVSD.VxD и среди прочих обязанностей отвечающий за назначение буквы приводу. Это и есть секторный уровень взаимодействия с диском, никаких файловых систем здесь нет и в помине. Несмотря на то, что данный драйвер специфичен для конкретной модели привода CD-ROM, он совершенно независим от его физического интерфейса, поскольку опирается на CD-ROM device SCSI'zer (цифра 21), преобразующий IOP-запросы, поступающие от вышележащих драйверов, в SRB-пакеты, направляемые нижележащим драйверам (подробнее об этом *см. раздел "Доступ через SCSI-порта" данной главы*). Еще ниже находится SCSI CD-ROM helper (цифра 23), обеспечивающий стыковку SCSI'zer'a с SCSI-портом. Сам же SCSI-port, создаваемый менеджером SCSI-портов (цифра 26) представляет собой унифицированное системно-независимое средство взаимодействия драйверов среднего уровня с физическим (или виртуальным) оборудованием. К одному из таких SCSI-портов и подключается ASPI-драйвер (цифра 18), реализованный в файле APIX.VxD и восходящий к своим "оберткам" — Wnaspi32.dll и Wnaspi.dll (цифры 11 и 12 соответственно). Ниже SCSI-менеджера расположены драйвера мини-портов, переводящие SCSI-запросы в язык конкретной интерфейсной шины. В частности, драйвер, обеспечивающий поддержку IDE-устройств, реализован в файле ESDI_506.PDR (цифра 29). Естественно, при желании мы можем общаться с IDE-устройствами и через IDE/ATAPI-порты (цифра 25), реализованные все тем же драйвером ESDI_506.PDR (ASPI-драйвер по соображениям производительности именно так, собственно, и поступает). Левую часть блок-схемы, изображающую иерархию драйверов прочих дисковых устройств, мы не рассматриваем, т. к. она не имеет никакого отношения к теме нашего обсуждения.

Для программирования под ASPI требуются как минимум две вещи: ASPI-*драйвер* и ASPI-*SDK*. Драйвер можно бесплатно скачать с сервера самой Adaptec (ею разработаны драйвера для следующих операционных систем: MS-DOS, Novell, Windows 9x, Windows NT/W2K/XP), а вот SDK (Software

Development Kit) с некоторого момента распространяется за деньги. И хотя его стоимость чисто символическая (что-то около 10$, если мне не изменяет память), неразвитость платежных систем в России превращает процесс покупки в довольно затруднительное дело. Однако все необходимое для работы (документация, заголовочные файлы, библиотеки) можно позаимствовать из Windows Me DDK (кстати, входящего в состав DDK для Windows 2000). Так что, если у вас уже есть Windows 2000 DDK, вам не о чем беспокоиться. В противном случае попробуйте обратиться к MSDN, распространяемой вместе с Microsoft Visual Studio 6.0. Здесь вы найдете документацию и заголовочные файлы, ну а недостающие библиотеки из соответствующих DLL можно получить и самостоятельно (lib.exe с ключом /DEF) либо же вовсе обойтись без них, загружая все необходимые функции через LoadLibrary/GetProcAddress.

Поскольку ASPI-интерфейс хорошо документирован (руководство по программированию насчитывает около 35 листов), то его освоение не должно вызвать никаких непреодолимых проблем (во всяком случае, после знакомства с SPTI). К тому же, в Windows Me DDK входит один законченный демонстрационный пример использования ASPI, найти который можно в папке \src\win_me\block\wnaspi32\. Несмотря на досадный суффикс "Me", он отлично уживается и с другими операционными системами, как-то: Windows 98, Windows 2000, Windows XP и т. д.

Впрочем, реализован этот пример на редкость "криво" и с большим количеством ошибок, а его наглядность такова, что менее наглядного примера для демонстрации ASPI, пожалуй, и не подобрать! Уж лучше исследовать исходные тексты программы CD slow, которые можно легко найти в Интернете (однако она написана на ассемблере, а с ассемблером знаком не всякий).

Кратко перечислим основные недочеты демонстрационного примера aspi32ln.c:

- Во-первых, это не консольная программа, а относящаяся к GUI *(Graphical User Interface)* — графическому интерфейсу пользователя, а потому бо́льшая часть ее кода к ASPI вообще никакого отношения не имеет.
- Во-вторых, используется единая функция для получения уведомлений сразу от выполнения двух команд: `SCSI_INQUIRY` и `SCSI_READ10`, причем последняя в половине случаев заменена своей константой 0x28, что тоже не способствует ее пониманию.
- В-третьих, накопители на CD-ROM программой поддерживаются лишь частично. Плохо спроектированная архитектура программы не позволила разработчикам осилить поставленную перед ними задачу. Поэтому ветка, отвечающая за чтение с CD-ROM в функции `ASPI32Post`, специальным образом закомментирована. Если же наложенную блокировку убрать, то при чтении будет происходить ошибка, поскольку программа ориентирована лишь на те накопители, чей размер сектора составляет 0x200 байт.

Приводы CD-ROM дисков, чей сектор вчетверо больше, очевидно, к этой категории не относятся, и, чтобы не переписывать всю программу целиком, единственное, что можно сделать, — это увеличить размер запрашиваемого блока данных до 0x800 байт (с жестких дисков будет считываться по четыре сектора за раз, что вполне допустимо).

- В-четвертых, инкремент (т. е. вычисление адреса следующего считываемого блока) реализован неверно и поэтому вообще не работоспособен.

Ладно, не будет увлекаться критикой сопроводительных примеров (даже плохой программный код все же лучше, чем совсем ничего) и перейдем непосредственно к изучению ASPI-интерфейса, а точнее — его важнейшей команды `SendASPI32Command`, обеспечивающей передачу SRB-блоков устройству (со всеми остальными командами вы без труда справитесь и самостоятельно).

Структура `SRB_ExecSCSICmd`, в которую, собственно, и упаковывается SRB-запрос, как две капли воды похожа на структуру `SCSI_PASS_THROUGH_DIRECT`. Во всяком случае, между ними больше сходства, чем различий (листинг 4.11).

Листинг 4.11. Структура `SRB_ExecSCSICmd`

```
typedef struct
{
      BYTE SRB_Cmd;                    // ASPI command code = SC_EXEC_SCSI_CMD
      BYTE SRB_Status                  // ASPI command status byte
      BYTE SRB_HaId;                   // ASPI host adapter number
      BYTE SRB_Flags;                  // ASPI request flags
      DWORD SRB_Hdr_Rsvd;              // Reserved, MUST = 0
      BYTE SRB_Target;                 // Target's SCSI ID
      BYTE SRB_Lun;                    // Target's LUN number
      WORD SRB_Rsvd1;                  // Reserved for Alignment
      DWORD SRB_BufLen;                // Data Allocation Length
      LPBYTE SRB_BufPointer;           // Data Buffer Pointer
      BYTE SRB_SenseLen;               // Sense Allocation Length
      BYTE SRB_CDBLen;                 // CDB Length
      BYTE SRB_HaStat;                 // Host Adapter Status
      BYTE SRB_TargStat;               // Target Status
      LPVOID SRB_PostProc;             // Post routine
      BYTE SRB_Rsvd2[20];              // Reserved, MUST = 0
      BYTE CDBByte[16];                // SCSI CDB
      BYTE SenseArea[SENSE_LEN+2];     // Request Sense buffer
}
SRB_ExecSCSICmd, *PSRB_ExecSCSICmd;
```

Обратите внимание: для взаимодействия с устройством вам совершенно незачем знать его дескриптор! Достаточно указать его физический адрес на шине (т. е. правильно заполнить поля `SRB_HaId` и `SRB_Target`). А как их узнать? Да очень просто: достаточно разослать по всем физическим адресам команду `INQUIRY` (код `12h`). Устройства, реально (и/или виртуально) подключенные к данному порту, вернут идентификационную информацию (среди прочих полезных данных содержащую и свое имя), а несуществующие устройства не вернут ничего, и операционная система "отрапортует" об ошибке.

Простейшая программа опроса устройств может выглядеть, например, так, как показано в листинге 4.12.

Листинг 4.12. Последовательный опрос портов на предмет наличия подключенных к ним устройств

```
#define MAX_ID              8
#define MAX_INFO_LEN        48
SEND_SCSI_INQUITY()
{
      #define MAX_LUN       8      // макс. возможное кол-во логических устройств
      BYTE      AdapterCount;
      DWORD     ASPI32Status;
      unsigned char buf[0xFF];
      unsigned char str[0xFF];
      unsigned char CDB[ATAPI_CDB_SIZE];
      long a, real_len, adapterid, targetid;

      // получаем кол-во адаптеров на шине
      ASPI32Status = GetASPI32SupportInfo();
      AdapterCount = (LOBYTE(LOWORD(ASPI32Status)));

      // готовим CDB-блок
      memset(CDB, 0, ATAPI_CDB_SIZE);
      CDB[0] = 0x12;               // INQUIRY
      CDB[4] = 0xFF;               // размер ответа

      // спамим порты в надежде найти тех, кто нам нужен
      for (adapterid = 0; adapterid < MAX_LUN; adapterid++)
      { // внимание! нельзя здесь ^^^^^^^^^^^^^ использовать AdapterCount,
        // как это рекомендуется в некоторых руководствах, поскольку номера
        // адаптеров устройств далеко не всегда идут вплотную друг к другу,
        // и если в нумерации возникает "разрыв", одно или более устройств
```

```
        // останутся необнаруженными

            for (targetid = 0; targetid < MAX_ID; targetid++)
            {
                  a = SEND_ASPI_CMD(adapterid, targetid, CDB,
                            ATAPI_CDB_SIZE, 0, buf, 0xFF, ASPI_DATA_IN);
                  if (a == SS_COMP)
                  {
                        real_len=(buf[4]>MAX_INFO_LEN)?buf[4]:MAX_INFO_LEN;
                        memcpy(str,&buf[8],real_len);str[real_len]=0;
                        printf("%d.%d <-- %s\n",adapterid, targetid, str);
                  }
            }
      }
}
```

Результат работы программы на компьютере автора выглядит так, как показано в листинге 4.13 (обратите внимание, что адреса устройств, подключенных к виртуальной SCSI-шине, созданной драйвером ASPI, могут и не соответствовать их физическим адресам; в данном случае, привод PHILIPS, висящий на физическом IDE-порту с номером 0, попал на виртуальный порт с номером 1, поскольку нулевой порт занят драйвером Virtual CloneCD, при удалении последнего из системы, соответствие виртуальных и физических адресов полностью восстанавливается, однако ручаться за это нельзя). Первая слева цифра в листинге — adapter ID, следующая за ней — target ID.

Листинг 4.13. Устройства, подключенные к компьютеру автора

```
0.0 <-- ELBY    DVD-ROM         1.0
1.0 <-- IBM-DTLA-307015         TX2O
1.1 <-- PHILIPS CDRW2412A       P1.55VO1214DM10574
2.0 <-- ST380011A               3.06
2.1 <-- TEAC    CD-W552E        1.09
3.0 <-- AXV     CD/DVD-ROM      2.2a
3.1 <-- AXV     CD/DVD-ROM      2.2a
3.2 <-- AXV     CD/DVD-ROM      2.2a
```

Другое немаловажное достоинство ASPI-интерфейса по сравнению с SPTI состоит в поддержке *асинхронного режима* обработки запросов. Отдав запрос на чтение такого-то количество секторов, вы можете продолжить выполнение своей программы, не дожидаясь, пока процесс чтения секторов полностью не завершится. Конечно, для достижения аналогичного результата при использовании интерфейса SPTI достаточно всего лишь создать еще один

поток, но это уже не так элегантно и красиво. Демонстрационный пример программы, осуществляющей "сырое" чтение сектора с CD-диска, показан в листинге 4.14.

Листинг 4.14. Пример программы, осуществляющей "сырое" чтение сектора с CD-диска

```
#include "scsidefs.h"
#include "wnaspi32.h"

void ASPI32Post (LPVOID);

#define F_NAME                  "raw.sector.dat"

/* ASPI SRB packet length */
#define ASPI_SRB_LEN      0x100

#define RAW_READ_CM             0xBE

#define WHATS_READ              0xF8      // Sync & All Headers & User Data
                                          // + EDC/ECC
#define PACKET_LEN              2352

//#define WHATS_READ      0x10            // User Data
//#define PACKET_LEN      2048


#define MY_CMD                  RAW_READ_CMD

HANDLE hEvent;

//-[DWORD READ_RAW_SECTOR_FROM_CD]-------------------------------------------
//            функция читает один или несколько секторов с CD-ROM в "сыром"
//            (RAW) виде, согласно переданным флагам
//
//      ARG:
//      adapter_id      -       номер шины (0 - primary, 1 - secondary)
//      read_id         -       номер устройства на шине (0 - master, 1 - slaeyer)
//      buf             -       буфер, куда читать
//      buf_len         -       размер буфера в байтах
//      StartSector     -       с какого сектора читать, считая от нуля
```

```
//      N_SECTOR        -       сколько секторов читать \
//      flags           -       что читать (см. спецификацию на ATAPI)
//
//      RET:
//                      -       ничего не возвращает
//
//      NOTE:
//            - функция возвращает управления до завершения выполнения запроса,
//              поэтому на момент выхода из нее, содержимое буфера с данными еще
//              пусто, и реально он заполняется только при вызове функции
//              ASPI32Post (вы можете модифицировать ее по своему усмотрению)
//              для сигнализации о завершении операции рекомендуется использовать
//              события (Event)
//
//            - функция работает и под 9x/ME/NT/W2K/XP и _не_ требует для себя
//              прав администратора. Однако ASPI-драйвер должен быть установлен.
//-----------------------------------------------------------------------------
READ_RAW_SECTOR_FROM_CD(int adapter_id,int read_id,char *buf,int buf_len,
                        int StartSector,int N_SECTOR,int flags)
{
      PSRB_ExecSCSICmd SRB;
      DWORD      ASPI32Status;

      // выделяем память для SRB-запроса
      SRB = malloc(ASPI_SRB_LEN); memset(SRB, 0, ASPI_SRB_LEN);

      // ПОДГОТОВКА SRB-блока
      SRB->SRB_Cmd          = SC_EXEC_SCSI_CMD;          // выполнить SCSI-команду
      SRB->SRB_HaId         = adapter_id;                // ID адаптера
      SRB->SRB_Flags        = SRB_DIR_IN|SRB_POSTING;    // асинхр. чтение данных
      SRB->SRB_Target       = read_id;                   // ID-устройства
      SRB->SRB_BufPointer   = buf;                       // сюда читаются данные
      SRB->SRB_BufLen       = buf_len;                   // длина буфера
      SRB->SRB_SenseLen     = SENSE_LEN;                 // длина SENSE-буфера
      SRB->SRB_CDBLen       = 12;                        // размер ATAPI-пакета

      SRB->CDBByte [0]      = MY_CMD;                    // ATAI-команда
      SRB->CDBByte [1]      = 0x0;                       // формат CD - любой

                            // номер первого сектора
      SRB->CDBByte [2]      = HIBYTE(HIWORD(StartSector));
```

```
    SRB->CDBByte [3]      = LOBYTE(HIWORD(StartSector));
    SRB->CDBByte [4]      = HIBYTE(LOWORD(StartSector));
    SRB->CDBByte [5]      = LOBYTE(LOWORD(StartSector));

                          // кол-во читаемых секторов
    SRB->CDBByte [6]      = LOBYTE(HIWORD(N_SECTOR));
    SRB->CDBByte [7]      = HIBYTE(LOWORD(N_SECTOR));
    SRB->CDBByte [8]      = LOBYTE(LOWORD(N_SECTOR));

    SRB->CDBByte [9]      = flags                // что читать?
    SRB->CDBByte [10]     = 0;                   // данные подканала не нужны
    SRB->CDBByte [11]     = 0;                   // reserverd

    // адрес процедуры, которая будет получать уведомления
    SRB->SRB_PostProc     = (void *) ASPI32Post;

    // посылаем SRB-запрос устройству
    SendASPI32Command(SRB);

    // возвращаемся из функции _до_ завершения выполнения запроса
    return 0;
}

//-----------------------------------------------------------------------------
//      эта callback-функция вызывается самим ASPI и получает управление при
//      завершении выполнения запроса или же при возникновении ошибки.
//      в качестве параметра она получает указатель на экземпляр структуры
//      PSRB_ExecSCSICmd, содержащей всю необходимую информацию (статус, указатель
//      на буфер и т.д.)
//-----------------------------------------------------------------------------
void ASPI32Post (void *Srb)
{
    FILE *f;

    // наш запрос выполнен успешно?
    if ((((PSRB_ExecSCSICmd) Srb)->SRB_Status) == SS_COMP)
    {
        // ЭТОТ КОД ВЫ МОЖЕТЕ МОДИФИЦИРОВАТЬ ПО СВОЕМУ УСМОТРЕНИЮ
        //-------------------------------------------------------
        // записывает содержимое сектора в файл
        // внимание! PSRB_ExecSCSICmd) Srb)->SRB_BufLen содержит не актуальную
```

```
            // длину прочитанных данных, а размер самого буфера. если количество
            // байт, возвращенных устройством, окажется меньше размеров буфера, то
            // его хвост будет содержать мусор! здесь мы используем поле SRB_BufLen
            // только потому, что при вызове функции SendASPI32Command тщательно
            // следим за соответствием размера буфера количеству возвращаемой нам
            // информации
            if (f=fopen(F_NAME, "w"))
            {
                  // записывает сектор в файл
                  fwrite(((PSRB_ExecSCSICmd) Srb)->SRB_BufPointer,1,
                  ((PSRB_ExecSCSICmd) Srb)->SRB_BufLen, f);
                  fclose(f);
            }
            // кукарекаем и "размораживаем" поток, давая понять, что процедура
            // чтения закончилась
            MessageBeep(0); SetEvent(hEvent);
            //-------------------------------------------------------
      }
}

main(int argc, char **argv)
{
      void *p; int buf_len, TIME_OUT = 4000;

      if (argc<5)
      {
            fprintf(stderr,"USAGE:\n\tRAW.CD.READ.EXE adapter_id"\
            ", read_id, StartSector, n_sec\n"); return 0;
      }

      // вычисляем длину буфера и выделяем для него память
      // ВНИМАНИЕ: таким образом можно юзать только до 64КБ
      // если же вам требуются буфера больших объемов,
      // используйте функцию GetASPI32Buffer
      buf_len = PACKET_LEN*atol(argv[4]); p = malloc(buf_len);

      // создаем событие
      if ((hEvent = CreateEvent(NULL,FALSE,FALSE,NULL)) == NULL) return -1;

      // читаем один или несколько секторов с CD
      READ_RAW_SECTOR_FROM_CD(atol(argv[1]), atol(argv[2]),p,buf_len,
```

```
                    atol(argv[3]), atol(argv[4]),WHATS_READ);

    // ждем завершения выполнения операции
    WaitForSingleObject(hEvent, TIME_OUT);

    return 0;
}
```

Откомпилировав этот пример и запустив его на выполнение, убедитесь, что он успешно работает как под Windows 9x, так и под Windows NT, причем не требуя у вас наличия прав администратора! С одной стороны, это, бесспорно, хорошо, но с другой наличие ASPI-драйвера создает огромную дыру в системе безопасности, позволяя зловредным программам вытворять с вашим оборудованием все что угодно. Заразить *MBR (Master Boot Record)* boot-сектора? Пожалуйста! Уничтожить информацию со всего диска целиком — да проще этого ничего нет! Поэтому, если вы заботитесь о собственной безопасности — удалите ASPI32-драйвер со своего компьютера (для этого достаточно удалить файл ASPI.SYS из каталога WINNT\System32\Drivers). Разумеется, сказанное относится только к NT, поскольку в операционных системах Windows 9x прямой доступ к оборудованию можно заполучить и без этого.

Доступ через SCSI-порт

Как уже говорилось ранее (см. *разд. "Доступ через SPTI" этой главы*), независимо от физического интерфейса дискового накопителя (SCSI или IDE) мы можем взаимодействовать с ним через унифицированный SCSI-интерфейс. Другими словами, драйвер конкретного устройства (и привода CD-ROM, в частности) полностью абстрагирован от особенностей реализации шинного интерфейса данного устройства. Даже если завтра появятся накопители, работающие через инфракрасный порт, драйвер CDROM.SYS ничего об этом не "узнает" и будет по-прежнему управлять ими через SCSI-порт.

Даже если на вашем компьютере не установлено ни одного SCSI-контролера, пара-тройка вполне работоспособных SCSI-портов у вас обязательно есть. Конечно, это виртуальные, а не физические порты, но с точки зрения программного обеспечения они выглядят точь-в-точь как настоящие. Попробуйте с помощью функции `CreateFile` отрыть устройство `\\.\SCSI0:`, и оно успешно откроется, подтверждая наличие существования виртуальных SCSI-портов (только не забудьте про двоеточие на конце). Посылая определенные IOCTL-команды SCSI-порту, мы можем управлять подключенным к этому порту физическим или виртуальным устройством. Да! Между SCSI-портом (виртуальным) и интерфейсной шиной (физической)

расположен еще один уровень абстракции, занимаемый *SCSI-мини-портом*, который, собственно, и "отвязывает" драйвер SCSI-порта от конкретного физического оборудования (подробнее см. *разд. "Доступ через SCSI мини-порт" этой главы*).

Естественно, прежде чем посылать IOCTL-команды в SCSI-порт, неплохо бы узнать, какое именно оборудование к этому порту подключено. Существует множество способов решения этой проблемы: от *послать устройству команду идентификации* `IOCTL_SCSI_GET_INQUIRY_DATA` (см. исходный текст демонстрационного примера в NT DDK "NTDDK\src\storage\class\spti"), и тогда устройство среди прочей информации сообщит нам, как его зовут (например, PHILIPS CDRW2412A), до *заглянуть в таблицу объектов*, чем мы сейчас и займемся. В состав NT DDK входит утилита objdir.exe, которая, как и следует из ее названия, позволяет отображать содержимое дерева объектов в виде директории. Устройства, доступные для открытия функцией `CreateFile`, хранятся в каталоге с довольно нелепым именем \DosDevices\, глядя на которое, можно подумать, что оно содержит имена устройств, видимых из-под MS-DOS, которую Windows NT вынуждена эмулировать для сохранения обратной совместимости. На самом же деле этот каталог активно используется Win32-подсистемой Windows NT, и всякий раз, когда функция `CreateFile` обращается к тому или иному логическому устройству (например, пытается открыть файл C:\MYDIR\myfile.txt), подсистема Win32 обращается к каталогу \DosDevices\, чтобы выяснить, с каким именно внутренним устройством это логическое устройство связано. Внутренние устройства видны лишь из-под Native-NT, а для всех ее подсистем они лишены всякого смысла. В частности, диск C: под Native-NT именуется как \Device\HarddiskVolume1, а полный путь к файлу myfile.txt выглядит так: \Device\HarddiskVolume1\MYDIR\myfile.txt. Только не пытайтесь "скормить" эту строчку функции `CreateFile` — она скорее "поперхнется", чем "поймет", что же от нее хотят.

Таким образом, каталог \DosDevices\ служит своеобразным связующим звеном между подсистемой Win32 и ядром системы Windows NT. Вот и давайте, в плане "возвращения к нашим баранам", посмотрим, с каким native-устройством ассоциировано логическое устройство SCSI. Запустив objdir с ключом \Dos\Devices и не забыв перенаправить весь вывод в файл (objdir \DosDevices | MORE — как альтернативный результат), мы среди моря прочей информации обнаружим следующие строки (при отсутствии DDK можно воспользоваться отладчиком Soft-Ice, в котором для достижения аналогичного результата следует набрать команду `objdir \??` — именно так! Здесь имеется два знака вопроса, поскольку директория \DosDevices на самом деле никакая не директория, а символическая ссылка на директорию `\??` или, если так угодно, ее ярлык). В листинге 4.15 показан пример взаимосвязи SCSI-устройств с native-устройствами.

Листинг 4.15. Взаимосвязь логических SCSI-устройств с native-NT-устройствами

```
Scsi0:          SymbolicLink - \Device\Ide\IdePort0
Scsi1:          SymbolicLink - \Device\Ide\IdePort1
Scsi2:          SymbolicLink - \Device\Scsi\axsaki1
```

Оказывается, устройства `Scsi0:` и `Scsi1:` представляют собой не что иное, как символические ссылки на IDE-порты с номерами 0- и 1- соответственно. Впрочем, устройства с именами `IdePort0` и `IdePort1` не являются IDE-портами в физическом смысле этого слова. Это виртуальные SCSI-порты, создаваемые драйвером ATAPI.SYS в процессе его инициализации. Он же создает символические связи \DosDevices\SCSI0: и \DosDevices\SCSI1: к ним, а также ярлыки \Device\ScsiPort0 и \Device\ScsiPort1, недоступные подсистеме Win32, но предназначенные для внутреннего использования исключительно на уровне драйверов. Разумеется, драйвер ATAPI.SYS не только создает все ранее перечисленные устройства, но и обслуживает их, предоставляя драйверам более высоких уровней унифицированный интерфейс для взаимодействия с установленным оборудованием.

А вот устройство с именем `Scsi2:` ни с какими физическими шинами вообще не связно, и к соответствующему ему SCSI-порту подключен виртуальный привод CD-ROM, создаваемый программой Alcohol 120%, а точнее ее драйвером — AXSAKI.SYS! Драйвера высокого уровня (в частности, драйвер CDROM.SYS), не заподозрив никакого подвоха, будут работать с виртуальным диском точно так же, как и с настоящим, что, собственно, и не удивительно, т. к. концепция SCSI-порта обеспечивает независимость драйверов верхнего уровня от особенностей оборудования, с которым они, с позволения сказать, "работают". Именно поэтому под Windows NT так легко реализуются эмуляторы физических устройств!

Управлять SCSI-устройствами можно и с прикладного уровня через STPI-интерфейс, однако вместо буквенного имени привода следует задавать имя SCSI-порта, к которому этот привод подключен. Основное достоинство такого способа управления заключается в том, что для взаимодействия с приводом совершенно необязательно обладать правами администратора! Привилегий простого смертного пользователя будет более чем достаточно. К тому же, прямая работа со SCSI-портом несколько производительнее взаимодействия с устройством через длинную цепочку драйверов верхнего уровня многочисленных фильтров, окружающих их.

Однако все попытки передачи SRB-блока через SCSI-порт заканчиваются неизменной ошибкой. Следующий код (листинг 4.16) наотрез отказывается работать. Почему?

Листинг 4.16. Пример неправильной работы с виртуальным SCSI-портом

```
// получаем дескриптор SCSI-порта
hCD = CreateFile ("\\\\.\\SCSI1", GENERIC_WRITE|GENERIC_READ,
                FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,0,0);

// ФОРМИРУЕМ SRB-блок
…

// ОТПРАВЛЯЕМ SRB-блок непосредственно на SCSI-порт
status = DeviceIoControl(hCD, IOCTL_SCSI_PASS_THROUGH_DIRECT, &srb,
                      sizeof(SCSI_PASS_THROUGH), &srb, 0, &returned, FALSE);
```

Зарубежные телеконференции буквально кишат вопросами на этот счет, — у одних этот код исправно работает, а у других — нет (и их большинство). А ответ, между тем, находится в DDK (если, конечно, читать его сверху вниз, а не наискосок по диагонали). Вот, пожалуйста, цитата из раздела **9.2 SCSI Port I/O Control Codes**: *"If a class driver for the target type of device exists, the request must be sent to that class driver. Thus, an application can send this request directly to the system port driver for a target logical unit only if there is no class driver for the type of device connected to that LU"*[1] *("Если класс-драйвер для целевого устройства установлен, управляющие запросы должны посылаться класс-драйверу, но не самому порту устройства. Таким образом, приложения могут посылать непосредственные запросы драйверу системного порта для целевых логических устройств, только если класс-драйвер для соответствующего типа устройств, подключенных к данному LU, не установлен"*). В переводе на нетехнический язык, непосредственное управление портом с прикладного уровня возможно для тех *и только тех* устройств, чей класс-драйвер не установлен. Скажем, если вы подключили к компьютеру какую-то нестандартную железяку, то управлять ей напрямую через SCSI-порт вполне возможно (ведь класс-драйвера для нее нет!). Но приводы CD-ROM, про которые мы, собственно, и говорим, — совсем иное дело! Класс-драйвер для них всегда установлен, и потому операционная система всячески препятствует прямому взаимодействую с оборудованием через SCSI-порт, поскольку это единственный надежный путь избежать конфликтов.

Выходит, доступ к приводам через SCSI-порт невозможен? И так, и не так! Прямой доступ к SCSI-порту действительно блокируется системой, но та же самая система предоставляет возможность управления устройством через SCSI-мини-порт. Мини-порт? Что это такое?! А вот в этом мы сейчас и разберемся!

---

[1] См. также техническую заметку Q137247 из MSDN "**IOCTL_SCSI_MINIPORT and IOCTL_SCSI_PASS_THROUGH Limitations**"

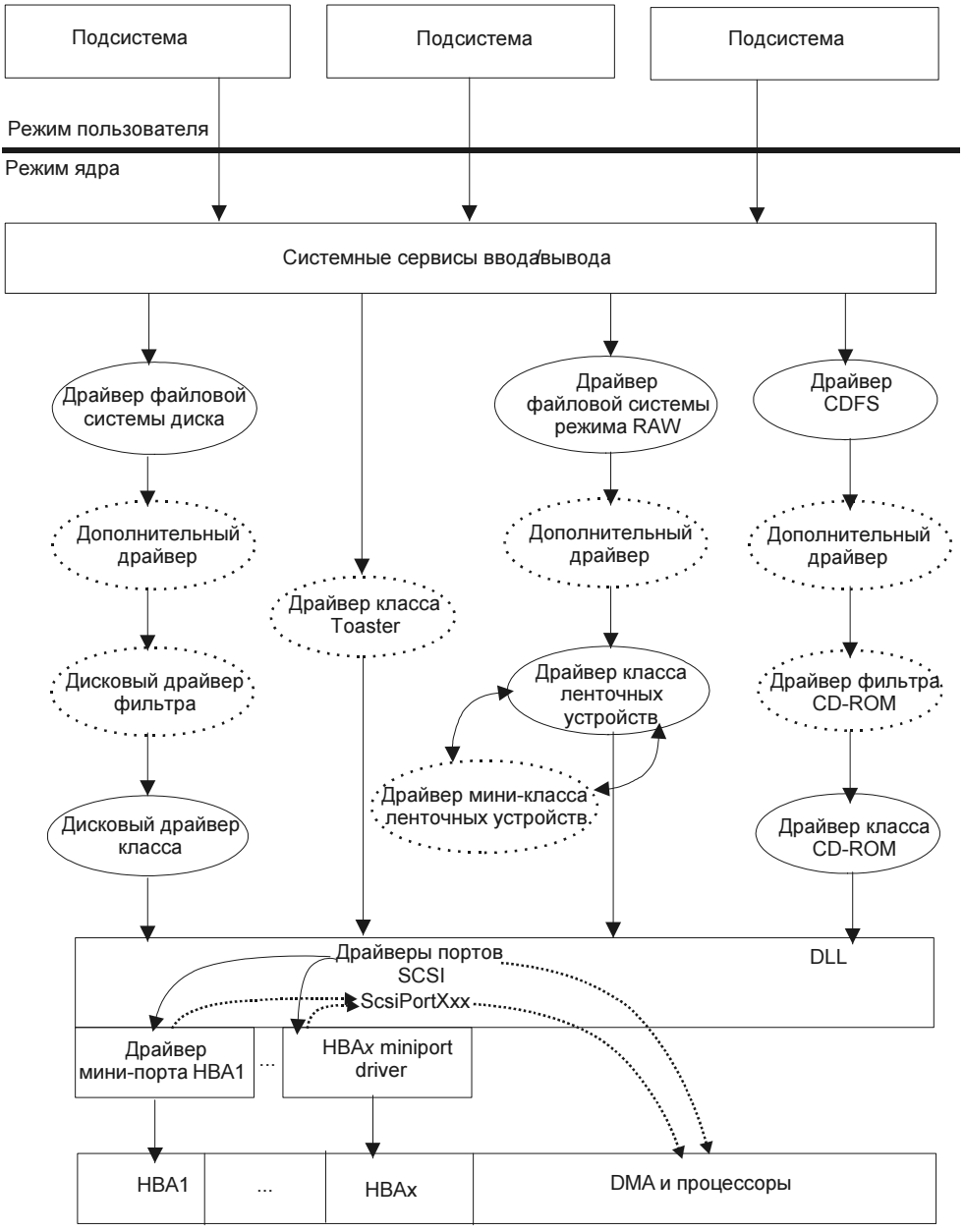


Рис. 4.5. Архитектура подсистемы ввода/вывода в Windows NT

Доступ через SCSI-мини-порт

Драйвер SCSI-мини-порта и есть тот самый драйвер, за счет которого системе удается абстрагироваться от особенностей физических интерфейсов конкретного оборудования. Условимся для краткости называть его просто "мини-драйвером", хотя это будет и не совсем верно, поскольку, помимо SCSI-мини-портов, существуют драйвера для видео- и сетевых мини-портов. Однако поскольку ни те, ни другие к рассматриваемому нами контексту никоим боком не относятся, то никаких разночтений и не возникает.

Иерархически драйвер мини-порта располагается между физическими (виртуальными) устройствами, подключенными к тем или иным интерфейсным шинам компьютера (IDE/PCI/SCSI) и драйвером SCSI-порта. Драйвер мини-порта представляет собой системно-независимый драйвер, но в то же время зависимый от специфики конкретных *HBA (Host Bus Adapter)*, то есть того самого физического/виртуального оборудования, которое он обслуживает. Драйвер мини-порта экспортирует ряд функций семейства `ScsiPortXXX`, предназначенных для использования драйверами верхних уровней, и обычно реализуется как динамическая библиотека (то есть DLL), естественно, исполняющийся в нулевом кольце "ядерного" уровня.

Именно он транслирует SCSI-запросы в команды подключенного к нему устройства, именно он создает виртуальные SCSI-порты с именами типа \Device\ScsiPort$_x$, именно он обеспечивает поддержку накопителей с физическими интерфейсами, отличными от SCSI-интерфейса. ATAPI.SYS, обслуживающий приводы CD-ROM с ATAPI-интерфейсом, DISK.SYS, обслуживающий жесткие диски, — все они реализованы как драйверы мини-порта.

Управление мини-портом осуществляется посредством специального IOCTL-кода, передаваемого функции `DeviceIoControl` и определенного в файле NTDDSCSI.H как IOCTL_SCSI_MINIPORT. Если же у вас нет NT DKK, то вот его непосредственное значение: 0x4D008. Естественно, прежде чем вызывать `DeviceIoControl`, соответствующий SCSI-порт должен быть заблаговременно открыт функцией `CreateFile`. Это может выглядеть, например, так, как это показано в листинге 4.17. Причем обратите внимание на то, что имя порта должно выглядеть как `SCSIx:`, а не как `ScsiPortx`; причем в его конце обязательно должен присутствовать символ двоеточия, иначе ничего не получится.

Листинг 4.17. Открытие SCSI-порта для управления драйвером мини-порта

```
h = CreateFile("\\\\.\\SCSI1:", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
            FILE_SHARE_WRITE, NULL,OPEN_EXISTING, 0, NULL);
```

Здесь мы открываем первый, считая от нуля, SCSI-порт, который, как мы уже знаем, соответствует первому каналу IDE или, другими словами, Secondary IDE-контроллеру (на компьютере автора привод CD-ROM "висит" именно на нем). Для определения расположения приводов на неизвестном нам компьютере можно воспользоваться IOCTL-кодом IOCTL_SCSI_GET_INQUIRY_DATA, который заставит драйвер мини-порта перечислить все имеющееся в его наличии оборудование, после чего нам останется только определить его тип (подробнее см. "NTDDK\SRC\STORAGE\CLASS\SPTI").

Однако управление мини-портом осуществляется *совсем не так*, как SCSI-портом! На этом уровне никаких стандартных команд уже не существует, и мы вынуждены работать с учетом специфики и особенностей реализации конкретного оборудования. Вместо SRB-запросов, мини-драйверу передается структура `SRB_IO_CONTROL`, обеспечивающая управление драйвером мини-порта и определенная следующим образом (листинг 4.18).

Листинг 4.18. Назначение полей структуры `SRB_IO_CONTROL`

```
typedef struct _SRB_IO_CONTROL
{
      ULONG HeaderLength;          // sizeof(SRB_IO_CONTROL)
      UCHAR Signature[8];          // сигнатура мини-драйвера
      ULONG Timeout;               // макс. время ожидания выполнения запроса в сек
      ULONG ControlCode;           // код команды
      ULONG ReturnCode;            // здесь нам вернут статус завершения
      ULONG Length;                // длина всего передаваемого буфера целиком
} SRB_IO_CONTROL, *PSRB_IO_CONTROL;
```

Ну, с полем HeaderLength все более или менее ясно, но вот что это за *сигнатура* такая?! Дело в том, что коды управления драйверами мини-порта не стандартизованы и определяются непосредственно самим разработчиком данного драйвера, а потому коды команд одного драйвера вряд ли подойдут к другому. Вот во избежание междоусобных конфликтов каждый драйвер мини-порта и содержит уникальную сигнатуру, которую тщательно сверяет с сигнатурой переданной приложением в поле `Signature` структуры `SRB_IO_CONTROL`. И, если эти сигнатуры не совпадают, драйвер "отвечает": `SRB_STATUS_INVALID_REQUEST` (типа, отвали, моя черешня). К сожалению, интерфейс штатных мини-драйверов ATAPI.SYS и DISK.SYS абсолютно незадокументирован, и если вы не умеете дизассемблировать, то вам остается лишь посочувствовать. Дизассемблер же сразу показывает, что сигнатуры обоих драйверов выглядят как `SCSIDISK`, а сигнатура мини-драйвера от Alcohol 120% — `Alcoholx` (впрочем, последний, в силу своей нештатности, не представляет для нас особенного интереса).

С кодами команды разобраться сложнее. Правда, специалисты, постоянно читающие MSDN и потому неплохо в нем ориентирующиеся, вероятно, смогут вспомнить, что: "this specification describes the API for an application to issue SMART commands to an IDE drive under Microsoft Windows 95 and Windows NT. Under Windows 95, the API is implemented in a Vendor Specific Driver (VSD), Smartvsd.vxd. SMART functionality is implemented as a "pass through" mechanism whereby the application sets up the IDE registers in a structure and passes it to the driver through the DeviceIoControl API" (*"эта спецификация описывает API для приложений, передающих SMART-команды жестким дискам с IDE-интерфейсом под Microsoft Windows 95 и Windows NT. Под Windows 95 API реализован в драйвере, специфичном для конкретного производителя (VSD — Vendor Specific Driver), называемом Smartvsd.vxd. SMART-функциональность реализована как "pass through"-механизм, посредством которого приложения устанавливают IDE-регистры, передавая их драйверу через специальную структуру, помещаемую во входной буфер функции DeviceIoControl"*).

Ага! Один из драйверов позволяет нам манипулировать регистрами IDE-контроллера по своему усмотрению, то есть фактически предоставляет низкоуровневый доступ к диску! Очень хорошо! Интерфейс со SMART-драйвером достаточно хорошо документирован (см. "MSDN → Specifications → Platforms → SMART IOCTL API Specification"), правда, раздражает гробовое молчание насчет Windows NT. То, что в NT никаких VxD нет — это и ежу ясно. Но в то же время заявляется, что SMART API в ней как будто бы реализована. Если напрячь свои мозги и проявить чудеса интуиции, можно догадаться, что поддержка SMART в NT обеспечивается штатными средствами! Весь вопрос в том, какими именно средствами и как? Ни SDK, ни DDK не содержат никакой информации на этот счет, но вот копание в заголовочных файлах из комплекта NT DDK может кое-что дать! Посмотрите, что обнаруживается в файле scsi.h при тщательном его изучении (листинг 4.19). Здесь представлены команды управления технологией SMART в Windows NT, которые мы можем передавать драйверу мини-порта через поле `ControlCode` структуры `SRB_IO_CONTROL`.

Листинг 4.19. Команды управления SMART в Windows NT

```
//
// SMART support in atapi
//
#define IOCTL_SCSI_MINIPORT_SMART_VERSION          ((FILE_DEVICE_SCSI<<16)+0x0500)
#define IOCTL_SCSI_MINIPORT_IDENTIFY               ((FILE_DEVICE_SCSI<<16)+0x0501)
#define IOCTL_SCSI_MINIPORT_READ_SMART_ATTRIBS     ((FILE_DEVICE_SCSI<<16)+0x0502)
#define IOCTL_SCSI_MINIPORT_READ_SMART_THRESHOLDS  ((FILE_DEVICE_SCSI<<16)+0x0503)
#define IOCTL_SCSI_MINIPORT_ENABLE_SMART           ((FILE_DEVICE_SCSI<<16)+0x0504)
```

```
#define IOCTL_SCSI_MINIPORT_DISABLE_SMART           ((FILE_DEVICE_SCSI<<16)+0x0505)
#define IOCTL_SCSI_MINIPORT_RETURN_STATUS           ((FILE_DEVICE_SCSI<<16)+0x0506)
#define IOCTL_SCSI_MINIPORT_ENABLE_DISABLE_AUTOSAVE ((FILE_DEVICE_SCSI<<16)+0x0507)
#define IOCTL_SCSI_MINIPORT_SAVE_ATTRIBUTE_VALUES   ((FILE_DEVICE_SCSI<<16)+0x0508)
#define IOCTL_SCSI_MINIPORT_EXECUTE_OFFLINE_DIAGS   ((FILE_DEVICE_SCSI<<16)+0x0509)
#define IOCTL_SCSI_MINIPORT_ENABLE_DISABLE_AUTO_OFFLINE   (FILE_DEVICE_SCSI<<16)+0x050a
```

"Оторви Тиггеру хвост", если в Windows NT функциональность SMART реализуется не в драйвере мини-порта! И дизассемблирование ATAPI.SYS действительно подтверждает это! Вот вам и качество документации от Microsoft, — уродство сплошное в стиле маразм крепчает. Какой смысл включать в заголовочный файл IOCTL-команды, но не документировать их?! Причем согласно лицензии дизассемблирование любых компонентов операционной системы запрещено. Ладно, не будем скулить по поводу и без, а лучше еще раз перечитаем "SMART IOCTL API Specification", откуда поймем, что для управления драйвером мини-порта под Windows NT в поле `ControlCode` структуры `SRB_IO_CONTROL` мы должны передать код одной из приведенных выше команд. Пусть это будет, например, `IOCTL_SCSI_MINIPORT_IDENTIFY`.

Сразу же за концом структуры `SRB_IO_CONTROL` должна быть расположена структура `SENDCMDINPARAMS`, дающая прямой доступ к IDE-регистрам и определенная, как это показано в листинге 4.20.

Листинг 4.20. Структура `SENDCMDINPARAMS`

```
typedef struct _SENDCMDINPARAMS
{
     DWORD        cBufferSize;     // размер буфера в байтах или нуль
     IDEREGS      irDriveRegs;     // структура, содержащая значение IDE-регистров
     BYTE         bDriveNumber;    // физический номер диска, считая от нуля
     BYTE         bReserved[3];    // зарезервировано
     DWORD        dwReserved[4];   // зарезервировано
     BYTE         bBuffer[1];      // отсюда начинается входной буфер
} SENDCMDINPARAMS, *PSENDCMDINPARAMS, *LPSENDCMDINPARAMS;
```

То есть входной буфер функции `DeviceIoControl` должен выглядеть, как на рис. 4.6.

SRB_IO_CONTROL	SENDCMDINPARAMS	DBuffer (если есть)

Рис. 4.6. Структура входного буфера функции `DeviceIoControl` для управления драйвером мини-порта под Windows 9x/NT

Первый элемент структуры — `cBufferSize`, содержащий размер bBuffer'a слишком очевиден и не интересен. А вот структура `IDREGS` обеспечивает низкоуровневый доступ к IDE-регистрам и представляет собой настоящий клад. Вот взгляните сами (листинг 4.21), только не упадите со стула, ибо потрясение будет столь же острым, сколь и глубоким.

Листинг 4.21. Структура `IDEREGS`

```
typedef struct _IDEREGS
{
      BYTE bFeaturesReg;       // IDE Features-регистр
      BYTE bSectorCountReg;    // IDE SectorCount-регистр
      BYTE bSectorNumberReg;   // IDE SectorNumber-регистр
      BYTE bCylLowReg;         // IDE CylLowReg-регистр
      BYTE bCylHighReg;        // IDE CylHighReg-регистр
      BYTE bDriveHeadReg;      // IDE DriveHead-регистр
      BYTE bCommandReg;        // командный регистр
      BYTE bReserved;          // зарезервировано
} IDEREGS, *PIDEREGS, *LPIDEREGS;
```

Всякий, кто читал спецификацию на ATA/ATAPI и хоть однажды сталкивался с программированием устройств с интерфейсом IDE, должен немедленно узнать до боли знакомые регистры Command, Drive/Head, Cylinder High, Cylinder Low, Sector Number, Sector Count и Features, правда, в структуре `IDEREGS` они перечислены почему-то в обратном порядке, но это уже мелочи реализации. Главное, что с помощью этой структуры мы можем вытворять с приводом все мыслимые и немыслимые фокусы, на которые только способно "железо". Даже не верится, что в подсистеме безопасности существует такая дыра размерами со "слонопотама". И это при том, что для управления мини-портом наличие прав администратора совсем не обязательно! Дрожа и подпрыгивая от нетерпения, наскоро заполняем оставшиеся поля структуры `SENDCMDINPARAMS`, как-то: `bDriveNumber` — физический номер привода, считая от нуля, и буфер для передачи данных.

Но ведь мы пока не собираемся записывать никаких данных на диск, верно? Вот и оставим это поле пустым.

Увы! При попытке "скормить" приводу команду, отличную от команд семейства SMART, нас постигает глубокое разочарование, ибо драйвер мини-порта "далеко не дурак" и проверяет содержимое структуры `IDEREGS` перед ее передачей IDE-приводу. Исключение составляет лишь команда идентификации накопителя — 0xEC, о чем Microsoft прямо и заявляет "There are three IDE commands supported in this driver, ID (0xEC), ATAPI ID (0xA1), and SMART (0xB0). The "subcommands" of the SMART commands (features register

values) are limited to the currently defined values (0xD0 through 0xD6, 0xD8 through 0xEF). SMART subcommand 0xD7, write threshold value, is not allowed. Any other command or SMART subcommand will result in an error being returned from the driver. Any SMART command that is not currently implemented on the target drive will result in an ABORT error from the IDE interface" *("Только три IDE-команды поддерживаются этим драйвером: ID (код 0xEC), ATAPI ID (0xA1) и SMART (0xB0). "Подкоманды" базовой команды SMART (передаваемые через feature-регистр), ограничены лишь теми значениями, которые специфицированы на настоящий момент: от 0xD0 до 0xD6 и от 0xD8 до 0xEF. Использование подкоманды с кодом 0xD7, записывающей пороговое значение SMART, заблокировано. Любые другие команды и подкоманды будут игнорироваться драйвером и возвращать сообщение об ошибке. Любые SMART-команды, что не реализованы на текущий момент, в целевом приводе будут возвращать ABORT-ошибку").*

Кажется, что это полный провал, но нет! Ведь эту проверку в принципе можно и отключить! Давайте дизассемблируем драйвер ATAPI.SYS (листинг 4.22), отвечающий за проверку передаваемых IDE-команд на соответствие принадлежности к "белому" списку, и посмотрим, что мы можем сделать.

Листинг 4.22. Фрагмент дизассемблерного листинга драйвера ATAPI.SYS

```
.text:00013714 aScsidisk      db 'SCSIDISK',0      ; DATA XREF: SCSI_MINIPORT+CC↓o
; вот она наша сигнатура          ^^^^^^^^
;
.text:000137DF
.text:000137DF  loc_137DF:                ; CODE XREF: SCSI_MINIPORT+B5↑j
.text:000137DF  mov    [edi], ebx
.text:000137E1  mov    eax, [ebx+18h]
.text:000137E4  push   8                  ; длина сравниваемой строки
.text:000137E6  add    eax, 4
.text:000137E9  push   offset aScsidisk   ; эталонная сигнатура
.text:000137EE  push   eax                ; сигнатура, переданная приложением
.text:000137EF  call   ds:RtlCompareMemory; сигнатуры совпадают?
.text:000137F5  cmp    eax, 8
.text:000137F8  jnz    loc_13898          ; нет, не совпадают, сваливаем отсюда
.text:000137F8
.text:000137FE  mov    esi,[ebx+18h]
.text:00013801  mov    eax,[esi+10h]      ; извлекаем ControlCode
.text:00013804  cmp    eax, 1B0500h       ; IOCTL_SCSI_MINIPORT_SMART_VERSION
.text:00013809  jz     loc_1389F          ; → обработка …SMART_VERSION
.text:0001380F  mov    ecx, 1B0501h       ; IOCTL_SCSI_MINIPORT_IDENTIFY
.text:00013814  cmp    eax, ecx           ;
```

```
.text:00013816  jz     short loc_1382D   ; → обработка …IDENTIFY
.text:00013818  jbe    short loc_13898   ; IF ControlCode < IDENTIFY THEN на выход
.text:0001381A  cmp    eax, 1B050Ah      ; IOCTL_SCSI_MINIPORT_ENABLE_DISABLE…
.text:0001381F  ja     short loc_13898   ; IF ControlCode > ENABLE_DISAB… на выход
.text:00013821  push   ebx               ;
.text:00013822  push   edi               ;
.text:00013823  call   sub_12412         ; обрабатываем остальные SMART-команды
.text:00013828  jmp    loc_1393E
.text:0001382D  ; ———————————————————————————————————————————————————————

.text:00012412  sub_12412    proc near   ; CODE XREF: SCSI_MINIPORT+106↓p
…
.text:00012433  cmp   [ebp+var_1E], 0B0h ; SMART-command
.text:00012437  jnz   loc_12633          ; если это не SMART, то выходим
.text:00012437  ; отсюда начинаются проверки

.text:0001243D  movzx  eax, [ebp+var_1C]
.text:00012441  mov    eax, [ebx+eax*4+0B0h] ; загружаем Drive/Head-регистр в EAX
.text:00012448  test   al, 1              ; сравниваем младший бит AL с единицей
.text:0001244A  jz     loc_1262F          ; если младший бит равен нулю, выходим
.text:00012450  test   al, 2              ; сравниваем следующий бит AL с единицей
.text:00012452  jnz    loc_1262F          ; если он не равен нулю, то выходим
.text:00012458  mov    al, [ebp+var_24]   ; загружаем Feature-регистр в AL
.text:0001245B  cmp    al, 0D0h           ; это SMART READ DATA?
.text:0001245D  mov    [ebx+0CCh], al
.text:00012463  jz     loc_12523          ; если да, то переходим к его обработке
.text:00012469  cmp    al, 0D1h           ; это Obsolete?
.text:0001246B  jz     loc_12523          ; если да, то переходим к его обработке
.text:00012471  cmp    al, 0D8h           ; это SMART ENABLE OPERATIONS?
.text:00012473  jz     short loc_12491    ; если да, то переходим к его обработке
.text:00012475  cmp    al, 0D9h           ; это SMART DISABLE OPERATIONS?
.text:00012477  jz     short loc_12491    ; если да, то переходим к его обработке
.text:00012479  cmp    al, 0DA            ; это SMART RETURN STATUS?
.text:0001247B  jz     short loc_12491    ; если да, то переходим к его обработке
.text:0001247D  cmp    al, 0D2h           ; это SMART ENBL/DSBL ATTRIBUTE AUTOSAVE?
.text:0001247D  cmp    al, 0D2h           ; процессор, ты не ошибся в натуре?!
.text:0001247F  jz     short loc_12491    ; если да, то переходим к его обработке
.text:00012481  cmp    al, 0D4h           ; это SMART EXECUTE OFF-LINE IMMEDIATE?
.text:00012483  jz     short loc_12491    ; если да, то переходим к его обработке
.text:00012485  cmp    al, 0D3h           ; это SMART SAVE ATTRIBUTE VALUES?
.text:00012487  jz     short loc_12491    ; если да, то переходим к его обработке
.text:00012489  cmp    al, 0DBh           ; это SMART ENABLE OPERATIONS?
```

```
.text:0001248B  jnz    loc_12633          ; если нет, то сваливаем
.text:00012491
.text:00012491  loc_12491:                ; CODE XREF: sub_12412+61↑j
.text:00012491  ; отсюда начинается обработка команд
.text:00012491  ;
.text:00012491  push   1
.text:00012493  pop    eax
.text:00012494  cmp    ds:0FFDF02C0h, eax
.text:0001249A  jnz    short loc_124A5
.text:0001249C  cmp    dword ptr [ebx+4], 640h
.text:000124A3  jz     short loc_124A7
.text:000124A5
.text:000124A5  loc_124A5:                ; CODE XREF: sub_12412+88↑j
.text:000124A5  xor    eax, eax
.text:000124A7
.text:000124A7  loc_124A7:                ; CODE XREF: sub_12412+91↑j
.text:000124A7  ; отсюда начинается запись в порт!
.text:000124A7  ;
.text:000124A7  mov    esi, ds:WRITE_PORT_UCHAR
.text:000124AD  test   al, al
.text:000124AF  jz     short loc_124C0
.text:000124B1  mov    al, [ebp+var_1C]
.text:000124B4  shr    al, 1
.text:000124B6  and    al, 1
.text:000124B8  push   eax
.text:000124B9  push   432h
.text:000124BE  call   esi ; WRITE_PORT_UCHAR
```

Таким образом, чтобы разрешить драйверу отправлять IDE-приводу любые команды, мы должны удалить условный переход, расположенный по адресу `0x12437` (в листинге он выделен жирным шрифтом и обведен прямоугольником) на безусловный переход, передающий управление на команду записи по адресу `0x12491`. Только не забудьте после модификации драйвера скорректировать его контрольную сумму, что можно сделать, например, с помощью утилиты EDITBIN.EXE, входящей в состав Microsoft Visual Studio, иначе Windows NT наотрез откажется загружать такой "взломанный" драйвер.

Разумеется, такую операцию допустимо проделывать только на своем собственном драйвере, поскольку всем остальным вряд ли понравится дыра, проделанная в системе безопасности! К тому же, распространение модифицированного ATAPI.SYS вопиющим образом нарушает авторское право самой Microsoft со всеми вытекающими отсюда последствиями. Тем не менее, ваше приложение может безбоязненно "падчить" ATAPI.SYS непосредственно

на компьютерах пользователей, естественно, запрашивая у них подтверждение на правомерность такой операции или, на худой конец, должно быть выполнено упоминание этого аспекта в сопроводительной документации.

В любом случае, данный способ взаимодействия с приводом не стоит сбрасывать со счетов, поскольку это значительно усложняет взлом защиты, созданной на его основе. Ведь далеко не все хакеры осведомлены о тонкостях управления мини-портом и потому с вероятностью, близкой к единице, сядут в глубокую лужу, если, конечно, не упадут в яму глубокого информационного вакуума.

Пример программы, приведенной в листинге 4.23, как раз и демонстрирует передачу ATA-команд IDE-приводу через драйвер мини-порта.

Листинг 4.23. Пример программы, демонстрирующий технику взаимодействия с мини-портом интерфейса SCSI

```
int ATAPI_MINIPORT_DEMO(void)
{
    int     a;
    HANDLE  h;
    char    *buf;
    int     LU = 0;
    DWORD   returned;
    int     controller;
    char    ScsiPort [16];
    char    buffer [sizeof (SRB_IO_CONTROL) + SENDIDLENGTH];

    SRB_IO_CONTROL  *p   = (SRB_IO_CONTROL  *) buffer;
    SENDCMDINPARAMS *pin = (SENDCMDINPARAMS *) (buffer + sizeof (SRB_IO_CONTROL));

    // перебираем оба IDE-контроллера в цикле
    for (controller = 0; controller < 2; controller++)
    {
          // формируем ScsiPort для каждого из котроллеров
          sprintf (ScsiPort, "\\\\.\\Scsi%d:", controller);

          // открываем соответствующий ScsiPort
          h= CreateFile (ScsiPort,GENERIC_READ | GENERIC_WRITE,
               FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0,0);

          if (h == INVALID_HANDLE_VALUE) { // ЕСЛИ ПРОИЗОШЛА ОШИБКА - СВАЛИВАЕМ
               printf("-ERR:Unable to open ScsiPort%d\n",controller);return -1;
          }
```

```
        // перебираем оба устройства на каждом из IDE-контроллеров
        for (LU = 0; LU < 2; LU++)
        {
              // инициализируем входной буфер
              memset (buffer, 0, sizeof (buffer));

              // ПОДГОТАВЛИВАЕМ СТРУКТУРУ SRB_IO_CONTROL,
              // предназначенную для драйвера мини-порта
              p -> Timeout      = 10000;                    // ждать до черта
              p -> Length       = SENDIDLENGTH;             // макс. длина
              p -> HeaderLength = sizeof (SRB_IO_CONTROL);  // размер заголовка
              p -> ControlCode  = IOCTL_SCSI_MINIPORT_IDENTIFY;
              // ^^^ код команды, посылаемой драйверу

              // сигнатура. для ATAPI.SYS это "SCSIDISK"
              strncpy ((char *) p -> Signature, "SCSIDISK", 8);

              // ПОДГОТАВЛИВАЕМ СТРУКТУРУ SENDCMDINPARAMS,
              // содержащую ATA-команды, передаваемые IDE-приводу
              pin -> bDriveNumber                 = LU;
              pin -> irDriveRegs.bCommandReg      = IDE_ATA_IDENTIFY;

              // ПОСЫЛАЕМ УПРАВЛЯЮЩИЙ ЗАПРОС ДРАЙВЕРУ МИНИ-ПОРТА
              if (DeviceIoControl (h, IOCTL_SCSI_MINIPORT, buffer,
              sizeof (SRB_IO_CONTROL) + sizeof (SENDCMDINPARAMS) - 1,
              buffer, sizeof (SRB_IO_CONTROL) + SENDIDLENGTH, &returned, 0))
                    if (buffer[98]!=0)
                    {// в ответ нам возвращается строка с идентификационным
                    // именем IDE-привода, которую мы и выводим на экран
                            for (a = 98; a < 136; a+=2 )
                            printf("%c%c",buffer[a+1],buffer[a]);
                            printf("\n");
                    }
        }
        CloseHandle (h); // закрыть дескриптор данного SCSI мини-порта
    }
    return 0;
}
```

Взаимодействие через порты ввода/вывода

Операционная система Windows NT тщательно оберегает порты ввода/вывода от посягательства со стороны прикладных приложений. Мера эта вынужденная и реализованная под давлением выбранной политики безопасности. Свобода прикладных приложений умышленно ограничивается так, чтобы предотвратить возможные "террористические акты", направленные на подрыв системы или несанкционированный захват конфиденциальной информации. Правом непосредственного доступа к оборудованию обладают лишь драйвера и динамические библиотеки, исполняющиеся в режиме ядра (см. *разд. "Доступ через SCSI мини-порт" этой главы*).

Поневоле вспоминаются слова одного из "отцов-основателей" США, что нация, обменявшая свободу на безопасность, не заслуживает ни того, ни другого. И правда! Как будто бы нельзя "завесить" систему через тот же SPTI/ASPI! Причем для этого даже не понадобится обладать правами администратора! Какая там политика безопасности, какое к черту разграничение доступа, когда ASPI дает доступ к диску на секторном уровне безо всяких проверок на предмет правомерности осуществления этой операции. Хоть сейчас boot-вирусы в загрузочный сектор внедряй! И это при том, что отсутствие доступа к портам ввода/вывода существенно усложняет задачу управления оборудованием и уж тем более создания надежных и трудноломаемых защитных механизмов!

Операционные системы семейства Windows 9x ведут себя более "демократично", однако их "снисходительность" распространяется исключительно на программы MS-DOS, а Win32-приложения возможности прямого доступа к портам, увы, лишены.

Тем не менее, управлять оборудованием с прикладного уровня все-таки возможно. Существует по меньшей мере два пути решения этой проблемы:

- *создание драйвера-посредника, реализующего более или менее прозрачный интерфейс для взаимодействия с портами через механизм IOCTL*;
- *модификация карты разрешения ввода-вывода (I/O Permission Map, — IOPM) с таким расчетом, чтобы обращение к портам перешло в разряд непривилегированных операций, осуществимых и с прикладного уровня.*

Далее оба этих способа подробно рассмотрены. Начнем с интерфейсного драйвера.

В состав NT DDK входит весьма любопытный учебный драйвер PORTIO, создающий виртуальное устройство и реализующий специальный IOCTL-интерфейс, посредством которого прикладные приложения могут манипулировать с портами этого устройства произвольным образом (его исходный текст с минимумом необходимых комментариев расположен в каталоге \NTDDK\src\general\portio). Конечно, виртуальное устройство — это не совсем то, что нам нужно, поскольку диапазон принадлежащих ему портов

ввода/вывода не может пересекаться с портами, принадлежащими другим устройствам, в противном случае система грязно "выругается" и поставит в диспетчере устройств восклицательный знак, предупреждая пользователя об имеющемся конфликте ресурсов. И хотя на работоспособность системы такой конфликт никак не повлияет, созерцание восклицательных знаков уж точно не пойдет впрок пользователям нашей программы.

На самом деле, драйверу, работающему в режиме ядра, никто не запрещает обращаться к любым портам, каким ему только вздумается. Достаточно исключить из тела файла genport.c следующие строки (листинг 4.24), и мы сможем с его помощью читать весь диапазон портов ввода/вывода.

Листинг 4.24. Проверка адресов портов, к которым происходит обращение на принадлежность к диапазону портов виртуального устройства

```
if (nPort >= pLDI->PortCount ||
      (nPort + DataBufferSize) > pLDI->PortCount ||
      (((ULONG_PTR)pLDI->PortBase + nPort) & (DataBufferSize - 1)) != 0)
{
      return STATUS_ACCESS_VIOLATION;   // Illegal port number
}
```

Также следует обратить внимание на то, что драйвер ожидает получить не абсолютный адрес порта, а *относительный*, отсчитываемый от адреса базового порта, задаваемого при добавлении виртуального устройства в систему (листинг 4.25).

Листинг 4.25. Вычисление действительного адреса порта через базовый

```
case IOCTL_GPD_READ_PORT_UCHAR:
    *(PUCHAR)pIOBuffer=READ_PORT_UCHAR((PUCHAR)((ULONG_PTR)pLDI->PortBase+nPort));
    break;
```

Очевидно, что текст, выделенный жирным шрифтом, следует удалить, — в этом случае драйвер сможет оперировать абсолютными, а не относительными портами, и мы без труда сможем прорваться к любому порту системы! Причем, если мы перенесем модифицированный нами драйвер на Windows 9x, наши приложения будут работать в обеих операционных системах и останутся зависимыми разве что от самого оборудования. Но, с другой стороны, всякий, кто стремится дорваться до портов, должен отдавать себе отчет в том, зачем это ему нужно и какие сложности ему придется преодолеть.

Конечно, поскольку возможность бесконтрольного доступа ко всем имеющимся портам ввода/вывода существенно ослабляет и без того уязвимую

операционную систему, нелишним будет ввести в драйвер кое-какие дополнительные проверки и ограничения. Скажем, запретить прямое обращение ко всему, что не является CD-ROM-приводом. В противном случае, если ваша программа получит сколь-нибудь широкое распространение, толпы вандалов ринутся писать зловредных троянских коней, военная мощь которых окажется практически безграничной, и совладеть с ними будет очень и очень трудно. С другой стороны, за все время существования интерфейса ASPI не было зафиксировано ни одной попытки использовать его для деструктивных целей, хотя такая возможность до сих пор имеется.

Другой недостаток предложенного способа управления устройствами заключается в его катастрофически низком быстродействии. Вызовы `DeviceIoControl` распадаются на десятки тысяч машинных команд (!), "благодаря" чему время обработки запросов становится слишком большим, а измерение физических характеристик спиральной дорожки (если мы действительно захотим эти характеристики измерять) — неточным. К тому же, функция `DeviceIoControl` громоздка и неизящна, а самое неприятное в том, что на нее очень легко поставить точку останова (breakpoint), и потому участь такой защиты заранее предрешена. Во времена MS-DOS, когда взаимодействие с оборудованием осуществлялось посредством машинных команд `IN` и `OUT`, локализовать защитный код в теле программы было значительно сложнее, а управлять устройствами с их помощью существенно легче и — главное — намного производительнее.

Считается, что в среде Windows NT прямое обращение к портам возможно только на уровне ядра, а приложения вынуждены общаться с портами через высокоуровневый интерфейс, предоставляемый драйвером. И хотя этот интерфейс может быть полностью прозрачным (драйверу ничего не стоит перехватить исключение, возникающее при попытке чтения/записи в порт с прикладного уровня, и выполнить этот запрос самостоятельно), и все-таки это не то.

На самом деле, выполнять команды `IN`/`OUT` можно и на прикладном уровне, правда, не без помощи недокументированных возможностей операционной системы и документированных, но малоизвестных особенностей реализации защищенного режима работы в процессорах Intel 80386+. Вот с процессоров мы, пожалуй, и начнем. Давайте откроем *"Instruction Set Reference"* и посмотрим, как "устроена" машинная команда `OUT`. Среди прочей полезной информации мы найдем и ее псевдокод, который выглядит приблизительно так, как это показано в листинге 4.26.

Листинг 4.26. Псевдокод инструкции `OUT`

```
if ((PE == 1) && ((CPL > IOPL) || (VM == 1)))
{
      /* Protected mode with CPL > IOPL or virtual-8086 mode */
```

```
        if (Any I/O Permission Bit for I/O port being accessed == 1)
              #GP(0);            /* I/O operation is not allowed */
        else
              DEST ← SRC;        /* Writes to selected I/O port */
}
        else
{
        /* Real Mode or Protected Mode with CPL <= IOPL */
        DEST ← SRC;              /* Writes to selected I/O port */
}
```

Обратите внимание! Обнаружив, что полномочий текущего уровня привилегий категорически недостаточно для выполнения данной машинной инструкции, процессор не спешит выбрасывать исключение general protection fault, а дает ей еще один шанс, осуществляя дополнительную проверку на предмет состояния *карты разрешения ввода/вывода* (*I/O permission bitmap*), и если бит памяти, соответствующий данному порту, не равен единице, то вывод в порт осуществляется, несмотря ни на какие запреты со стороны CPL!

Таким образом, для взаимодействия с портами с прикладного уровня нам достаточно всего лишь скорректировать карту разрешения ввода/вывода, после чего подсистема защиты операционной системы Windows NT перестанет нам мешать, поскольку контроль доступа к портам осуществляется не на программном, а на аппаратном уровне, и если процессор перестанет выбрасывать исключения, то операционная система ничего не узнает о происходящем!

Проблема в том, что подавляющее большинство авторов книг по ассемблеру о карте разрешения ввода/вывода даже не упоминают, и лишь немногие программисты знают о ее существовании — те, кто предпочитает оригинальную документацию корявым переводам и пересказам.

Обратившись к *"Architecture Software Developer's Manual Volume 1: Basic Architecture"*, мы узнаем, что карта ввода/вывода находится в *сегменте состояния задачи* (*TSS — Task State Segment*), — точнее, ее действительное смещение относительно начала TSS определяется 32-битным полем, расположенным в 0x66 и 0x67 байтах сегмента состояния задачи. Нулевой бит этой карты отвечает за нулевой порт, первый — за первый, второй — за второй и т. д. вплоть до старшего бита 0x2000 байта, отвечающего за 65 535 порт. Битовую карту завершает так называемый *байт-терминатор*, имеющий значение `0xFF`. Вот, собственно, и все. Порты, чьи биты сброшены в нулевое значение, доступны с прикладного уровня безо всяких ограничений. Разумеется, сама карта ввода/вывода доступна лишь драйверам, но не приложениям, поэтому без написания собственного драйвера нам все равно не обойтись. Однако этот драйвер будет работать только на стадии

своей инициализации, а весь дальнейший ввод/вывод пойдет напрямую, даже если выгрузить драйвер из памяти.

Теперь плохая новость. В Windows NT смещение карты ввода/вывода по умолчанию находится за пределами сегмента состояния задачи и потому модифицировать карту ввода/вывода не так-то просто, поскольку ее вообще нет! Процессор, кстати говоря, на такую ситуацию реагирует вполне спокойно, но доступ к портам ввода/вывода с прикладного уровня тем не менее запрещает.

На самом же деле карта ввода/вывода в TSS все-таки есть, но она умышленно заблокирована системой, чтобы не дать прикладным приложениям своевольничать. Исключение составляют лишь высокопроизводительные графические библиотеки, напрямую обращающиеся к портам ввода/вывода с прикладного режима. Как нетрудно догадаться, такой трюк дает Microsoft значительную фору перед конкурентами, вынужденными управлять портами либо с уровня ядра, либо через интерфейс, предоставляемый видеодрайвером. Естественно, оба этих способа значительно проигрывают в производительности прямому доступу к портам.

Однако попытка подкорректировать указатель на карту ввода/вывода ни к чему не приводит, поскольку коварная Windows NT хранит копию этого значения в контексте процесса, а потому при переключении контекста указатель на прежнюю карту автоматически восстанавливается. С одной стороны, это хорошо, так как каждый процесс может иметь свою собственную карту ввода/вывода, а с другой, штатная документация от Microsoft не содержит и намека на то, как с этой картой работать.

Правда, можно схитрить и увеличить размер сегмента состояния задачи так, чтобы адрес карты ввода/вывода, прежде указывающий за его конец, теперь приходился на действительную и подвластную нам область памяти. А поскольку в хвосте последней страницы, занятой TSS, имеется всего лишь 0xF55 байт, максимальный размер карты, которую мы только можем создать в этом промежутке, охватывает всего лишь 31392 портов ввода/вывода. Хотя, если говорить честно, остальные порты нам все равно вряд ли понадобятся, так что ничего трагичного в таком ограничении и нет.

Впрочем, существуют и более изящные способы решения этой проблемы. Усилиями Дейла Робертса были обнаружены три полностью недокументированные функции: `Ke386SetIoAccessMap()`, `Ke386QueryIoAccessMap()` и `Ke386IoSetAccessProcess()`, которые, как и следует из их названий, обеспечивают вполне легальный способ управления картой ввода/вывода. "Полностью недокументированные" в том смысле, что даже заголовочные файлы из DDK не содержат их прототипов (а, как известно, в заголовочных файлах DDK перечислено множество недокументированных функций). Тем не менее, библиотека NTOSKRNL их все-таки экспортирует, и они легко доступны с уровня драйверов.

Подробнее обо всем этом можно прочитать в статье их первооткрывателя — Дейла Робертса, — перевод которой можно найти, в частности, по следующему адресу: **http://void.ru/?do=printable&id=701**. Здесь же мы рассмотрим их лишь кратко. Итак, функция `Ke386SetIoAccessMap` принимает два аргумента: двойное слово, которое, будучи установленным в единицу, заставляет функцию копировать карту ввода/вывода указатель, на которую передан ей со вторым аргументом. Функция `Ke386QueryIoAccessMap` принимает те же самые аргументы, но осуществляет прямо противоположную операцию, извлекая текущую карту ввода/вывода из сегмента состояния задачи и копируя ее в указанный буфер. Наконец, функция `Ke386IoSetAccessProcess` принимает со своим вторым аргументом указатель на структуру процесса, полученный вызовом документированной функции `PsGetCurrentProcess()`. Первый аргумент играет ту же самую роль, что и в предыдущих функциях: нулевое значение переводит указатель на карту ввода/вывода за границы TSS, тем самым запрещая доступ к портам с прикладного уровня, а единичное — активизирует ранее переданную карту ввода/вывода.

Пример драйвера, открывающего прямой доступ к портам ввода/вывода на прикладном уровне и приведенный в листинге 4.27, все это, собственно, и демонстрирует.

Листинг 4.27. [/etc/GIVEIO.c] Демонстрационный пример драйвера

```
/*---------------------------------------------------------------------------
 *
 *                  ДРАЙВЕР. РАЗРЕШАЕТ ВЫПОЛНЕНИЕ
 *            МАШИННЫХ КОМАНД IN/OUT НА ПРИКЛАДНОМ УРОВНЕ
 *            ==========================================
 *
 *   ВНИМАНИЕ! Я, Крис Касперски, не имею никакого отношения к этой программе!
 * ------------------------------------------------------------------------
 *
 * GIVEIO.SYS: by Dale Roberts
 * КОМПИЛЯЦИЯ: Используйте средство DDK BUILD
 * НАЗНАЧЕНИЕ: Предоставить доступ к прямому в/в процессам режима пользователя
---------------------------------------------------------------------------*/

#include <ntddk.h>

/* Имя нашего драйвера устройства */
#define DEVICE_NAME_STRING L"giveio"

// Структура" IOPM. это просто массив байт размером 0x2000, содержащий
```

```
// 8K * 8 бит == 64K бит IOPM, которые  покрывают все  64 Кб  адресное
// пространство ввода/вывода x86 процессоров.
// Каждый нулевой бит  предоставляет  доступ к соответствующему  порту
// для user-mode процесса; каждый единичный бит запрещает доступ к в/в
// через соответствующий порт
#define IOPM_SIZE 0x2000
typedef UCHAR IOPM[IOPM_SIZE];

// массив нулей, который  копируется в настоящую IOPM в TSS посредством
// вызова dsKe386SetIoAccessMap()
// необходимая память выделяется во время загрузки драйвера
IOPM *IOPM_local = 0;

// это две полностью недокументированные функции, которые мы используем,
// чтобы дать доступ к в/в вызывающему процессу
// * Ke386IoSetAccessMap()     - копирует  переданную  карту в/в в  TSS
// * Ke386IoSetAccessProcess() - изменяет указатель смещения IOPM, после
//                               чего только  что  скопированная  карта
//                               начинает использоваться
void Ke386SetIoAccessMap(int, IOPM *);
void Ke386QueryIoAccessMap(int, IOPM *);
void Ke386IoSetAccessProcess(PEPROCESS, int);

// ОСВОБОДИТЬ ВСЕ ВЫДЕЛЕННЫЕ РАНЕЕ ОБЪЕКТЫ
VOID GiveioUnload(IN PDRIVER_OBJECT DriverObject)
{
      UNICODE_STRING uniDOSString;
      WCHAR DOSNameBuffer[] = L"\\DosDevices\\" DEVICE_NAME_STRING;

      if(IOPM_local) MmFreeNonCachedMemory(IOPM_local, sizeof(IOPM));
      RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);
      IoDeleteSymbolicLink (&uniDOSString);
      IoDeleteDevice(DriverObject->DeviceObject);
}

//-----------------------------------------------------------------------------
//      устанавливаем IOPM (карту разрешения в/в) вызывающего процесса так, чтобы
//      ему предоставлялся  полный  доступ к в/в. Массив  IOPM_local[]  содержит
//      одни нули, соответственно, IOPM обнулится.
//      Если OnFlag == 1, процессу предоставляется доступ к в/в;
//      Если он равен 0, доступ запрещается.
//-----------------------------------------------------------------------------
```

```
VOID SetIOPermissionMap(int OnFlag)
{
      Ke386IoSetAccessProcess(PsGetCurrentProcess(), OnFlag);
      Ke386SetIoAccessMap(1, IOPM_local);
}

void GiveIO(void)
{
      SetIOPermissionMap(1);
}

//-----------------------------------------------------------------------------
//      служебный обработчик для user-mode вызова CreateProcess().
//      эта функция введена в таблицу вызовов функций объекта драйвера с помощью
//      DriverEntry(). Когда user-mode приложение вызывает CreateFile(), эта
//      функция получает управление все еще в контексте вызвавшего приложения,
//      но с CPL (текущий уровень привилегий процессора) установленым в 0.
//      Это позволяет производить операции, возможные только в kernel mode.
//      GiveIO вызывается для предоставления вызывающему процессу доступа к в/в.
//      Все, что приложение режима пользователя, которому нужен доступ к в/в,
//      должно сделать - это открыть данное устройство, используя CreateFile()
//      Никаких других действий не нужно.
//-----------------------------------------------------------------------------
NTSTATUS GiveioCreateDispatch(IN PDEVICE_OBJECT DeviceObject,IN PIRP Irp)
{
      GiveIO(); // give the calling process I/O access

      Irp->IoStatus.Information      = 0;
      Irp->IoStatus.Status            = STATUS_SUCCESS;

      IoCompleteRequest(Irp, IO_NO_INCREMENT); return STATUS_SUCCESS;
}

//-----------------------------------------------------------------------------
//      процедура входа драйвера. эта процедура вызывается только раз после
//      загрузки драйвера в память. она выделяет необходимые ресурсы для работы
//      драйвера. в нашем случае она выделяет память для массива IOPM и создает
//      устройство, которое может открыть приложение режима пользователя.
//      она также создает символическую ссылку на драйвер устройства.
//      что позволяет user-mode приложению получить доступ к нашему драйверу,
//      используя \\.\giveio нотацию.
//-----------------------------------------------------------------------------
```

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,IN PUNICODE_STRING RegistryPath)
{
      NTSTATUS      status;
      PDEVICE_OBJECT      deviceObject;
      UNICODE_STRING      uniNameString, uniDOSString;

      WCHAR NameBuffer[]      = L"\\Device\\" DEVICE_NAME_STRING;
      WCHAR DOSNameBuffer[]      = L"\\DosDevices\\" DEVICE_NAME_STRING;

      // выделим буфер для локальной IOPM и обнулим его
      IOPM_local = MmAllocateNonCachedMemory(sizeof(IOPM));
      if(IOPM_local == 0) return STATUS_INSUFFICIENT_RESOURCES;
      RtlZeroMemory(IOPM_local, sizeof(IOPM));

      // инициализируем драйвер устройства и объект устройства (device object)
      RtlInitUnicodeString(&uniNameString, NameBuffer);
      RtlInitUnicodeString(&uniDOSString, DOSNameBuffer);
      status = IoCreateDevice(DriverObject, 0, &uniNameString,FILE_DEVICE_UNKNOWN,
                                        0, FALSE, &deviceObject);
      if(!NT_SUCCESS(status)) return status;

      status = IoCreateSymbolicLink (&uniDOSString, &uniNameString);
      if (!NT_SUCCESS(status)) return status;

      // инициализируем точки входа драйвера в объекте драйвера
      // все, что нам нужно, это операции создания (Create) и выгрузки (Unload)
      DriverObject->MajorFunction[IRP_MJ_CREATE] = GiveioCreateDispatch;
      DriverObject->DriverUnload = GiveioUnload;

      return STATUS_SUCCESS;
}
```

Пример демонстрации ввода/вывода в порт с прикладного уровня показан в листинге 4.28.

Листинг 4.28. [/etc/GIVEIO.demo.c] Пример ввода/вывода в порт с прикладного уровня

```
/*-----------------------------------------------------------------------------
 *
 *            ДЕМОНСТРАЦИЯ ВЫЗОВА IN/OUT НА ПРИКЛАДНОМ УРОВНЕ
 *      (внимание! драйвер  GIVEIO.SYS должен быть предварительно загружен!)
```

```
 *      ==================================================================
 *
 *   ВНИМАНИЕ! Я, Крис Касперски, не имею никакого отношения к этой программе!
 * ---------------------------------------------------------------------------
 *
 * GIVEIO.TST: by Dale Roberts
 * НАЗНАЧЕНИЕ: Тестирование драйвера GIVEIO, производя какой-нибудь в/в.
 *           : (мы обращаемся к внутреннему динамику PC)
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <windows.h>
#include <math.h>
#include <conio.h>

typedef struct {
      short int pitch;
      short int duration;
} NOTE;

// ТАБЛИЦА НОТ
NOTE notes[] = {{14, 500}, {16, 500}, {12, 500}, {0, 500}, {7, 1000}};

// УСТАНОВКА ЧАСТОТЫ ДИНАМИКА PC В ГЕРЦАХ
// ДИНАМИК УПРАВЛЯЕТСЯ ТАЙМЕРОМ INTEL 8253/8254 С ПОРТАМИ В/В 0X40-0X43
void setfreq(int hz)
{
      hz = 1193180 / hz;          // базовая частота таймера 1.19MHz
      _outp(0x43, 0xb6);          // Выбор таймера 2, операция записи,режим 3
      _outp(0x42, hz);            // устанавливаем делитель частоты
      _outp(0x42, hz >> 8);       // старший байт делителя
}

//-----------------------------------------------------------------------------
//      длительность ноты задается в долях частоты 400 Hz, число 12 задает масштаб
//      Спикер управляется через порт 0x61. Установка двух младших битов разрешает
//      канал 2 таймера 8253/8254 и включает динамик.
//-----------------------------------------------------------------------------
void playnote(NOTE note)
{
      _outp(0x61, _inp(0x61) | 0x03);            // включаем динамик
            setfreq((int)(400 * pow(2, note.pitch / 12.0))); Sleep(note.duration);
      _outp(0x61, _inp(0x61) & ~0x03);            // выключаем
```

```
}

//-----------------------------------------------------------------------------
//      открытие и закрытие устройства GIVEIO, что дает нам прямой доступ к в/в;
//      потом пытаемся проиграть музыку
//-----------------------------------------------------------------------------
int main()
{
      int            i;
      HANDLE   h;

      h = CreateFile("\\\\.\\giveio", GENERIC_READ, 0, NULL, OPEN_EXISTING,
                              FILE_ATTRIBUTE_NORMAL, NULL);
      if(h == INVALID_HANDLE_VALUE)
      {
            printf("Couldn't access giveio device\n"); return -1;
      }
      CloseHandle(h);

      for(i = 0; i < sizeof(notes)/sizeof(int); ++i) playnote(notes[i]);

      return 0;
}
```

Теперь поговорим о том, как данный способ взаимодействия с портами ввода/вывода может быть использован на благо защитных механизмов. Допустим, наша защита привязывается к физическому дефекту поверхности лазерного диска. Тогда все, что нам надо — попытаться как можно незаметнее прочитать этот сектор: если он действительно не читается, диск можно считать оригинальным и наоборот. Прямое управление приводом через порты ввода/вывода с вероятностью, близкой к единице, останется незамеченным даже бывалыми хакерами, потому такой вариант им попросту не придет в голову! Единственное, о чем следует позаботиться, — не дать обнаружить защитный код по перекрестным ссылкам, оставленным "ругательным" сообщением, которое выводится на экран в том случае, если диск признан пиратским.

Тем не менее, матерых хакеров на такую наживку не возьмешь! Злорадно ухмыльнувшись, они просто поставят точку останова на ввод/вывод в порты 0x1F7/0x177 (для Primary и Secondary приводов соответственно). А чтобы не утонуть в море обращений к приводу через функции API, задействуют условные точки останова, приказывая отладчику "всплывать" только в том случае, если адрес машинной команды, осуществляющей ввод/вывод, находится

ниже адреса `0x70000000`, т. е., другими словами, принадлежит пользовательскому приложению, а не ядру.

Но что нам мешает с прикладного уровня выполнить команду ввода/вывода по адресу, принадлежащему ядру? Достаточно просто просканировать верхнюю половину адресного пространства на предмет наличия команд `OUT DX, AL` (опкод `0xEE`) и `IN AL, DX` (опкод `0xEC`). Спрашиваете: а как мы сможем вернуть управление? Да очень просто, — с помощью обработки структурных исключений. Если машинная команда, следующая за `IN`/`OUT`, возбуждает исключение (а таких команд — хоть пруд пруди), то, перехватив его, мы сможем как ни в чем не бывало продолжить выполнение программы.

Достоинство этого приема в том, что точка останова, поставленная хакером на порты ввода/вывода, не сработает (точнее, сработает, но будет тут же проглочена фильтром), а недостаток: неоправданное усложнение защитного механизма.

Доступ через MSCDEX-драйвер

Знаменитый MSCDEX, созданный еще во времена царствования MS-DOS, несмотря на свои многочисленные недостатки, все-таки обеспечивал программистов всем необходимым им функционалом и достаточно полно поддерживал возможности существующих в то время приводов. Так, например, чтение отдельных секторов осуществлялось функцией `1508h` прерывания INT 2Fh, а если возникала необходимость спуститься на "сырой" уровень, мы всегда могли попросить MSCDEX передать приводу ATAPI-пакет напрямую, чем занималась функция `1510h` того же прерывания (загляните в (Interrupt List), если нуждаетесь в более подробной информации).

Забавно, но возможности штатного драйвера "новейшей" и "могучей" Windows 9x не в пример беднее, и спуститься на секторный уровень, при этом не набив себе шишек, под ее управлением, по-видимому, нельзя. Судя по всему, архитекторы системы сочли секторный обмен ненужным и к тому же системно-зависимым, а "правильные" приложения должны разрабатываться как полностью переносимые и довольствующиеся исключительно стандартными вызовами интерфейса Win32 API. Все остальное от лукавого!

Между тем, для сохранения обратной совместимости с программами, написанными для MS-DOS и Windows 3.1, операционная система Windows 95 поддерживает MSCDEX-интерфейс, причем по соображениям производительности, реализует его не в "настоящем" MSCDEX, который и вовсе может отсутствовать на диске, а в CD-ROM-драйвере, исполняющемся в 32-разрядном защищенном режиме. Выходит, что весь необходимый нам функционал в системе все-таки есть, а значит, есть и надежда как-то до него добраться. Естественно, с уровня ядра эта задача решается без проблем, но писать свой собственный драйвер только для того, чтобы пробить интерфейсную шахту к уже существующему драйверу, — это маразм какой-то!

К счастью, готовый (и даже задокументированный!) интерфейс между win32-приложениями и MSCDEX-драйвером в системе Windows 9x действительно есть. К несчастью, он реализован через ж (и именно через ж). В общих чертах схема прокладывания туннеля к MSCDEX выглядит приблизительно так: создав 16-разрядную DLL, мы получаем возможность взаимодействовать с DPMI через функции прерывания INT 31h.

Конкретно нас будет интересовать функция 1508h — `DPMI Simulate Real Mode Interrupt`, позволяющая вызывать прерывания реального режима из защищенного. Обращаясь к эмулятору MSCDEX-драйвера через родное для него прерывание INT 2Fh, мы можем делать с приводом практически все, что нам только вздумается, поскольку интерфейс драйвера MSCDEX, как уже отмечалось ранее, могуч и велик.

Таким образом, вырисовывается следующий программистский маршрут: Win32-приложение → 16-разрядная DLL → DMPI Simulate RM Interrupt → MSCDEX → CDFS. Не слишком ли наворочено, а? Уж лучше воспользоваться ASPI или засесть за написание собственного драйвера. Тем не менее, даже если вы не собираетесь управлять приводом через MSCDEX, знать о существовании такого способа взаимодействия с оборудованием все-таки небесполезно, особенно, если вы планируете заняться взломом чужих программ. В этом случае точки останова, установленные на API-функции, ничего не дадут, поскольку чтение секторов осуществляется через прерывания INT 31h (DMPI) и INT 2Fh. К сожалению, прямая установка точек останова на последние дает очень много ложных срабатываний, а применение фильтров вряд ли окажется эффективным, поскольку количество возможных вариаций слишком велико. Уж лучше поискать вызовы прерываний в дизассемблерном тексте программы!

Дополнительную информацию по этому вопросу можно найти в технической заметке Q137813, входящей в состав MSDN, распространяемой вместе с Microsoft Visual Studio и озаглавленную как *"How Win32 Applications Can Read CD-ROM Sectors in Windows 95"*. Полный перечень DMPI- и MSCDEX-функций содержится в Interrupt-List Ральфа Брауна, так что никаких проблем с использованием данного приема у вас возникнуть не должно (правда, раздобыть компилятор, способный генерировать 16-разрядный код и линкер под Windows 3.1 сегодня не так-то просто! К слову сказать, Microsoft Visual Studio 6.0 для этой цели уже не подходит, ибо, начиная с некоторой версии — уже сейчас и не вспомню какой — он утратил возможность создания проектов под MS-DOS/Windows 3.1).

Далее приводится ключевой фрагмент программы (листинг 4.29), позаимствованный из MSDN и демонстрирующий технику вызова прерываний реального режима из 16-разрядных DLL, исполняющихся в среде Windows.

Листинг 4.29. Ключевой фрагмент программы, демонстрирующей технику взаимодействия с драйвером MSCDEX из 16-разрядного защищенного режима

```
BOOL FAR PASCAL MSCDEX_ReadSector(BYTE bDrive, DWORD StartSector, LPBYTE RMlpBuffer)
{
    RMCS      callStruct;
    BOOL      fResult;

    // Prepare DPMI Simulate Real Mode Interrupt call structure with
    // the register values used to make the MSCDEX Absolute read call.
    // Then, call MSCDEX using DPMI and check for errors in both the DPMI
    // call and the MSCDEX call
    BuildRMCS (&callStruct);
    callStruct.eax = 0x1508;              // MSCDEX функция "ABSOLUTE READ"
    callStruct.ebx = LOWORD(RMlpBuffer);  // смещение буфера для чтения сектора
    callStruct.es  = HIWORD(RMlpBuffer);  // сегмент буфера для чтения сектора
    callStruct.ecx = bDrive;              // буква привода 0=A, 1=B, 2=C и т. д.
    callStruct.edx = 1;                   // читаем один сектор
    callStruct.esi = HIWORD(StartSector); // номер читаемого сектора(старшее слово)
    callStruct.edi = LOWORD(StartSector); // номер читаемого сектора(младшее слово)

    // вызываем прерывание реального режима
    if (fResult = SimulateRM_Int (0x2F, &callStruct))
          fResult = !(callStruct.wFlags & CARRY_FLAG);

    return fResult;
}

BOOL FAR PASCAL SimulateRM_Int(BYTE bIntNum, LPRMCS lpCallStruct)
{
    BOOL fRetVal = FALSE;      // Assume failure

    __asm
    {
          push di                 ; сохраняем регистр DI
          mov  ax, 0300h          ; DPMI Simulate Real Mode Interrupt
          mov  bl, bIntNum        ; номер прерывания реального режима для вызова
          mov  bh, 01h            ; бит 0 = 1; все остальные должны быть равны нулю
          xor  cx, cx             ; ничего не копируем из стека PM в стек RM
          les  di, lpCallStruct   ; указатель на структуру со значением регистров
          int  31h                ; шлюз к DMPI
```

```
        jc   END1                ; если ошибка, - прыгаем на END1
        mov  fRetVal, TRUE       ; все OK
    END1:
        pop di                   ; восстанавливаем регистр DI
    }

    // возвращаемся
    return (fRetVal);
}
```

Взаимодействие через собственный драйвер

Несмотря на то, что Windows позволяет управлять устройствами и с прикладного уровня, достаточно многие разработчики предпочитают осуществлять такое управление через свой собственный драйвер, который может взаимодействовать с приводом как напрямую, так и через его драйвер. Последний способ более предпочтителен, поскольку он позволяет абстрагироваться от конкретного оборудования и обеспечивает единый унифицированный интерфейс для всех приводов. Большинство таких драйверов "подключаются" к ATAPI и/или SCSI-порту и взаимодействуют с диском приблизительно так же, как и ASPI-драйвер, уже рассмотренный нами.

Взаимодействие с прикладными приложениями обычно осуществляется посредством специальных кодов IOCTL, передаваемых драйверу функцией `DeviceIoControl`. "Специальных" — потому, что разработка протокола взаимодействия драйвера с устройством целиком лежит на совести (и фантазии) создателя этого самого драйвера, и никакой стандартизацией здесь даже отдаленно не пахнет! К тому же, `DeviceIoControl` — это не единственно возможный вариант. Драйверу, исполняющемуся в нулевом кольце, формально доступны все ресурсы операционной системы, и при желании можно осуществить самые "крутые извращения". Например, взаимодействовать с приложением через общую область памяти. Тогда точки останова, установленные на `DeviceIoControl` не дадут никакого результата! Однако подавляющее большинство драйверов работают через IOCTL и не блистают оригинальностью. В каком-то смысле такая позиция вполне оправдана. Действительно, с ростом извращенности драйвера увеличивается и его конфликтность, а совместимость с другими программами (и операционными системами) резко падает. К тому же, "навороченный" драйвер значительно труднее довести до ума, чем простой. С другой стороны, неизвращенный драйвер очень легко взломать, и его разработка ничем не оправдает себя. Уж лучше воспользоваться тем же ASPI, который обеспечивает полнофункциональный

низкоуровневый и при этом системно-независимый интерфейс. Тогда вам не придется создавать реализации своего драйвера под все существующие операционные системы и лихорадочно переписывать код при выходе новых версий Windows.

Сводная таблица характеристик различных интерфейсов

В сводной таблице, приведенной далее (табл. 4.6), показаны основные характеристики всех ранееописанных методик доступа. Как видно, наибольшее количество очков набрал метод доступа через ASPI, обеспечивающий простой, симпатичный и к тому же системно-независимый интерфейс управления накопителями. Следом за ним идет STPI, основной недостаток которого заключается в том, что он поддерживается лишь операционными системами семейства NT и не работает на "народной" Windows 9x. Неплохой идеей выглядит создание собственного драйвера, — будучи реализованным под Windows NT и Windows 9x (кстати, WDM-драйвера на уровне исходного кода совместимы с этими двумя системами), обеспечит возможность работы ваших приложений как в NT, так и в Windows 9x.

Таблица 4.6. *Сравнение различных методов доступа*

Характеристика	Метод доступа								
	CDFS	cocked-mode	MSC DEX	ASPI	SPTI	SCSI port	mini port	own driver	IOPM
Наличие в ОС Windows 9x	—	—	+	+	—	—	—	+	н/д
Наличие в ОС Windows NT	+	+	—	+	+	+	+	+	+
Требует права системного администратора	нет	нет	—	нет	да	нет	нет	хз*	**
Поддерживает CD-DA	да	нет	да	да	да	да	да	да	да
Поддерживает CD data	да	да	да	да	да	да	да	да	да
"Сырое" чтение с CD-DA	да	нет	да	да	да	да	да	да	да
"Сырое" чтение с CD data	нет	нет	да	да	да	да	да	да	да
Потенциально опасен	нет	нет	нет	да	нет	нет	нет	да	да

Таблица 4.6 *(окончание)*

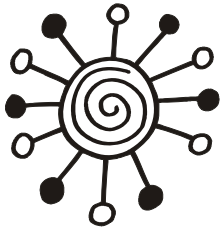
Характеристика	**Метод доступа**								
	CDFS	**cocked-mode**	**MSCDEX**	**ASPI**	**SPTI**	**SCSI port**	**mini port**	**own driver**	**IOPM**
Хорошо документирован	да	да	да	да	нет	нет	Нет	да	нет
Легкость использования	да	да	нет	да	да	да	нет	нет	нет

* — здесь и далее "хз" обозначает "зависит от реализации"

** — установка драйвера требует наличия прав администратора на локальной машине, но вот его последующее использование — нет

Ячейки с неблагоприятными значениями характеристик выделены черным цветом.

Глава 5



Способы разоблачения защитных механизмов

Защита, нуждающаяся в низкоуровневом доступе к CD, обязательно выдаст себя наличием функций `DeviceIoControl` и/или `SendASPI32Command` в таблице импорта. Если же защитный механизм загружает эти функции динамически, "поймать его за хвост" можно установкой точек останова на LoadLibrary/GetProcAddress (однако опытные программисты могут отважиться на самостоятельный поиск требуемых им функций в памяти, — и это отнюдь не такая трудная задача, какой она кажется).

Также в теле программы могут присутствовать строки: "`\\.\`", "`SCSI`", "`CdRom`", "`Wnaspi32.dll`" и другие. Установив точку останова на первый байт строки, мы сможем мгновенно локализовать защитный код при первом его к ним обращении. Чтобы этого не произошло, разработчики часто шифруют все текстовые строки, однако большинство из них ограничивается примитивной статической шифровкой (которая обычно осуществляется программой ASPack или подобной ей), а потому, если дождаться завершения расшифровки и вызвать отладчик после, а не до запуска программы, все текстовые строки предстанут перед нами в прямом виде! Динамическая шифровка намного надежней. В этом случае текстовые строки расшифровываются непосредственно перед их передачей в соответствующую API-функцию, а потом зашифровываются вновь. Но и динамическую шифровку при желании можно преодолеть! Достаточно поставить условную точку останова на функцию `CreateFile`, которой эти текстовые строки и передаются, всплывая в том и только в том случае, если первые четыре байта имени файла равны "`\\.\`". Пример ее вызова может выглядеть, например, так: `bpx CreateFileA if (*esp->4=='\\\\.\\')`, после чего останется только "пожинать урожай".

Естественно, под "урожаем" понимается, во-первых, имя самого открываемого файла, а точнее — драйвера (это уже многое что дает), и, во-вторых, возвращенный функцией `CreateFile` дескриптор. Далее можно поступить двояко: либо установить точку останова на ту ячейку памяти, в которой этот дескриптор сохраняется, либо установить условную точку останова на функцию

DeviceIoControl, отлавливая только те ее вызовы, которые нам необходимы. Пример сеанса работы с отладчиком приведен в листинге 5.1.

Листинг 5.1. Пример изобличения и разоблачения защитного механизма в Soft-Ice

```
:bpx CreateFileA if (*esp->4=='\\\\.\\')    (ставим точку останова)
:x                                          (выходим из отладчика)
…
(отладчик немного думает, а потом всплывает в момент вызова CreateFileA)
:P RET                                      (выходим из CreateFileA)
:? eax                                      (узнаем значение дескриптора)
00000030  0000000048  "0"                   (ответ отладчика)
:DeviceIoControlA if (*esp->4==0x30)        (ставим точку останова на DeviceIoCntrl)
(подумав, отладчик всплывает в момент вызова DeviceIoControl)
:P RET                                      (выходим из DeviceIoControl)
: U                                         (все! мы нашли защиту!)
001B:00401112     LEA      ECX,[EBP-38]
001B:00401115     PUSH     ECX            ;
001B:00401116     PUSH     0004D004       ; вот он, IOCTL_SCSI_PASS_THROUGH_DIRECT!
001B:0040111B     MOV      EDX,[EBP-0C]
001B:0040111E     PUSH     EDX
001B:0040111F     CALL     [KERNEL32!DeviceIoControl]
```

Как видно, поиск функции DeviceIoControl не занял много времени. Остается проанализировать передаваемый ей IOCTL-код (в нашем случае IOCTL_SCSI_PASS_THROUGHT_DIRECT) и его параметры, передаваемые через стек одним двойным словом выше.

Некоторые разработчики помещают критическую часть защитного кода в драйвер, надеясь, что хакеры там ее не найдут. Наивные! Драйвера в силу своего небольшого размера очень просто анализируются, и спрятать защитный код там попросту негде. А вот если "размазать" защиту по нескольким мегабайтам прикладного кода, то на ее анализ уйдет чертова уйма времени, и если у хакера нет никаких особых стимулов для взлома (как-то: спортивный интерес, повышение собственного профессионализма и т. п.), то он скорее приобретет легальную версию, чем в течение нескольких недель будет метаться от дизассемблера к отладчику.

Какие же фокусы используют разработчики, чтобы затруднить анализ драйверов? Ну, вот, например: шифруют текстовую строку с символьным именем устройства, которое создает драйвер при своей загрузке. В результате, хакер точно знает, что защитный код открывает устройство "\\.\MyGoodDriver", но не может быстро установить: какому именно драйверу это имя соответствует.

Если же шифровка отсутствует, то задача решается простым контекстным поиском. Вот, например, захотелось нам узнать: какой именно драйвер создает устройство с именем MbMmDp32 — заходим при помощи файлового менеджера FAR в папку WINNT\System32\Drivers, нажимаем <ALT>+<F7> и в строку поиска вводим "MbMmDp32", не забыв установить флажок **Use all installed character tables** (в противном случае FAR ничего не найдет, т. к. строка должна задаваться в Unicode). Прошуршав некоторое время диском, FAR выдаст единственно правильный ответ: ASPI32.SYS. Это и есть тот самый драйвер, который нам нужен! А теперь представьте, что строка с именем зашифрована. Если драйвер загружается динамически, то это еще полбеды: просто ставим точку останова на функцию `IoCreareDevice` и ждем "всплытия" отладчика. Затем даем команду `P RET` и по карте загруженных моделей (выдаваемых командой `mod`) смотрим — кто "проживает" в данном регионе памяти. С драйверами, загружающимися вместе с самой операционной системой, справиться значительно сложнее, и, как правило, отыскивать нужный драйвер приходится методом "тыка". Часто в этом помогает дата создания файла, — драйвер, устанавливаемый защищенным приложением, обычно имеет ту же самую дату создания, что и остальные его файлы. Однако защитный механизм может свободно манипулировать датой создания по своему усмотрению, так что это не очень-то надежный прием. Хороший результат дает сравнение содержимого каталога WINNT\System32\Drivers до и после инсталляции защищенного приложения, — очевидно, защита может скрываться только среди вновь появившихся драйверов.

"Отжиг" дисков. За, против и немного вокруг

> Может ли Бог сотворить такой камень, который сам же и не сможет поднять — неизвестно, но вот программист запросто находит такое, что хрен потом отладит.
>
> *Программистский фольклор*

Для защиты дисков от копирования, приступать к созданию своей собственной программы "прожига" совершенно необязательно. Вместо этого вы можете манипулировать с "сырыми" образами дисков, поддерживаемыми программами Alcohol 120% или CloneCD. Несмотря на то, что все эти программы налагают на записываемые ими образы определенные ограничения, создание качественных защитных механизмов все-таки остается возможным. Другими словами, эти программы спокойно записывают то, что самостоятельно скопировать оказываются не в состоянии!

При создании собственного копировщика защищенных дисков без умения прожигать диски можно, в принципе, и обойтись, — достаточно лишь подготовить образ диска (т. е. корректно прочитать защищенный диск), ну, а тиражирование "хакнутого" образа уже не проблема. Лучше сфокусироваться непосредственно на анализе защищенных дисков, чем изобретать велосипед, в очередной раз разрабатывая то, что уже давно разработано до вас. Alcohol 120% и CloneCD имеют превосходные возможности "прожига", но вот читающий движок у них явно слабоват, и даже незначительные искажения служебных структур лазерного диска способны сбить их с толку.

Если же, несмотря ни на что, вы по прежнему убеждены, что свой собственный Нерон, сжигающий Рим, вам все-таки нужен, что ж! Добро пожаловать в гости к Демону Максвелла, попасть в лапы к которому гораздо сложнее, чем вырваться из них. Шутка! А вся доля правды в том, что техника "прожига" дисков — чрезвычайно обширный вопрос, даже краткое изложение которого потребовало бы отдельной книги. Одних лишь стандартов и спецификаций по SCSI-командам здесь окажется недостаточно, поскольку в них опущены многочисленные подробности процесса генерации различных структур данных, требующихся приводу для корректной записи исходного образа на лазерный диск. Лучшим из имеющихся пособий по "прожигу" дисков автор считает приложение *"Functional Requirements for CD-R (Informative)"* к документу *"SCSI-3 Multimedia Commands"*, электронную версию которого можно найти по адресу: **http://www.t10.org/ftp/t10/drafts/mmc/mmc-r10a.pdf** (обратите внимание, что в более поздних ревизиях документа это приложение было изъято).

Попробуйте также обратиться к исходным текстам утилиты *CDRTOOLS*, которые можно найти по адресу: **http://prdownloads.sourceforge.net/cdromtool/cdromtool_2002-11-26.zip?download**. Конечно, семь с небольшим мегабайт исходных текстов — не самое лучшее средство для вдохновения, но более простые программы "прожига" автору неизвестны.

Более трудоемким (но вместе с тем и соблазнительным!) способом является дизассемблирование исполняемых файлов программ Alcohol 120%, CloneCD, CDRWin и других, включая Nero Burning ROM. Собственно, полное дизассемблирование проводить совершенно необязательно, достаточно перехватить передаваемые приводы SCSI-команды и проанализировать последовательность их вызовов, не забывая при этом о значениях аргументов, в которых все ключевые структуры данных, собственно, и содержатся.

В зависимости от способа, выбранного разработчиком приложения для взаимодействия с устройством, "шпионаж" осуществляется либо перехватом функции `DeviceIoControl` с аргументами `IOCTL_SCSI_PASS_THROUGH`/`IOCTL_SCSI_PASS_THROUGH_DIRECT` (4D004h/4D014h), либо `SendASPI32Command` для интерфейсов SPTI и ASPI соответственно. Приложения, взаимодействующие

с приводом посредством своего собственного драйвера, также поддаются перехвату, но универсальных решений здесь нет, и каждый конкретный случай следует рассматривать индивидуально.

Давайте исследуем копировщик Alcohol 120% на предмет выявления алгоритма очистки и "прожига" CD-RW-дисков (CD-R-диски "прожигаются" аналогичным образом, но, по понятным причинам, не поддаются очистке). Итак, запускаем программу Alcohol 120%, переходим к вкладке *"Настройки"*, щелкаем по ссылке *"Общие"* и в ниспадающем боксе *"Интерфейс управления дисками"* выбираем ***"Интерфейс WinASPI Layer (безопасный режим)"***, если только он уже не был выбран ранее. После смены интерфейса программа Alcohol 120% потребует перезапуска программы, — что ж, выходим из нее и тут же запускаем вновь, убеждаясь в работоспособности последней.

Теперь вызываем отладчик Soft-Ice (или любой другой отладчик, поддерживающий точки останова на функции API) и, предварительно загрузив ASPI-экспорт в память (NuMega Symbol Loader → File → Load Exports → Wnaspi32.dll), открываем процесс Alcohol.exe, при необходимости распаковав (по обыкновению он упакован упаковщиком исполняемых файлов UPX).

Пытаемся установить точку останова на функции `SendASPI32Command`, выдавая отладчику следующую команду `bpx SendASPI32Command`, но ничего хорошего из этого у нас не получится, — отладчик Soft-Ice ругается, что не может найти такую функцию, несмотря на то, что ее имя написано без ошибок. Это не покажется удивительным, если предположить, что библиотека Wnaspi32.dll загружается динамически в ходе выполнения программы и на этапе загрузки Alcohol.exe адреса ASPI-функций еще не известны.

Можно поставить точку останова на функции `LoadLibraryA`, отслеживая загрузку всех динамических библиотек, но, поскольку программа Alcohol 120% загружает огромное количество разнообразных DLL, то на отладку уйдет чудовищное количество времени, в течение которого мы будем "тупо пялиться" на экран, монотонно нажимая комбинацию клавиш <CTRL>+<D> на клавиатуре. Более прогрессивным средством мониторинга будет установка *условной* точки останова, которая автоматически отсечет все заведомо ложные вызовы. Соответствующая ей команда может выглядеть, например, так: *`bpx LoadLibraryA IF *(ESP->4) == "SANW"`*, где `SANW` — это четыре первых символа имени "Wnaspi32.dll", записанные задом наперед с учетом регистра, выбранного разработчиком программы (если регистр наперед неизвестен, можно использовать функцию сравнения, не чувствительную к регистру).

Затем команда "`bpx GetProcAddress`" позволит перехватить загрузку всех ASPI-функций и `SendASPI32Command` в том числе. Имя загружаемой функции может быть просмотрено командой `d esp` → `4`. Дождавшись появления `SendASPI32Command`, выдаем команду `P RET` и, установив точку останова на "`BPX EAX`", нажимаем комбинацию клавиш <Ctrl>+<D> для выхода из отладчика Soft-Ice (все остальные точки останова при желании можно удалить).

По факту "всплытия" отладчика, наскоро набираем команду `d esp → 4`, и в окне дампа памяти появляется содержимое структуры `SRB_ExecSCSICmd`. Теперь 30h байт по счету — это первый байт CDB-пакета (внимание! это именно первый байт пакета, а не указатель на сам пакет); 03h и 10h байты — это флаги направления передачи данных и указатель на буфер обмена соответственно.

Далее приведены примеры "шпионских" протоколов, перехваченных в процессе очистки и "прожига" болванки CD-RW (листинг 5.2).

Листинг 5.2. Содержимое перехваченных CDB-блоков, посылаемых программой Alcohol 120% устройству при быстрой очистке лазерного диска

```
1E 00 00 00 01 00   ← PREVENT REMOVAL (ON) -----------+
51 00 00 00 00 00   ← READ DISK INFORMATION-------+   |
1E 00 00 00 00 00   ← PREVENT REMOVAL (OFF) ------|---+
BB 00 FF FF FF FF   ← SET SPEED ----------+       |   |
5A 00 2A 00 00 00   ← MODE SENSE -----+   |       |   |
BB 00 FF FF 02 C2   ← ----------------|---+       |   |
5A 00 2A 00 00 00   ← ----------------+           |   |
1E 00 00 00 00 00   ← ----------------------------|---+
51 00 00 00 00 00   ← ----------------------------+
A1 11 00 00 00 00   ← BLANK
```

Обратите внимание, что для очистки диска программа Alcohol 120% использует SCSI-команду `BLANK`, подробное описание которой содержится в документах *"Multimedia Commands — 4"* и *"Information Specification for ATAPI DVD Devices"*. Именно команда `BLANK`, а не `ERASE`, как пытается убедить нас товарищ Всеволод Несвижский в своей книге "Программирование устройств SCSI и IDE". Приведенные им листинги программ не работают, да и не должны работать в принципе. Команда `ERASE` (10), с кодом операции `2Ch`, местами упоминается как команда с кодом операции `1Ch`, соответствующим SCSI-команде RECEIVE DIAGNOSTIC RESULTS, кстати, не поддерживаемой оптическими накопителями вообще. Команды `ERASE` (12) не то чтобы совсем не существовало в природе, однако согласно приведенному автором коду операции — `ACh` — это есть не что иное, как команда `GET PERFORMANCE`. Интересно, как автор планировал что-либо стирать с ее помощью?

Ладно, оставим критику в сторону и продолжим нашу "шпионскую деятельность", наблюдая за процессом "прожига" лазерного диска. Последовательность SCSI-команд, посылаемых устройству, будет следующей.

Листинг 5.3. Содержимое перехваченных CDB-блоков, посылаемых программой Alcohol 120% устройству при "прожиге" образа лазерного диска

```
выбираем "прожиг" в меню
BB 00 FF FF FF FF   ← SET SPEED
5A 00 2A 00 00 00   ← MODE SENSE
AC 00 00 00 00 52   ← GET PERFORMANCE

появляется диалог "запись"
1E 00 00 00 00 01   ← PREVENT REMOVAL (LOCK)
51 00 00 00 00 00   ← READ DISK INFORMATION
1E 00 00 00 00 00   ← PREVENT REMOVAL (UNLOCK)

запись диска в прогрессе
43 02 04 00 00 00   ← READ ATIP
51 00 00 00 00 00   ← READ DISK INFORMATION
…
52 00 00 00 00 00   ← READ TRACK/ZONE INFORMATION
5A 00 05 00 00 00   ← MODE SENSE
55 10 00 00 00 00   ← MODE SELECT
51 00 00 00 00 00   ← READ DISK INFORMATION
2A 00 FF FF D2 AC   ← WRITE(10) -+
2A 00 00 00 D2 BC   ← -----------+-- write Lead-In
2A 00 00 00 D2 CC   ← -----------+
…
2A 00 00 00 65 B3   ← WRITE(10) -+
2A 00 00 00 65 CD   ← -----------+-- write track
2A 00 00 00 65 E7   ← -----------+
```

В заключение отметим перечень SCSI-команд, непосредственно относящихся к записи и рекомендованных для внимательнейшего изучения. Это: `BLANK`, `CLOSE TRACK/SESSION`, `FORMAT UNIT`, `READ BUFFER CAPACITY`, `READ DISC INFORMATION`, `READ MASTER CUE`, `READ TRACK INFORMATION`, `REPAIR TRACK`, `RESERVE TRACK`, `SEND CUE SHEET`, `SEND OPC INFORMATION`, `SYNCHRONIZE CACHE`, `WRITE` (10). Все вышеперечисленные команды относятся к Стандарту MMC-1 и потому максимально просты для понимания. Сам же текст стандарта можно найти по адресу: **http://www.t10.org/ftp/t10/drafts/mmc/mmc-r10a.pdf**.

Блокирование/разблокирование кнопки Eject

Если приложение, взаимодействующие с CD, выполняет операцию, которая не должна быть ни при каких обстоятельствах прервана, можно воспользоваться ICTL-командой блокировки лотка — `IOCTL_CDROM_MEDIA_REMOVAL` (а вот и ее непосредственное значение: `0x24804`). При попытке сделать диску "Eject" — при заблокированном лотке, мой PHILIPS CDRW начинает злобно моргать красным огоньком, показывая, что диск IN, но он "is locked". Вплоть до момента разблокирования лотка извлечь диск можно разве что булавкой или перезагрузив операционную систему.

Уже одно это создает богатое поле для всевозможных пакостей со стороны многочисленных злоумышленников, да и просто некорректно работающих программ, успевающих "умереть" от критической ошибки прежде, чем разблокировать лоток. Как с этим бороться? Да очень просто — разблокировать лоток самостоятельно!

Дело в том, что система не требует, чтобы разблокирование выполнялось в контексте того процесса, который выполнил блокирование. Она просто ведет счет количества блокировок, и если тот равен нулю, — лоток свободен. Соответственно, если счет блокировок равен, например, шести — мы должны шесть раз вызывать команду разблокирования, прежде чем лазерный диск удастся извлечь на свет божий.

Утилита, исходный текст которой приведен в листинге 5.4, позволяет манипулировать счетчиком блокировок диска по вашему собственному усмотрению. Аргумент командной строки "+" увеличивает значение счетчика на единицу, а "−" — уменьшает. При достижении счетчиком нуля дальнейшие попытки его уменьшения не возымеют никакого действия.

Как это можно использовать? Ну, например, для преждевременного извлечения диска из записывающей программы, что полезно для экспериментов. Другое применение: отлучаясь от своего компьютера на несколько минут, вы можете заблокировать диск, чтобы быть уверенными, что окружающие коллеги его не упрут. А если все-таки упрут (перезагрузив компьютер), заблокируйте лотки их приводов CD-ROM — пусть теперь перезагружаются!

Листинг 5.4. Утилита для блокирования/разблокирования лотка в приводе CD-ROM

```
/*---------------------------------------------------------------------------
 *
 *                    БЛОКИРУЕТ/РАЗБЛОКИРУЕТ ЛОТОК CD-ROM
 *                    ===================================
 *
```

```
 * build 0x001 @ 04.06.2003
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>

#define IOCTL_CDROM_MEDIA_REMOVAL 0x24804
main(int argc, char **argv)
{
      BOOL                        act;
      DWORD                       xxxx;
      HANDLE                      hCD;
      PREVENT_MEDIA_REMOVAL       pmrLockCDROM;

      // ПРОВЕРКА АРГУМЕНТОВ
      if (argc<3){printf("USAGE: CD.lock.exe \\\\.\\X: {+,-}\n"); return -1;}

      if (argv[2][0]=='+') act=TRUE;                // УВЕЛИЧИТЬ СЧЕТЧИК БЛОКИРОВОК
           else if (argv[2][0]=='-') act=FALSE;     // УМЕНЬШИТЬ СЧЕТЧИК БЛОКИРОВОК
                 else {printf(stderr,"-ERR: in arg %c\n",argv[2][0]); return -1;}

      // ПОЛУЧИТЬ ДЕСКРИПТОР УСТРОЙСТВА
      hCD=CreateFile(argv[1],GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,0,0);
      if (hCD == INVALID_HANDLE_VALUE) {printf("-ERR: get CD-ROM\n");return -1;}

      // ЗАБЛОКИРОВАТЬ/РАЗБЛОКИРОВАТЬ ЛОТОК CD-ROM'a
      pmrLockCDROM.PreventMediaRemoval = act;
      DeviceIoControl (hCD, IOCTL_CDROM_MEDIA_REMOVAL,
                 &pmrLockCDROM, sizeof(pmrLockCDROM), NULL, 0, &xxxx, NULL);
}
```

Хакерские секреты. Рецепты "тормозной жидкости" для CD

Появление высокоскоростных приводов CD-ROM породило огромное количество проблем, и, по общему мнению, пользователей плюсов здесь гораздо меньше, чем минусов. Это реактивный гул, вибрация, разорванные в клочья диски — скажите, на кой черт все это вам нужно? К тому же, многие из алгоритмов привязки к CD на высоких скоростях чувствуют себя крайне неустойчиво, и защищенный диск запускается далеко не с первого раза, если вообще запускается. Какой же из всего этого выход? Естественно — тормозить! Благо,

команду `SET CD SPEED` (опкод `0BBh`) большинство приводов все-таки поддерживает. Казалось бы, задал нужные параметры и вперед! Ан нет, — тут все не так просто.

Неприятность первая (маленькая, но зато досадная!). Скорость задается не в "иксах", а в килобайтах в секунду (именно в килобайтах, а не байтах!). Причем однократной скорости передачи соответствует пропускная способность в 176 Кбайт/с. А двукратной? Думаете, 176 × 2 = 352? А вот и нет — 353! Зато трехкратная скорость вычисляется в полном соответствии с привычной нам математикой: 176 × 3 = 528, но уже четырехкратная скорость опять отклоняется от "иксов": 176 × 4 = 704, против 706 по стандарту. Неправильно заданная скорость приводит к установке скорости, на ступень меньше ожидаемой, причем соответствие между "иксами" и ступенями далеко не однозначное. Допустим, привод поддерживает следующий ряд скоростей: 16x, 24x, 32x и 40x. Если заданная скорость (в килобайтах в секунду) не дотягивает до нормативной скорости 32 "икса", то привод переходит на ближайшую "снизу" поддерживаемую им скорость, т. е. в нашем случае 16x. Отсюда мораль: для перевода "иксов" в килобайты в секунду их нужно умножать не на 176, а на **177**!

Неприятность вторая (крупнее и досаднее). Команды, выдающей полный список поддерживаемых скоростей в стандартной спецификации, нет, и добывать эту информацию приходится исключительно методом перебора. Корректно работающая программа перед началом такого перебора должна убедиться в отсутствии носителя в приводе, а если он там есть — принудительно открыть лоток. Дело в том, что раскручивание некачественного CD-ROM-диска до высоких скоростей может привести к его разрыву и вытекающей отсюда порче самого привода. Пользователь должен быть абсолютно уверен в том, что установленный в привод диск будет вращаться именно с той скоростью, с которой его просят, и его программа не станет самопроизвольно увеличивать скорость без видимых на то причин.

Неприятность третья (или тихий ужас). Некоторые приводы (в частности TEAK 522E) успешно "заглатывают" команду `SET CD SPEED` и подтверждают факт изменения скорости, возвращая в `MODE SENSE` ее новое значение, однако физически скорость диска остается неизменной вплоть до тех пор, пока к нему не произойдет того или иного обращения. Поэтому, вслед за командой `SET CD SPEED`, недурно бы дать команду чтения сектора с диска, если, конечно, диск вообще присутствует. Изменять же скорость привода без диска в лотке — совершенно бессмысленная операция, пригодная разве что для построения ряда поддерживаемых скоростей, т. к. после вставки нового диска в привод прежние скоростные установки оказываются недействительными, и наиболее оптимальная (с точки зрения привода!) скорость для каждого диска определяется индивидуально. Также привод вправе изменять скорость диска по своему усмотрению, понижая ее, если чтение идет неважно, и, соответственно, увеличивая обороты, если же все идет хорошо.

Примеры исследования реальных программ

В качестве закрепления всего ранеесказанного и обретения минимальных практических навыков давайте исследуем несколько популярных программ, работающих с лазерными дисками на низком уровне, на предмет выяснения: как именно осуществляется такое взаимодействие.

Вызвав незаменимый отладчик Soft-Ice и установив точку останова на `bpx CreateFileA if (*esp->4=='\\\\.\\')`, мы будем последовательно запускать три следующих программы: *Alcohol 120%*, *Easy CD Creator* и *CloneCD*, каждый раз отмечая имя открываемого устройства. Итак, приступим.

Alcohol 120%

Программа Alcohol 120% в зависимости от настроек может обращаться к диску тремя путями: через *собственный драйвер* (по умолчанию), через *ASPI/SPTI-интерфейс* и через *ASPI Layer*. Начнем с собственного драйвера. Установка точки останова на функции `CreateFileA` показывает, что программа Alcohol 120% открывает устройство `\\.\SCSI2:` (естественно, на других компьютерах номер может быть и другим), и дальнейшая проверка подтверждает, что функция `DeviceIoControl` получает тот же самый дескриптор, что возвратился при открытии устройства SCSI! Следовательно, под "собственным" драйвером программа Alcohol 120% понимает тот самый драйвер мини-порта, которой он и установил в систему при своей установке.

Теперь изменим настройки программы Alcohol 120% так, чтобы она работала посредством интерфейса SPTI/ASPI. После перезапуска программы (а при смене метода доступа программа Alcohol 120% требует обязательного перезапуска), мы снова "словим" открытие устройства `\\.\SCSI2`, а затем произойдет открытие диска `\\.\G:` (естественно, на других компьютерах буква может быть и другой). Собственно, при взаимодействии с устройством через SPTI-интерфейс именно так все и происходит. Точнее, *должно происходить*. Программа Alcohol 120% открывает диск `\\.\G:` *многократно*, что указывает на "корявость" его архитектуры. Это существенно усложняет нашу задачу, поскольку мы вынуждены следить за всеми дескрипторами одновременно, и если упустить хотя бы один из них, реконструированный алгоритм работы программы окажется неверным (разве не интересно узнать, как именно Alcohol 120% осуществляет копирование защищенных дисков?).

Наконец, переключив программу Alcohol 120% на последний оставшийся способ взаимодействия с диском, мы получим следующий результат: `\\.\\SCSI2`, `\\.\MbMmDp32`, `\\.\G:`. Устройство с именем `MbMmDp32` и есть уже знакомый нам ASPI-драйвер. Правда, не совсем понятно, зачем программа Alcohol 120% явно открывает диск "`\\.\G:`", ведь ASPI-интерфейс этого не требует.

Easy CD Creator

Программа Easy CD Creator обращается к приводу непосредственно по его "родному" имени (в моем случае это CDR4_2K), а затем открывает устройство MbDlDp32, которое сам драйвер CDR4_2K, собственно, и регистрирует.

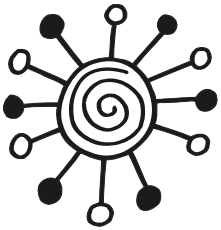
Следовательно, программа Easy CD Creator работает с диском через свой собственный драйвер, и, чтобы разобраться с ним, нам потребуется:

1. Дизассемблировать драйвер CDR4_2K и проанализировать, каким IOCTL-кодам какие действия драйвера соответствуют.
2. Отследить все вызовы функции `DeviceIoControl` (просто поставьте на нее условную точку останова, всплывающую при передаче "своего" дескриптора, возвращенного функцией `CreateFileA("\\\\.\\CRDR_2K", …)` и `CreateFileA("\\\\.\\MbDlDp32", …)`.

Оформив последовательность IOCTL-вызовов в виде импровизированной программы, мы сможем воссоздать протокол взаимодействия с диском и найти защиту (если она там есть).

CloneCD

Точка останова, установленная на функцию `CreateFileA`, показывает, что программа CloneCD общается с диском через свой собственный драйвер — `\\.\ELBYCDIO`, причем по не совсем понятным причинам его открытие происходит в цикле, так что дескриптор драйвера возвращается многократно.



Часть III

ЗАЩИТЫ ОТ КОПИРОВАНИЯ И СПОСОБЫ ИХ ПРЕОДОЛЕНИЯ

Глава 6. **Механизмы защиты**

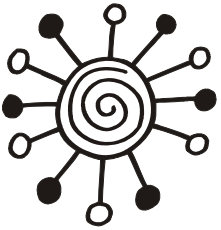
Глава 7. **Защиты, препятствующие проигрыванию диска в PC CD-ROM**

Глава 8. **Защиты от пофайлового копирования диска (защиты уровня файловой системы)**

Глава 9. **Защиты, основанные на привязке к носителю**

Глава 10. **Техника восстановления данных с лазерных дисков или практическое знакомство с сессиями**

Глава 6



Механизмы защиты

В этой главе рассматриваются различные механизмы защит от несанкционированного копирования CD с объяснением принципов их работы, примерами программной реализации и демонстрацией, как эти защиты могут быть нейтрализованы.

Классифицировать защиты от несанкционированного доступа можно по разным критериям (на редкость умная фраза, конечно, но надо же как-то начать).

Наиболее важными из них являются следующие:

- *стойкость к взлому* (копируется штатным копировщиком; копируется специализированным копировщиком или допускает эмуляцию защищенного носителя; не копируется в автоматическом режиме вообще);
- *принцип защиты* (нестандартная разметка диска; привязка к физическим характеристикам конкретного носителя);
- *степень совместимости с программно/аппаратной средой* (защитный механизм полностью соответствует стандарту и совместим со всем стандартным оборудованием; защитный механизм формально не нарушает стандарт, но закладывается на никем не гарантированные особенности реализации оборудования; защитный механизм явно нарушает стандарт, закладываясь на вполне определенный модельный ряд оборудования);
- *уровень реализации* (программный — создание мастер-диска осуществляет на штатном оборудовании; аппаратный — создание мастер-диска требует специального оборудования);
- *интерфейс взаимодействия с приводом* (стандартная библиотека языков Си и/или Паскаль; API операционной системы; низкоуровневый доступ к оборудованию);
- *объект защиты* (защита от копирования всего диска целиком, защита от пофайлового копирования, защита цифрового "грабежа" аудиоконтента).

Что касается стойкости к взлому, то абсолютной защиты от копирования оптических носителей не существует, да и не может существовать в принципе, поскольку, коль скоро диск можно прочитать, можно его и скопировать. Конечно, при наличии достойной аппаратной защиты, насильно вмонтированной в чипсет привода, процедура взлома рискует оказаться весьма непростой, а то и вовсе не реализуемой на штатном оборудовании. Но что помешает хакеру модифицировать прошивку своего привода или внести в него определенные конструктивные изменения, заблокировав защиту? Достаточно вспомнить нашумевшую историю с чипами MOD, чтобы все иллюзии держателей авторских прав рассеялись, словно дым.

Бороться с профессиональными взломщиками — абсолютно бессмысленно. Этот бой еще не выигрывал никто. Чем стремительнее совершенствуются защитные механизмы, тем соблазнительнее выглядит их взлом. Даже самые совершенные защиты ломаются — это только вопрос времени, стимула и вложенных во взлом денег (а финансовый потенциал пиратов поистине безграничен — имейте в виду).

Поэтому защищаться следует не от хакеров, а от квалифицированных пользователей. По минимуму, защищенный диск не должен копироваться никакими штатными копировщиками (Ahead Nero, Roxio Easy CD Creator), а по максимуму — и специализированными копировщиками защищенных дисков тоже (Alcohol 120%, CloneCD). Впрочем, некопируемость бывает очень разной. Физические дефекты поверхности на бытовом оборудовании не копируются в принципе, но легко *имитируются* искажением контрольной суммы сектора, которая, впрочем, умными защитами элементарно распознается (правда, для этого защита должна спуститься с уровня API на несколько ступень вглубь, получив прямой доступ к железу, что не есть хорошо в плане конфликтности и безопасности).

Еще сложнее справиться с *эмуляцией* оригинального носителя. Некоторые продвинутые копировщики (например, Alcohol 120%) создают виртуальный привод, ведущий себя точь-в-точь как защищенный диск, старательно воспроизводя все, ну, или практически все, физические характеристики поверхности. Отсюда: трудоемкость копирования оригинального носителя определяется отнюдь не сложностью имитации тех или иных ключевых особенностей, а скрытностью их размещения.

Другими словами, необходимо найти такой набор отличительных признаков, факт присутствия которого было бы чрезвычайно трудно обнаружить. Хорошим кандидатом на эту роль выглядят каналы подкода нечитаемых секторов — в силу конструктивных особенностей оптических приводов, точность позиционирования на субканальные данные невелика и результат SCSI/ATAPI-команды `READ SUBCHANNEL` не только непредсказуем, но еще и не воспроизводим! При каждом ее выполнении она возвращает субканальные данные сектора $N \pm \delta + k$, где N — адрес запрошенного сектора, δ — случайная, а k — систематическая погрешность привода. Таким образом,

на "грабеж" субканальных данных всех секторов уйдет очень много времени, а, если субканальные данные умышленно перепутаны и/или искажены, задача их копирования становится вообще нереальной. Скопировать такой диск практически невозможно — ни существующими, ни последующими копировщиками.

Стойкость защиты к bit-hack'у в общем-то не критична. Какой бы "крутой" она не была, ее все равно взломают — был бы стимул! Поэтому речь идет лишь о *затруднении* копирования оригинальных носителей штатными или хакерскими средствами. А как можно затруднить копирование?

Во времена господства MS-DOS и 3,5"/5,25" накопителей на гибких магнитных дисках (или приводов floppy) широко использовались такие приемы защиты, как *нестандартная разметка диска* и *создание трудновоспроизводимых дефектов диска*, которые могли быть реализованы как на *аппаратном*, так и на *программном* уровне. Под "аппаратным" уровнем в данном контексте понимается нестандартное оборудование, используемое для записи защищенного диска (например, устройство, формирующее лазерные метки, путем испарения магнитного покрытия в строго определенных местах или же банальный конденсатор переменной емкости (КПЕ), подключенный параллельно к кварцу для изменения его частоты, а значит, и длины дорожки). "Программные" же методики защиты ограничивались лишь штатным оборудованием, значительно снижая накладные расходы на тиражирование оригинальных дисков (при небольших партиях это было весьма актуально). Самое забавное, что стойкость аппаратных защит была ничуть не выше программных и все они ломались программными методами, зачастую в полностью автоматическом режиме.

Вообще говоря, все существующие методики защиты лазерных дисков можно разделить на два типа:

- *нестандартная разметка диска*;
- *привязка к физическим характеристикам конкретного носителя.*

Защиты первого типа живут за счет нарушения стандартов, в то время как подавляющее большинство "честных" программ эти самые стандарты стараются соблюдать. В результате, защищенный диск не может быть скопирован штатным способом, на что держатели авторских прав, собственно, и рассчитывают. Идея нестандартных разметок не нова и широко использовалась еще во времена "древних" компьютеров, таких как "Амига", "Спектурм" и прочих. Устоять против "человека с мозгами" такая защита все равно не могла, а вот проблемы, которые она вызывала у легальных пользователей, уже давно вошли в анналы. Любое, даже самое незначительное отступление от стандартов, лишает вас всяких гарантий того, что диск вообще будет читаться! Большое количество разнородного оборудования, присутствующего на рынке, не позволяет протестировать защищенные диски на всех существующих моделях приводов, а значит, есть риск, что обладатели

непротестированных моделей столкнутся с серьезными проблемами. И с ростом тяжести нарушения стандарта, этот риск многократно усиливается.

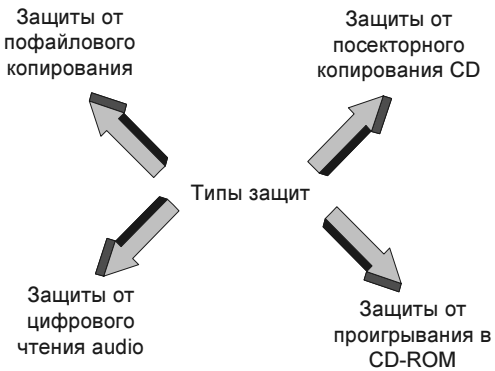


Рис. 6.1. Классификация защитных механизмов

Защита диска от проигрывания в компьютерных приводах CD-ROM, также называемая защитой от *цифрового воспроизведения*, — самый гнусный тип защиты из всех защитных механизмов вообще. Основная цель защиты — предотвратить несанкционированное копирование диска и его "грабеж" в MP3, в то же самое время никак не препятствуя его нормальному проигрыванию. Очевидно, что требования, предъявляемые к защите, взаимоисключающие, поскольку воспроизведение аудиодиска в общем случае его "грабеж" и есть. Куда подаются "сграбленные" данные (на вход ЦАП или MP3-компрессора), это уже не суть важно, т. к. провод не сообщает диску, через какие электронные цепи он его читает (со своим завтраком не разговаривают!)

Правда, на практике все обстоит не совсем так. Бытовые аудиопроигрыватели и компьютерные приводы CD-ROM имеют множество конструктивных различий и по-разному интерпретируют одну и ту же информацию, записанную на CD. Да и сами компьютерные приводы CD-ROM чаще всего воспроизводят аудиодиски через специальный аудиотракт, природа которого заметно отличается от цифрового канала чтения данных. Таким образом, создание защитного механизма сводится к преднамеренному внесению в структуру диска таких искажений, которые бы проявлялись лишь в режиме цифрового "грабежа" данных, а во всех остальных случаях оставались незамеченными.

Как уже говорилось ранее, любые искажения структуры диска, выходящие за рамки стандарта, делают диск нестандартным, а поведение нестандартных дисков на произвольном оборудовании — непредсказуемо! Причем обработка нестандартных аудиодисков значительно отличается от обработки нестандартных дисков с данными. Защитный механизм, помещенный в исполняемый файл и, по сути своей, являющийся программой, знает все об

искажениях формата защищенного диска и знает, как его следует обрабатывать. От читающего провода требуется лишь одно — не мешать и делать то, и только то, что ему скажут. Аудиодиск, обрабатывающийся микропрограммой самого привода, — это совсем другое дело, ибо представляет не исполняемый код, а данные, и эти данные приходится обрабатывать *внешней* программе ("прошивке" привода), которая спроектирована в соответствии со стандартом и *любые* отклонения от данного вправе трактовать как "non-audio disk or no disk present". Огромное количество "разношерстной" цифровой техники чрезвычайно затрудняет задачу тестирования защитных механизмов на совместимость.

Фактически защита от копирования аудиодисков превращается в защиту от их воспроизведения. Зачастую такие диски отказываются работать даже на обыкновенных аудиоплеерах, не говоря уже про компьютерные приводы CD-ROM, на которых если они и воспроизводятся, то только через аудиотракт, который мало-помалу уже начинает отмирать. Так, операционные системы Windows 2000 или Windows XP вовсю используют цифровое воспроизведение аудиодисков, а на Mac'ах это, судя по слухам, и вовсе их основной режим. Не собираясь рассуждать о достоинствах цифрового воспроизведения перед аналоговым (это тема совсем другой книги), отметим лишь то, что каждый пользователь вправе сам выбирать наиболее предпочтительный способ прослушивания. А потому, взлом таких защит не то что незаконное, но даже благородное дело!

Компания PHILPS (один из изобретателей технологии CD) открыто выступает против любых нарушений стандарта и запрещает маркировать защищенные диски логотипом "CD". Законодательство многих стран (в том числе и России) придерживается такого же мнения. Защищенный нестандартным форматом диск должен содержать недвусмысленное предупреждение, что приобретаемый вами кусок пластика только с виду похож на компакт-диск, но в действительности таковым *не является*.

Ностальгия

"Во времена Спектрума с дисководом одна фирма, которая выпускала компьютерный журнал (не могу вспомнить название за давностью событий), придумала вот что — на микросхему контроллера дисковода подается определенная тактовая частота, исходя из которой он и оперирует. А поскольку метод записи там MFM, т. е. FM, то от частоты зависит длина дорожки. Обычно (если я правильно помню) длина дорожки была около 6200 байт (это неформатированная длина). Так вот — эта фирма (наверное, скорее — группа товарищей) делала на диске дорожку длиной, например, 5000 байт (т. е. меньше). За счет ФАПЧ в контроллере она читалась вполне нормально, но длина в 5000 байт сохранялась.

Так вот — такую защиту скопировать было ***невозможно*** *на стандартном компьютере (это понятно). Но я как раз-то писал именно копировщик таких защищенных дисков… И придумал соответственно на время копирования такой — дорожки подключать параллельно кварцу в контроллере КПЕ, крутя который, следовало добиваться нужной длины (на экране все отображалось)"*, — Bob Johnson.

Встроенная защита CD-дисков

Защита от несанкционированного копирования была заложена в лазерные диски изначально. Даже в те незапамятные времена, когда еще не существовало ни персональных компьютеров, ни лазерных дисков с данными, ни самой проблемы пиратства, "Красная книга" уже описывала специальный бит digital copy prohibited/permitted, запрещающий цифровое чтение с носителя, если держателю авторских прав было так угодно (см. поле CONTROL Q-канала подкода, если первый, считая от нуля, бит сброшен, то цифровое чтение разрешено и, соответственно, наоборот). Однако ни один их известных автору копировщиков этот бит не учитывает (Ahead Nero Burning ROM выдает предостерегающее предупреждение, но не более того), и ни один из приводов не блокирует цифровое чтение, даже если оно и "запрещено". Отчасти это вызвано тем, что цифровое воспроизведение имеет значительные преимущества перед аналоговым (меньше посторонних шумов, программная коррекция звука и т. д.), чем и объясняется стремительное вытеснение последнего с рынка. Устройству, не поддерживающему цифровое воспроизведение, будет очень трудно найти своего покупателя, вот производителям и приходится игнорировать стандарт в угоду интересам пользователя.

Защиты, основанные на нестандартных форматах диска

Искажение TOC и его последствия

Искажение TOC — жестокий, уродливый, но на удивление широко распространенный прием, использующийся в доброй половине защитных механизмов. Штатные копировщики (Easy CD Creator, Stomp Record Now, Ahead Nero Burning ROM) на таких дисках в буквальном смысле слова "сходят с ума" и "едут крышей". Копировщики защищенных дисков (CloneCD, Alcohol 120%) к искаженному TOC относятся гораздо лояльнее, но требуют для своей работы определенного сочетания пишущего и читающего приводов, да и в этом случае копируют такой диск не всегда.

Пишущий привод обязательно должен поддерживать режим *RAW DAO (Disc At Once)*, — в котором весь диск записывается за один проход лазера. Режим *RAW SAO (Session At Once)* для этих целей совершенно непригоден, поскольку предписывает приводу писать сначала содержимое сессии, а потом — TOC. Как следствие — приводу приходится самостоятельно анализировать TOC, чтобы определить стартовый адрес сессии и ее длину. Попытка записать искаженный TOC в режиме RAW SAO в общем случае приводит к непредсказуемому поведению привода, и работоспособной копии защищенного диска нечего и думать! Первая, встретившаяся приводу, сессия с искаженным TOC

обычно оказывается и последней, т. к. остальные сессии писать уже *некуда* (искажение TOC обычно преследует цель увеличения размера сессии до нескольких гигабайт).

Читающий привод помимо режима "сырого" чтения (который поддерживают практически все приводы) должен уметь распознавать искаженный TOC, автоматически переходя в этом случае на использование "резервного" средства адресации — Q-канала подкода. В противном случае, сессия, содержащая искаженный TOC, окажется недоступной для чтения даже на секторном уровне.

Таким образом, *копирование дисков с искаженным TOC осуществимо не на всяком оборудовании* и порядка 1/3 моделей пишущих приводов для этих целей непригодны. Узнать: поддерживает ли выбранная вами модель привода режим RAW DAO или нет можно, в частности, из раздела *"Tech support"* справочной системы по программе CloneCD, где перечислены характеристики достаточно большого количества всевозможных приводов (впрочем, моих приводов там, увы, нет). Другой путь — "скормить" приводу SCSI/ATAPI команду 46h (`GET CONFIGURATION`) и посмотреть, что он ответит. Из двух моих пишущих приводов режим RAW DAO поддерживает один лишь NEC. С определением возможности чтения искаженных сессий дела обстоят на порядок сложнее, ибо данная особенность поведения является исключительно внутренней характеристикой привода и не афишируется ни самим приводом, ни его производителями. Приходится выяснять эту информацию экспериментально. Возьмите диск с искаженным TOC (о том, как его создать — рассказано далее в этой главе), вставьте его в привод и попробуйте прочесть несколько секторов из искаженной сессии. Реакция приводов может быть самой разнообразной. Тот же PHILIPS, в зависимости от "настроения" своих электронных цепей, то рапортует об ошибке чтения, то возвращает совершенно бессмысленный "мусор", в котором не угадывается даже синхропоследовательность, возглавляющая заголовок "сырого" сектора.

Основной недостаток защитных механизмов с искаженным TOC состоит в том, что некоторые приводы такие диски просто не "видят" и потому не могут их воспроизвести. Легальный пользователь, испытавший несовместимость защиты со своей аппаратурой, в лучшем случае обложит ее разработчика матом и поспешит вернуть диск продавцу. Если, конечно, сможет вытащить эту "бяку" из недр CD-ROM'а, что вовсе не факт, поскольку микропроцессорная начинка некоторых приводов при попытке анализа искаженного TOC просто "зависает" и привод полностью абстрагируется от всех раздражителей внешнего мира, не реагируя в том числе и на настойчивые попытки пользователя сделать диску Eject. Отверстие для аварийного выброса диска, правда, еще никто не отменял, но, по слухам, не везде оно есть (хотя лично мне такие приводы еще не встречались), а там где есть — зачастую оказывается скрытым за декоративной панелью или, что более вероятно, пользователь может вообще не знать, что это за отверстие такое, для

чего оно предназначено и как им, собственно, следует пользоваться. Посмотрите внимательно на лицевую панель своего привода CD-ROM, видите, — внизу лотка расположено крохотное отверстие около 1 мм в диаметре? Воспользовавшись любым длинным, тонким и достаточно прочным предметом, например, металлической канцелярской скрепкой, слегка приоткройте лоток, введя "отмычку" в указанное отверстие до упора и еще чуть-чуть надавив. Все! — дальше лоток можно выдвинуть уже руками.

Внимание

Во-первых проделывайте эту операцию только при выключенном компьютере, а во-вторых, держите "отмычку" строго горизонтально, иначе вы можете промазать и угодить в какой-нибудь нежный узел, основательно его повредив.

На "Макинтошах" (Macintosh) таких отверстий нет — это точно (или же пользователи этих компьютеров сплошь идиоты). Во всяком случае, количество судебных исков, поданных последними, в буквальном смысле слова не поддается ни разуму, ни исчислению. Самое интересное, что подавляющее большинство этих исков были удовлетворены и разработчикам пришлось оплатить и "ремонт" аппаратуры, и моральный ущерб, и, собственно, сами судебные издержки. *(Между нами говоря, снятие защиты с дисков, записанных с грубыми нарушениями стандарта, коими в частности и являются диски с искаженным TOC, не считается взломом и не преследуется по закону, поэтому: ломайте, ломайте и еще раз ломайте.)*

Некорректный стартовый адрес трека

Для создания защищенного диска с искаженным TOC нам понадобится: любая программа записи на диск, умеющая создавать многосессионные диски (например, Roxio Easy CD Creator), копировщик защищенных дисков, сохраняющий содержимое TOC в текстовом файле, доступном для редактирования (мы выбираем CloneCD), и, естественно, сам пишущий привод, поддерживающий режим "сырой" записи в режиме DAO. Для облегчения восприятия материала все действия будут расписаны по шагам, хотя это выглядит и не слишком литературно.

Шаг первый. Создание оригинального диска. Достаем из упаковки новую болванку CD-R или, что даже лучше, вставляем в привод "потертый жизнью" диск CD-RW и записываем на него пару сессий в штатном режиме. Будет лучше (вернее нагляднее), если вторая сессия будет включать в себя файлы первой сессии, той самой сессии, чей TOC мы и собираемся искажать. Интересно, сможет ли привод прочесть ее содержимое или нет?

Шаг второй. Получение образа оригинального диска. Запускаем программу CloneCD и указываем ей создать образ оригинального диска (выбираемый профиль настроек на данном этапе некритичен, поскольку диск еще не защищен, то с равным успехом можно использовать как *"CD с данными"*, так

и *"Protected PC Game"*; флажок *"создавать Cue-Sheet"* устанавливать необязательно — все равно он действителен лишь на односессионных компакт-дисках).

Шаг третий. Искажение стартового адреса первого трека в образе. Если все сделано правильно и программно/аппаратное обеспечение во всей своей совокупности работает нормально, на жестком диске должны образоваться три файла: IMAGE.CCD, — несущий в себе содержимое Q-канала подкода области Lead-In или, попросту говоря, TOC; IMAGE.IMG — "сырой" образ диска со всеми секторами от 00:00:02 до "сколько на диске есть там" и IMAGE.SUB — содержимое полей подкода "программной" части диска. Последний файл в принципе может и отсутствовать (он создается, только если установлен флажок *"Чтение субканалов из треков с данными"*), но это некритично, т. к. сейчас нас в первую очередь интересуют не каналы подкода, а сам TOC! Откроем файл IMAGE.CCD в любом текстовом редакторе и попытаемся перевести расклад геометрии диска на человеческий язык (листинг 6.1). Обобщенно говоря, диск содержит две секции, — по одному треку каждая. Абсолютный адрес начала первого трека — 00:00:02, абсолютный адрес Lead-Out области первой сессии — 00:29:33 (адрес последнего сектора трека на две секунды короче), абсолютный адрес начала второго трека — 03:01:33, а абсолютный адрес Lead-Out второй сессии — 03:24:33. Максимально достижимая емкость диска — 22:14:34 (хотя на самом диске и написано, что он 23-минутный).

Листинг 6.1. Содержимое неискаженного TOC в "сыром" виде

```
[CloneCD]        ; данные о CloneCD
Version=3        ; версия CloneCD. Идет лесом

[Disc]                  ; данные диска
TocEntries=12           ; кол-во элементов TOC'a
Sessions=2              ; кол-во сессий = 2
DataTracksScrambled=0   ; поле DVD (см. inf-8090),
                        ; для CD эта информация лишена смысла
CDTextLength=0          ; CD-Text'a в полях подкода Lead-In области нет

[Session 1]        ; данные сессии 1
PreGapMode=1       ; тип трека Mode 1(трек с данными, 2048 байт данных)
PreGapSubC=0       ; данных подканала - нет

[Session 2]        ; данные сессии 2
PreGapMode=1       ; тип трека Mode 1(трек с данными, 2048 байт данных)
PreGapSubC=0       ; данных подканала - нет
```

```
[Entry 0]          ; данные элемента TOC'a №0
Session=1          ; элемент сессии 1
Point=0xa0         ; номер первого трека сессии 1 в PMin/тип диска в PSec
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=1             ; номер первого трека сессии 1
PSec=0             ; тип диска CD-DA и CD-ROM диск в Mode 1
PFrame=0           ; не несет никакой полезной информации
PLBA=4350          ; номер трека, представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 1]          ; данные элемента TOC'a №1
Session=1          ; элемент сессии 1
Point=0xa1         ; номер последнего трека сессии 1 в PMin
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=1             ; номер последнего трека сессии 1 (в сессии только один трек)
PSec=0             ; не несет никакой полезной информации
PFrame=0           ; не несет никакой полезной информации
PLBA=4350          ; номер трека, представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 2]          ; данные элемента TOC'a №2
Session=1          ; элемент сессии 1
Point=0xa2         ; положение Lead-out области в PMin:PSec:PFrame
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
```

```
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=0             ; \
PSec=29            ;  + - абсолютный адрес Lead-out области сессии 1
PFrame=33          ; /
PLBA=2058          ; LBA-адрес Lead-out области сессии 1

[Entry 3]          ; данные элемента TOC'a №3
Session=1          ; элемент сессии 1
Point=0x01         ; данные трека 1 сессии 1
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=0             ; \
PSec=2             ;  + - абсолютный адрес начала трека 1 сессии 1
PFrame=0           ; /
PLBA=0             ; LBA-адрес начала трека 1 сессии 1

[Entry 4]          ; данные элемента TOC'a №4
Session=1          ; элемент сессии 1
Point=0xb0         ; позиция следующей записываемой области в AMin:ASec:AFrame
ADR=0x05           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=2             ; \
ASec=59            ;  + - абсолютный адрес следующей записываемой области
AFrame=33          ; /
ALBA=13308         ; LBA-адрес следующей записываемой области
Zero=3             ; кол-во pointer'ов в Mode 5
PMin=22            ; \
PSec=14            ;  + - абсолютный адрес максимальной записываемой области
PFrame=34          ; /
PLBA=99934         ; LBA-адрес максимальной записываемой области

[Entry 5]          ; данные элемента TOC'a №5
Session=1          ; элемент сессии 1
Point=0xc0         ; стартовый адрес Lead-In области Hybrid диска (если он есть)
```

```
ADR=0x05          ; Mode 5 (Оранжевая книга)
Control=0x04      ; диск с данными, запрещенный ;-) для копирования
TrackNo=0         ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=162          ; рекомендуемая мощность лазера для записи
ASec=128          ; Application code
AFrame=140        ; зарезервировано
ALBA=288590       ; LBA-"адрес" трех предыдущих полей
Zero=0            ; зарезервировано
PMin=97           ; \
PSec=27           ;  + - абсолютный адрес Lead-In области Hybrid диска
PFrame=21         ; /    (адрес лежит за пределами диска, т.е. Hybrid-диска нет)
PLBA=-11604       ; LBA-адрес Lead-In области Hybrid'а(вычислен с переполнением)

[Entry 6]         ; данные элемента TOC'а №6
Session=1         ; элемент сессии 1
Point=0xc1        ; копия ATIP-информации
ADR=0x05          ; -+
Control=0x04      ; -+
TrackNo=0         ; -+
AMin=4            ; -+
ASec=120          ; -+
AFrame=96         ; -+
ALBA=26946        ; -+ - ATIP-информация
Zero=0            ; -+
PMin=0            ; -+
PSec=0            ; -+
PFrame=0          ; -+
PLBA=-150         ; -+

[Entry 7]         ; данные элемента TOC'а №7
Session=2         ; элемент сессии 2 (вот мы и добрались до сессии 2!)
Point=0xa0        ; номер первого трека сессии 2 в PMin/тип диска в PSec
ADR=0x01          ; q-Mode == 1
Control=0x04      ; диск с данными, запрещенный ;-) для копирования
TrackNo=0         ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0            ; \
ASec=0            ;  + - абсолютный адрес текущего трека
AFrame=0          ; /
ALBA=-150         ; LBA-адрес текущего трека
Zero=0            ; это поле должно быть равно нулю, как оно и есть
PMin=2            ; номер первого трека сессии 2 (нумерация треков сквозная!)
PSec=0            ; тип диска CD-DA и CD-ROM диск в Mode 1
```

```
PFrame=0            ; не несет никакой полезной информации
PLBA=8850           ; номер трека, представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 8]           ; данные элемента TOC'a №8
Session=2           ; элемент сессии 2
Point=0xa1          ; номер последнего трека сессии 2 в PMin
ADR=0x01            ; q-Mode == 1
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0              ; \
ASec=0              ;  + - абсолютный адрес текущего трека
AFrame=0            ; /
ALBA=-150           ; LBA-адрес текущего трека
Zero=0              ; это поле должно быть равно нулю, как оно и есть
PMin=2              ; номер последнего трека сессии 2 (в сессии только один трек)
PSec=0              ; не несет никакой полезной информации
PFrame=0            ; не несет никакой полезной информации
PLBA=8850           ; номер трека, представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 9]           ; данные элемента TOC'a №9
Session=2           ; элемент сессии 2
Point=0xa2          ; положение Lead-out области в PMin:PSec:PFrame
ADR=0x01            ; q-Mode == 1
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0              ; \
ASec=0              ;  + - абсолютный адрес текущего трека
AFrame=0            ; /
ALBA=-150           ; LBA-адрес текущего трека
Zero=0              ; это поле должно быть равно нулю, как оно и есть
PMin=3              ; \
PSec=24             ;  + - абсолютный адрес Lead-out области сессии 2
PFrame=23           ; /
PLBA=15173          ; LBA-адрес Lead-out области сессии 2

[Entry 10]          ; данные элемента TOC'a №10
Session=2           ; элемент сессии 2
Point=0x02          ; данные трека 2 сессии 2
ADR=0x01            ; q-Mode == 1
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
```

```
AMin=0              ; \
ASec=0              ;  + - абсолютный адрес текущего трека
AFrame=0            ; /
ALBA=-150           ; LBA-адрес текущего трека
Zero=0              ; это поле должно быть равно нулю, как оно и есть
PMin=3              ; \
PSec=1              ;  + - абсолютный адрес начала трека 2 сессии 2
PFrame=33           ; /
PLBA=13458          ; LBA-адрес начала трека 2 сессии 2

[Entry 11]          ; данные элемента TOC'a №11
Session=2           ; элемент сессии 2
Point=0xb0          ; адрес следующей записываемой области в AMin:ASec:AFrame
ADR=0x05            ; Mode 5
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=4              ; \
ASec=54             ;  + - абсолютный адрес следующей записываемой области
AFrame=23           ; /
ALBA=21923          ; LBA-адрес следующей записываемой области
Zero=1              ; кол-во pointer'ов Mode 5
PMin=22             ; \
PSec=14             ;  + - абсолютный адрес последней возможной Lead-out области
PFrame=34           ; / (на самом диске написано 23 мин,
                    ; это ж как надо округлять 22:14:34)
PLBA=99934          ; LBA-адрес последней возможной Lead-out области

[TRACK 1]           ; данные трека 1
MODE=1              ; режим Mode 1
INDEX 1=0           ; post-gap?

[TRACK 2]           ; данные трека 2
MODE=1              ; режим Mode 1
INDEX 1=0           ; post-gap?
```

Давайте теперь немного "поиздеваемся" над TOC и увеличим стартовый адрес первого трека так, чтобы он вышел далеко за пределы первой сессии и попал... ну, собственно, куда-нибудь он все равно попадет. Чтобы быстро отыскать соответствующую ему запись, воспользуемся контекстным поиском. Жмем клавишу <F7> и вводим `point=0x1` (листинг 6.2).

Листинг 6.2. Атрибуты трека 1

```
[Entry 3]          ; данные элемента TOC'a №3
Session=1          ; элемент сессии 1
Point=0x01         ; данные трека 1 сессии 1
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=0             ; \
PSec=2             ;  + - абсолютный адрес начала трека 1 сессии 1
PFrame=0           ; /
PLBA=0             ; LBA-адрес начала трека 1 сессии 1
```

Как мы видим, здесь присутствует как *абсолютный*, измеряемый в минутах: секундах: фреймах, так и *LBA-адрес* трека, представляющий собой не что иное, как порядковый номер сектора, считая от нуля. На самом деле, LBA-адрес — это "отсебятина", добавляемая в файл самим CloneCD, и в TOC он не хранится. Судя по всему, CloneCD вычисляет LBA-адрес исходя из соображений удобства (работать с LBA-адресацией, действительно, намного комфортнее). Однако при внесении каких-либо изменений в CCD-файл, за согласованием обоих типов адресов нам придется следить самостоятельно. Для перевода абсолютных адресов в LBA можно воспользоваться следующей формулой:

```
Logical Sector Address = (((Minute * 60) + Seconds) * 75 +Frame) - 150.
```

Далее в листинге 6.3 представлен вид атрибутов трека 1 до и после искажения.

Листинг 6.3. Атрибуты трека 1 до искажений (слева) и после искажения (справа)

```
[Entry 3]                    [Entry 3]
Session=1                    Session=1
Point=0x01                   Point=0x01
ADR=0x01                     ADR=0x01
Control=0x04                 Control=0x04
TrackNo=0                    TrackNo=0
AMin=0                       AMin=0
ASec=0                       ASec=0
AFrame=0                     AFrame=0
```

ALBA=-150		ALBA=-150
Zero=0		Zero=0
PMin=0	→	**PMin=10**
PSec=2	→	**PSec=2**
PFrame=0	→	**PFrame=0**
PLBA=0	→	**PLBA=-1**

На самом деле коварный автор схитрил и вместо вычислений LBA-адреса имел в виду тот факт, что его версия CloneCD всегда использует абсолютные адреса, а LBA — игнорирует. Выбор абсолютного адреса первого трека — произвольный, но осуществленный с таким расчетом, чтобы искаженный адрес гарантированно "вылетал" за границы первой сессии, область Lead-Out которой находится по адресу 00:29:33 (см. поля `PMin:PSec:PFrame` в секции `[Entry 2]` листинга 6.1).

Шаг четвертый. Монтирование искаженного образа на виртуальный привод. Теперь смонтируем искаженный образ диска на виртуальный привод, создаваемый программой Alcohol 120%, и посмотрим, что из этого получилось. Конечно, нет никакой уверенности в том, что виртуальный привод поведет себя как настоящий, но ведь и настоящие приводы на искаженных дисках ведут себя по-разному! Поэтому использовать Alcohol 120% в качестве рабочего "макетника" вполне допустимо, тем более что это экономит уйму времени и болванок, ведь монтирование виртуального диска в отличие от "прожига" болванки осуществляется мгновенно, если, конечно, оно вообще осуществляется. Вплоть до версии 1.4.3 включительно — самой свежей версии на момент написания этих строк — программа Alcohol 120% органически "не переваривала" искаженные образы дисков и отказывалась их монтировать, апеллируя к недоступности образа файла: *"Unable to mount image. File not accessible"*. Судя по всему, Alcohol 120% понимает искаженный TOC слишком буквально, пытаясь отыскать в файле-образе то, что там заведомо нет (трека, начинающегося с адреса 10:02:00 и заканчивающегося адресом 00:29:33, там точно нет!).

Какая жалость! Возможность монтирования дисковых образов с искаженным TOC позволила бы нам преодолевать защиту от копирования на любых пишущих приводах, а не только тех, что поддерживают режим RAW DAO, — просто "сбрасываем" образ защищенного диска на болванку в виде обыкновенного файла и динамически монтируем его программой Alcohol 120% по мере необходимости. Выходит, что на проверку эта программа оказывается гораздо менее "крута", чем это кажется!

Шаг пятый. Запись искаженного образа на диск. В порядке эксперимента попробуем "прожечь" искаженный образ в режиме RAW SAO, в котором, как уже было сказано ранее, корректная запись сессий с искаженным TOC невозможна. Для гарантированного исключения возможных побочных эффектов желательно использовать привод, не поддерживающий режим

RAW DAO чисто физически (ну мало ли, вдруг копировщик в плане проявления чудес искусственного интеллекта автоматически перейдет на более подходящий режим записи, игнорируя наши установки).

Мастер записи образов копировщика Alcohol 120% выдает следующую информацию о записываемом образе (листинг 6.4). Обратите внимание на размеры и адрес первого трека первой сессии (они выделены полужирным шрифтом).

Листинг 6.4. Сводная информация по записываемому образу, выдаваемая Alcohol 120%

```
Тип:      Файл-образ CloneCD
Путь:     L:\
Имя:      Image.ccd
          Image.img
          Image.sub
Размер:   8.81 MB
Сессий:   2
Треков:   2

Сессия 01:
  Трек 01: Mode 1, Длина: -42942(8191.92 GB), Адрес: 045000
Сессия 02:
  Трек 02: Mode 1, Длина: 001715(3.3 MB), Адрес: 013458
```

Вот это номер! Если верить программе Alcohol 120%, то длина первой трека составляет целых 8 Тбайт. Этот чудовищный объем не то что на CD-, даже на DVD-диск не залезет! На самом деле, длина треков в TOC нигде явным образом не хранится, но вычисляется как разница стартовых адресов двух смежных треков (если же сессия содержит всего один трек, то в ход идет адрес области Lead-Out, примыкающей к треку). Искажение стартового адреса первого трека привело к тому, что разница стартовых адресов области Lead-Out и этого самого трека стала отрицательной. Действительно, 00:29:33 − 10:02:00 = 2058 − 45000 = − 42942, а если вспомнить, что LBA-адреса по стандарту выражаются 32-разрядными неотрицательными числами, становится понятно, как программа Alcohol 120% получила такой неестественно огромный объем (отрицательные числа — это такие числа, чей старший бит взведен, отсюда — маленькое отрицательное число — это очень-очень большое положительное). Расчеты показывают, что заявленное Alcohol 120% значение в 8 Тбайт достигается лишь при использовании 43-битных переменных. Вот это да! Программа Alcohol 120% явно проектировалась с запасом на будущее (а в будущем нас, как известно, ждут диски с объемами от 30 и более Гбайт, для адресации которых 32 бит оказывается

уже недостаточно, плюс еще необходимо учесть резерв, предназначенный для "отлова" отрицательных длин, образовавшихся в результате жестоких "извращений" с TOC, ведь программа Alcohol 120% — это защищенный копировщик!).

И вот наступает волнующий момент — момент "заливки" искаженного образа на диск CD-R/CD-RW.

Внимание

Используя диск CD-RW, вы должны отдавать себе отчет в том, что можете его безвозвратно потерять! Если ваш единственный пишущий привод откажется опознавать такой диск, очистка последнего окажется невозможной!

Благополучно "проглотив" искаженный образ, безо всяких препирательств со своей стороны, привод зажигает огонек индикации записи (если, конечно, на вашем приводе он есть) и приступает к делу. Проходит минута, другая, а индикатор прогресса по-прежнему остается на нуле. К исходу шестой минуты, когда пишущая головка достигает кромки диска, процесс записи аварийно прерывается приводом, и Alcohol 120%, издав грустное "бэмс", сигнализирует об аппаратной ошибке.

Просмотр "недорезанного" диска на приводах ASUS и NEC обнаруживает лишь первую сессию, а от второй не видно и следа. С приводом PHILIPS дела обстоят еще хуже — он вообще отказывается признавать вставленную в него "штуку" лазерным диском, и после непродолжительного скрежета своих механических внутренностей, сопровождаемых натужными завываниями перебирающего различные скорости мотора, индикатор "DISC IN" прощально гаснет. "Прощально" в том смысле, что с испорченной болванкой вам придется расстаться. Конечно, если это всего лишь дешевый диск CD-R, то туда ему и дорога, но потерять CD-RW — жалко. К счастью, на приводе NEC очистка диска протекает успешно, и, воодушевленные этим обстоятельством, мы продолжаем свои "издевательства" вновь.

Копировщик CloneCD ведет себя в этом отношении иначе. Во-первых, он оценивает длину искаженного трека в 4 294 868 664 Кбайт (листинг 6.5), что указывает на использование 32-разрядных переменных и вытекающую отсюда невозможность отличать положительные длины от отрицательных. Обратите внимание на размер первого трека первой сессии (он выделен полужирным шрифтом).

Листинг 6.5. Сводная информация по записываемому образу, выдаваемая CloneCD

```
ИНФОРМАЦИЯ О ФАЙЛЕ-ОБРАЗЕ:

Число сессий: 2
```

```
Занято на диске: 34850 Кбайт
Секторов: 15173
Время: 03:22:23 (мин:сек:кадр)

ИНФОРМАЦИЯ О СЕССИИ 1:

Размер сессии: 4726 Кбайт
Число треков: 1
Pregap: Данные Mode 1, размер: 103359 Кбайт
Track 1: Данные Mode 1, размер: 4294868664 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:

Размер сессии: 3939 Кбайт
Число треков: 1
Track 2: Данные Mode 1, размер: 3939 Кбайт
```

Во-вторых, обнаружив, что запись искаженного TOC на данном приводе невозможна, CloneCD корректирует TOC так, чтобы его облик принял "человеческий" вид. В результате, процесс "прожига" протекает без каких-либо ошибок, и мы получаем *как будто бы* работоспособный диск. Стартовый адрес первого трека начинается там, где кончается область Lead-In первой сессии (точнее, Pre-gap первого трека начинается там, где кончается Post-gap области Lead-In первой сессии, но это уже детали). Такой диск нормально читается в любом приводе CD-ROM, но! Если защитный механизм прочитает содержимое TOC, он легко обнаружит, что имеет дело с копией, но не оригиналом. Спрашивается: и на кой черт нам такое копирование нужно?! Хоть бы предупреждение было какое. Ладно, профессионалы запросто определят, в чем подвох, но в каком положении окажутся новички и/или просто квалифицированные пользователи, использующие программу CloneCD для своих нужд? В общем, мрак, одним словом.

Правда, *в режиме RAW DAO "нарезка" искаженного образа протекает отлично и CloneCD не вносит в TOC никакой отсебятины, благодаря чему, у нас образуется действительно защищенный CD*, который мы сейчас и будем "ломать".

Шаг шестой. Проверка работоспособности защищенного диска. Просмотр защищенного диска под приводом NEC показывает все файлы, даже те, что принадлежат первому треку — тому самому треку, чей стартовый адрес был жестоко искажен. Двойной щелчок мышью доказывает, что файлы не только присутствуют в каталоге, но и успешно открываются ассоциированным с ними приложениям и, судя по всему, выглядят вполне нормальными. Нашу душу начинают грызть смутные сомнения: действительно ли пишущий привод записал стартовый адрес первого трека таким, который мы просили, или молчаливо исправил его "на лету"?

Для ответа на этот вопрос мы должны исследовать геометрию диска, т. е. попросту говоря, прочитать TOC. Запускам уже полюбившийся нам Roxio Easy CD Creator и в меню **CD** находим пункт **CD Information**. Щелкаем по нему мышью, и на экране тут же раскрывается диалоговое окно с раскладкой диска.

Внимание

Не все программы способы "переваривать" искаженный TOC! Easy CD Creator это умеет, а вот, например, Stomp Record NOW — нет. В отсутствии подходящей утилиты вы можете воспользоваться программой raw.TOC.exe, поставляемой вместе с этой книгой на компакт-диске.

Как и следовало ожидать, стартовый адрес первого трека лежит далеко за пределами своей "родной" сессии, и его длина, будучи выраженная положительным числом, значительно, превышает доступную емкость диска (рис. 6.2). Так что все наши волнения абсолютно безосновательны!

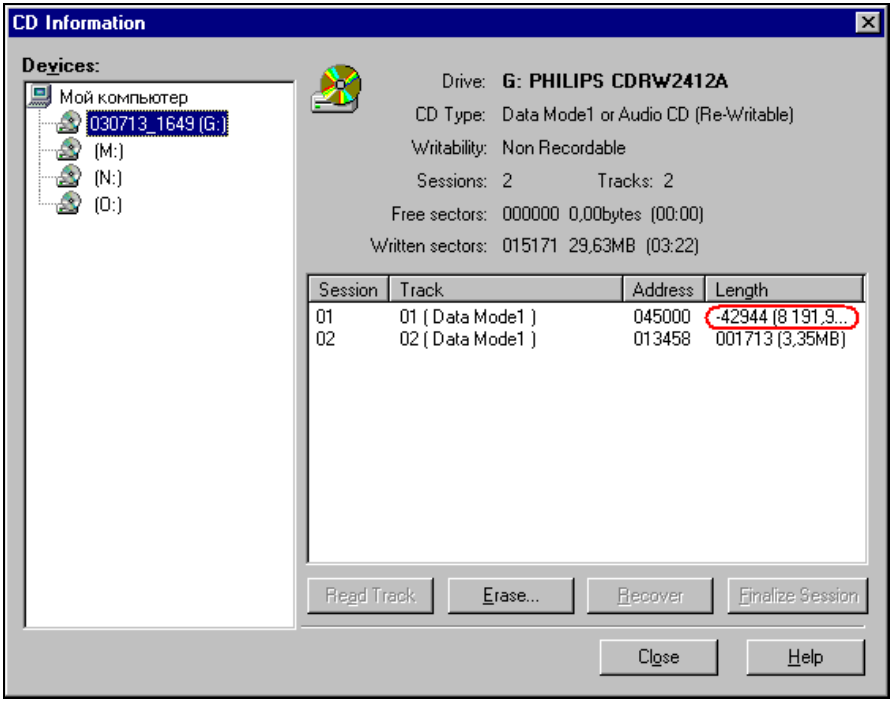


Рис. 6.2. Отрицательная длина первого трека сводит штатный копировщик "с ума"

Постойте, но как же тогда осуществляется доступ к содержимому первого трека? А кто вам вообще сказал, что лазерный диск адресуется по трекам?! Основной адресацией лазерного диска с данными является *сектор*. Абсо-

лютный же адрес всякого сектора однозначно определяется принадлежащим ему Q-каналом подкода (с учетом несовпадения границ секций и секторов максимально возможное расхождение, допускаемое стандартом, составляет 1 сек, т. е. 75 секторов, поэтому этот способ используется лишь для грубого позиционирования оптической головки). Точная наводка на цель выполняется непосредственно по самому секторному заголовку, в явном виде содержащему его абсолютный адрес. Номера треков в процессе обработки сектора вообще не участвуют, вернее, *могут и не участвовать.* Но могут ведь и участвовать! Все зависит от электронной начинки привода и его микропрограммной прошивки. Как именно они в этом участвуют, — сие есть великая тайна разработчиков привода, и простым смертным ее понять не дано. Но, так или иначе, встретив некорректный TOC, некоторые приводы запутываются, и в стройных битовых рядах возникает настоящая сумятица.

Результаты тестирования четырех моих приводов следующие: NEC и TEAC показывают содержимое обоих секций, корректно обрабатывая их содержимое. ASUS показывает только первую — искаженную — сессию и в упор не видит вторую, делая ее недоступной даже на секторном уровне. Зато файлы первой сессии обрабатываются вполне корректно. Привод PHILPS видит обе сессии, но корректно обрабатывает файлы лишь последней из них (т. е. той, что не искажена). Искаженная сессия доступна на секторном уровне, но нестабильно. Иногда без всяких видимых причин, привод PHILPS "едет крышей" и возвращает лишенный всякого смысла "мусор".

Мораль: *защитные механизмы, базирующиеся на искаженном TOC, не могут закладываться ни на одну из сессий. Поэтому обе сессии должны дублировать содержимое друг друга — авось хоть одну из них привод пользователя да прочитает.* Какой же тогда в этой защите смысл? А вот какой — пускай защита не может без риска для жизни привязываться к сессиям, она может привязаться к "сырому" содержимому TOC. О том, как осуществить такую привязку на практике, мы поговорим чуточку позднее, а пока попробуем скопировать защищенный диск нашими фаворитами — CloneCD и Alcohol 120%, естественно, не забывая и штатные копировщики.

Автоматическое копирование и обсуждение его результатов

Горячо любимая мной программа *Stomp Record Now* при попытке скопировать диск с искаженным стартовым адресом первого трека выводит сообщение *"Invalid disk"* и отказывается начинать операцию. В общем-то, это и не удивительно. Что можно взять с "юзерского" копировщика?

Гораздо интереснее протестировать поведение *Ahead Nero*, популярнейшего профессионального копировщика программ. Проверка показывает, что, независимо от состояния флажка **Ignore Illegal TOC Type**, находящегося во вкладке **Read options** и положения остальных опций, скопировать защищенный диск никак не получается. "Нюра" (программа Ahead Nero CD Speed) выдает сообщение *"Invalid track mode"* и даже не пытается начать чтение!

Служебная утилита Scan Disk Ahead Nero CD Speed также работает некорректно и выполняет сканирование отнюдь не первого трека, но той области, в которой расположен искаженный стартовый адрес (рис. 6.3). На рисунке видно, что утилита "согласилась" на сканирование диска с искаженным стартовым адресом, однако "залезла" совсем "не в ту степь", принявшись сканировать область диска, с адресом первого трека, указанную в TOC. Второй же трек здесь и вовсе не виден!

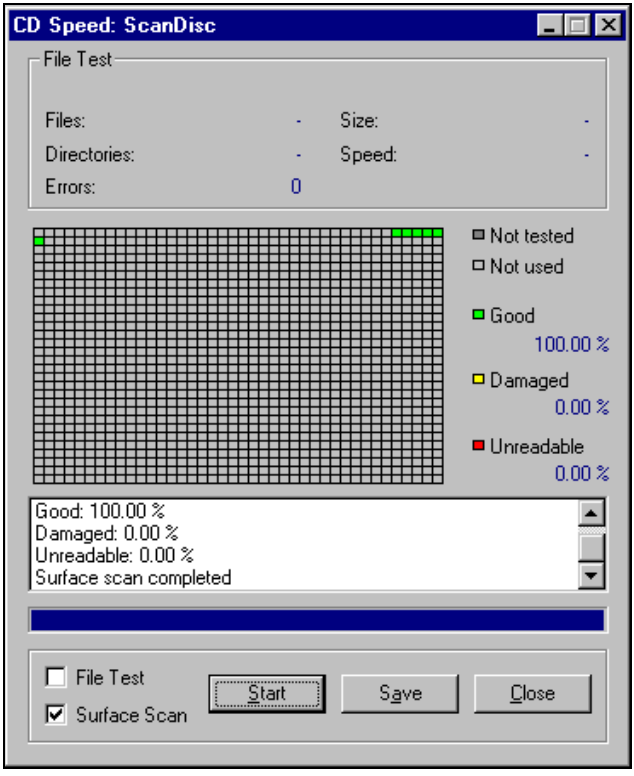


Рис. 6.3. Сканирование служебной утилитой Scan Disk программы Ahead Nero CD Speed

Теперь перейдем к копировщикам защищенных дисков, одним из которых является *CloneCD*, создатели которого утверждают, что он может справиться с любой, существующей ныне защитой.

В какой бы привод защищенный диск ни был вставлен, копировщик CloneCD выдает неизменно постоянный результат, не имеющий ничего общего с реальной действительностью. Обратите внимание (листинг 6.6), что он распознал на диске лишь одну сессию из двух (первую), да и то неправильно. По его скромному мнению, диск содержит всего одну сессию с общей

протяженностью в 4,6 Мбайт, но зато размер единственного трека составляет ни много ни мало — 3,9 Тбайт!

Листинг 6.6. Таким видит защищенный диск копировщик CloneCD

```
ИНФОРМАЦИЯ О CD В ДИСКОВОДЕ:

Число сессий: 1
Занято на диске: 4726 Кбайт
Секторов: 2058
Время: 00:27:33 (мин:сек:кадр)

ИНФОРМАЦИЯ О СЕССИИ 1:

Размер сессии: 4726 Кбайт
Число треков: 1
Pregap: Данные Mode 1, размер: 103359 Кбайт
Track 1: Data, размер: 4294868664 Кбайт
```

Еще до завершения процесса копирования нас начинают одолевать стойкие сомнения или, я бы даже сказал, непоколебимая уверенность, в том, что диск будет скопирован неправильно. И действительно, чего мы опасались, то мы и получили! Давайте создадим образ скопированного диска в плане сравнения копии TOC с оригиналом (листинг 6.7). Здесь несоответствующие поля выделены полужирным шрифтом.

Листинг 6.7. Образ защищенного диска, снятый программой CloneCD

```
[CloneCD]           ; данные о копировщике
Version=3           ; версия CloneCD

[Disc]                    ; данные о диске
TocEntries=7              ; кол-во элементов TOC'a == 7 (в оригинале было 12)
Sessions=1                ; кол-во сессий == 1 (в оригинале было 2)
DataTracksScrambled=0     ; поле DVD
CDTextLength=0            ; CD-Text'a в полях подкода Lead-In области нет

[Session 1]         ; данные сессии 1
PreGapMode=1        ; тип трека == Mode 1
PreGapSubC=0        ; данных подканала - нет

[Entry 0]           ; данные элемента TOC'a №0
```

```
Session=1          ; элемент сессии 1
Point=0xa0         ; номер первого трека сессии 1 в PMin/тип диска в PSec
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=1             ; номер первого трека сессии 1
PSec=0             ; тип диска CD-DA и CD-ROM диск в Mode 1
PFrame=0           ; не несет никакой полезной информации
PLBA=4350          ; номер трека представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 1]          ; данные элемента TOC'a №1
Session=1          ; элемент сессии 1
Point=0xa1         ; номер последнего трека сессии 1 в PMin
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
PMin=1             ; номер последнего трека сессии 1 (в сессии только один трек)
PSec=0             ; не несет никакой полезной информации
PFrame=0           ; не несет никакой полезной информации
PLBA=4350          ; номер трека представленный CloneCD как LBA-адрес, т.е. чушь

[Entry 2]          ; данные элемента TOC'a №2
Session=1          ; элемент сессии 1
Point=0xa2         ; положение Lead-out области в PMin:PSec:PFrame
ADR=0x01           ; q-Mode == 1
Control=0x04       ; диск с данными, запрещенный ;-) для копирования
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0             ; \
ASec=0             ;  + - абсолютный адрес текущего трека
AFrame=0           ; /
ALBA=-150          ; LBA-адрес текущего трека
Zero=0             ; это поле должно быть равно нулю, как оно и есть
```

```
PMin=0              ; \
PSec=29             ;  + - абсолютный адрес Lead-out области сессии 1
PFrame=33           ; /
PLBA=2058           ; LBA-адрес Lead-out области сессии 1

[Entry 3]           ; данные элемента TOC'a №3
Session=1           ; элемент сессии 1
Point=0x01          ; данные трека 1 сессии 1
ADR=0x01            ; q-Mode == 1
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=0              ; \
ASec=0              ;  + - абсолютный адрес текущего трека
AFrame=0            ; /
ALBA=-150           ; LBA-адрес текущего трека
Zero=0              ; это поле должно быть равно нулю, как оно и есть
PMin=10             ; \
PSec=2              ;  + - абсолютный адрес начала трека 1 сессии 1
PFrame=0            ; /
PLBA=45000          ; LBA-адрес начала трека 1 сессии 1

[Entry 4]           ; данные элемента TOC'a №4
Session=1           ; элемент сессии 1
Point=0xb0          ; позиция следующий записываемой области в AMin:ASec:AFrame
ADR=0x05            ; q-Mode == 1
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
TrackNo=0           ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=2              ; \
ASec=59             ;  + - абсолютный адрес следующей записываемой области
AFrame=33           ; /
ALBA=13308          ; LBA-адрес следующей записываемой области
Zero=3              ; кол-во pointer'ов в Mode 5
PMin=22             ; \
PSec=14             ;  + - абсолютный адрес максимальной записываемой области
PFrame=34           ; /
PLBA=99934          ; LBA-адрес максимальной записываемой области

[Entry 5]           ; данные элемента TOC'a №5
Session=1           ; элемент сессии 1
Point=0xc0          ; стартовый адрес Lead-In области Hybrid диска (если он есть)
ADR=0x05            ; Mode 5 (Оранжевая книга)
Control=0x04        ; диск с данными, запрещенный ;-) для копирования
```

```
TrackNo=0          ; трек, который мы сейчас читаем, - это Lead-In трек (т.е. TOC)
AMin=162           ; рекомендуемая мощность лазера для
ASec=200           ; Application code (в оригинале здесь было 128)
AFrame=224         ; в оригинале здесь было 140
ALBA=294074        ; LBA-"адрес" трех предыдущих полей
Zero=0             ; зарезервировано
PMin=97            ; \
PSec=27            ;  + - абсолютный адрес Lead-In области Hybrid диска
PFrame=21          ; /    (адрес лежит за пределами диска, т. е. Hybrid-диска нет)
PLBA=-11604        ; LBA-адрес Lead-In области Hybrid'а(вычислен с переполнением)

[Entry 6]          ; данные элемента TOC'а №6
Session=1          ; элемент сессии 1
Point=0xc1         ; копия ATIP-информации
ADR=0x05           ; -+
Control=0x04       ; -+
TrackNo=0          ; -+
AMin=4             ; -+
ASec=192           ; -+
AFrame=150         ; -+- ATIP (изменена!)
ALBA=32400         ; -+
Zero=0             ; -+
PMin=0             ; -+
PSec=0             ; -+
PFrame=0           ; -+
PLBA=-150

[TRACK 1]
MODE=0
INDEX 1=45000
```

Сокращение сессий с двух до одной очень сильно смущает. Куда девалась вторая — неискаженная(!) — сессия, вообще непонятно. И хотя искаженные данные первого трека сохранились, оказались неожиданно измененными поля `Application Code` и `ATIP` (и это несмотря на то, что запись производилась на ту же самую болванку CD-RW, что и раньше, хотя ее прожиг осуществлялся различными приводами). Самое удивительное — вместо действительного адреса выводной области программа CloneCD указала какую-то муть. По ее мнению, абсолютный Lead-Out-адрес равен 00:29:33, в то время как Lead-Out оригинального диска располагался в позиции 03:24:23, а стартовый адрес первого трека скопированного диска — 10:02:00. Да! Адрес выводной области оказался расположенным до начала стартового адреса первого

трека! Вот так копировщик — не справился с родной защитой диска, но "навесил" на него свою собственную. Между прочим, диски с искаженным адресом выводной области способны выводить механику приводов из строя на чисто физическом уровне. Как следствие: скопированный диск оказывается работоспособен не на всех приводах (ASUS, NEC и TEAC его прочитают, хотя увидят лишь первую сессию, а вот PHILIPS — откажется использовать такой диск вообще), к тому же защите ничего не стоит прочитать текущий TOC и сравнить его с эталонным. Благодаря тому обстоятельству, что TOC скопированного диска оказался чудовищно искажен, становится легко отличить оригинал от его пиратского дубликата (конкретный пример привязки см. в *разд. "Пример реализации защиты на программном уровне" этой главы*).

Короче говоря, факир был пьян и фокус не удался. Что ж, попробуем обратиться за помощью к программе Alcohol 120% — уж она-то должна наверняка с этим справиться! Действительно, Alcohol 120% видит обе сессии: как искаженную, так и неискаженную, однако по малопонятным причинам сохраняет в образ лишь вторую из них (CloneCD сохранял первую). Ну что это за "зоопарк" такой, а? Кажется, что содержимое TOC скопированного диска можно даже и не сравнивать — там будет далеко не то, что защита собирается ожидать. Тем не менее, вопреки всем пессимистическим предчувствиям, содержимое TOC, снятое программой Alcohol 120%, практически полностью соответствует оригиналу. Единственно, в чем ошибся Alcohol 120%, — определил тип Pre-gap обоих треков не как Mode 1, но и как Mode 2 (рис. 6.4). Впрочем, в силу отсутствия в образе первой сессии (рис. 6.5), полученная с его помощью копия диска все равно оказывается неработоспособной.

А ведь заявлялось, что копировщики CloneCD/Alcohol 120% способны копировать любые существующие на сегодняшний момент защищенные диски, и вдруг на проверку оказывается, что даже такую простую защиту, которую может создать "на кончике пенька" любой программист (даже начинающий!), они преодолеть ни вместе, ни по раздельности не в состоянии! Причем аппаратура, на которой все эти эксперименты и осуществлялись, возможность корректного копирования искаженного диска гарантированно поддерживает (сам проверял!), и потому отмахнуться физическими ограничениями приводов разработчикам обоих копировщиков уже не удастся!

Копировщик Alcohol 120% сообщает (рис. 6.6): "Размер образа НЕ соответствует Lead-Out, записанному в TOC!" (Что есть прямое следствие ошибочного снятия образа копировщиком из-за искажения стартового адреса первого трека.) "Адрес Lead-Out в TOC: 03:22:23. Размер образа: 12:54:65. Для продолжения выберите одну из опций ниже!" Предлагаемые опции:

- Сохранить TOC как на исходном CD, записать до конца файла-образа;
- Изменить адрес Lead-out по длине файла-образа;
- Сохранить TOC как на исходном CD, записать до адреса Lead-out.

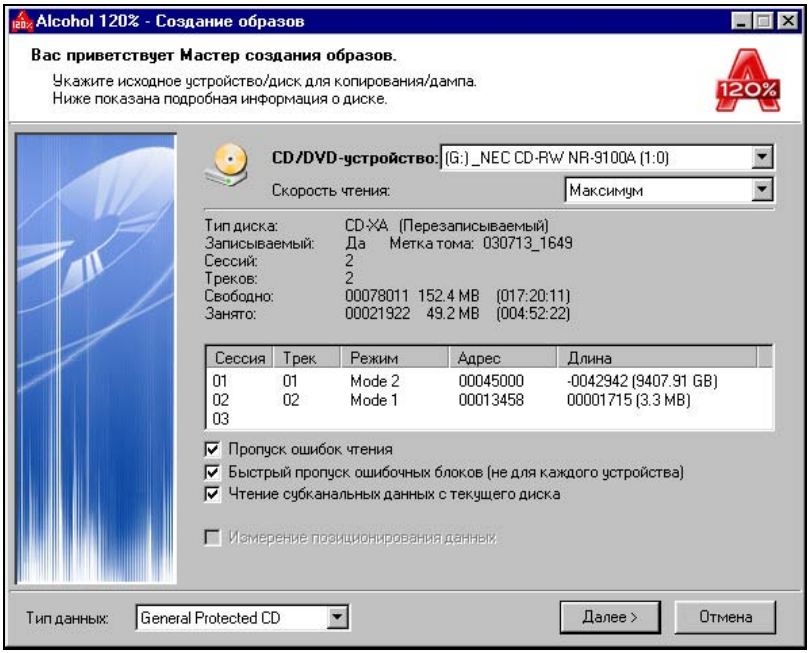


Рис. 6.4. 0x061 Alcohol 120% видит обе сессии защищенного диска, но...

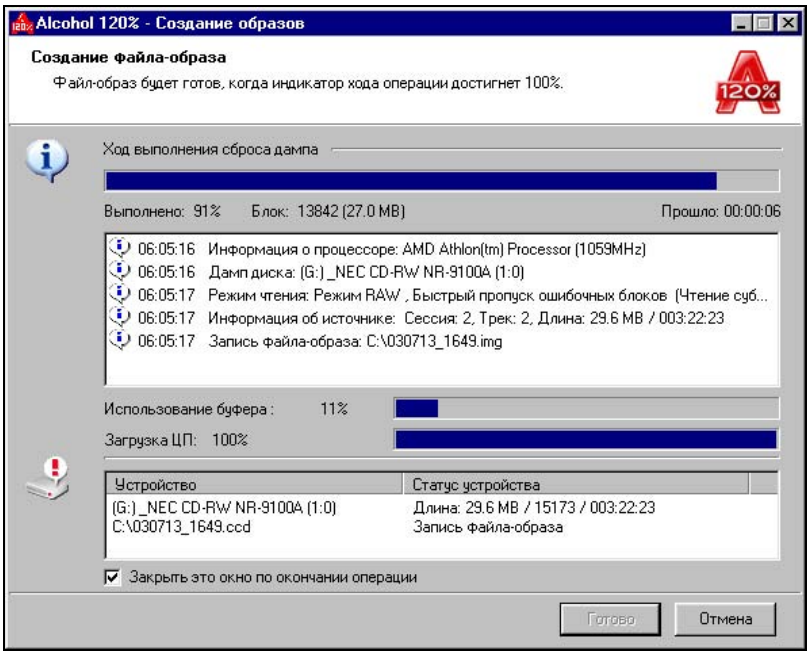


Рис. 6.5. Alcohol 120% копирует лишь вторую сессию, а первую нагло пропускает

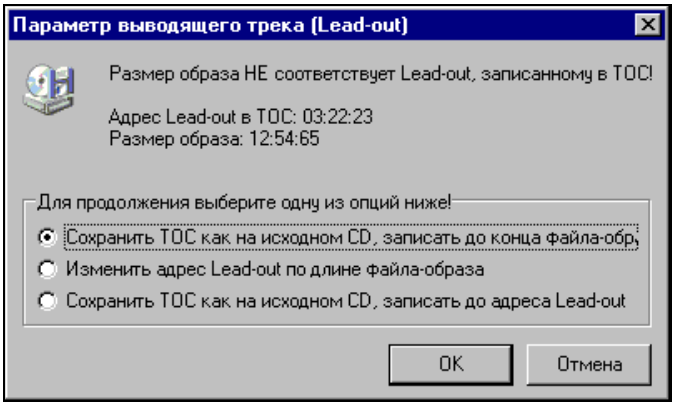


Рис. 6.6. Сообщение, выводимое программой Alcohol 120%

Выбираем "сохранить TOC как на исходном CD, записать до конца файла-образа" и получаем:

```
08:46:48 (G:) TEAC CD-W552E (1:1): Во время записи произошла ошибка!
 08:46:48 Ошибка:  [05/26/00] - Invalid Field In Parameter List
 08:46:48 (G:) TEAC CD-W552E (1:1): Во время записи произошла ошибка!
 08:46:48 Загрузка файла-образа отменена!
 08:46:51 Во время записи произошли какие-то неполадки! Просмотрите файл с отчётом
и сообщите обо всех ошибках в службу техподдержки.
```

Даже не верится, что такой простой прием "ослепляет" лучшие копировщики защищенных дисков! Неужели и вправду, создание некопируемых дисков вполне осуществимо на обыкновенном бытовом оборудовании?! Да! Именно так! Конечно, не стоит путать некопируемость диска автоматическими копировщиками с принципиальной невозможностью получения его идентичной копии. В ручном режиме копирование таких дисков вполне осуществимо (правда, при условии, что ваш пишущий привод поддерживает режим RAW DAO, а читающий — читает сектора из обоих секций), и сейчас мы продемонстрируем как.

Так как же все-таки скопировать такой диск?

Конечно, с помощью "Добермана Пинчера" ("Pinch of File") (или любого другого блочного копировщика файлов), HIEW, двух образов защищенного диска (один — с первой сессией — от CloneCD, другой — со второй сессией — от Alcohol 120%) и еще "чьей-то матери" мы можем воссоздать идентичную копию оригинального диска путем их совокупного объединения, но это будет как-то не по-хакерски, да и вообще некрасиво.

Чтобы не писать свою собственную программу прожига диска, ограничимся использованием CloneCD. При условии, что подсунутый ему образ диска запечатлен правильно, CloneCD обычно справляется с прожигом на ура.

Итак, у нас есть более-менее верный файл IMAGE.CCD, содержащий TOC (его можно позаимствовать от программы Alcohol 120%), но недостает файла-образа IMAGE.IMG. Попробуем его получить? Будем отталкиваться от того, что LBA-адреса всех секторов диска пронумерованы последовательно, включая области, занятые Lead-In/Lead-Out и прочим служебным "барахлом". Разумеется, непосредственное чтение служебных областей диска на секторном уровне невозможно, но именно на этом мы и собираемся сыграть! Последовательно читая диск с первого по последний сектор, мы обнаружим, что сектора с LBA-адресами с 0 по 2055 сектор включительно читаются без каких-либо проблем, после чего наступает "сумеречная зона" нечитающихся секторов, протянувшаяся вплоть до 13307 сектора. Здесь сектора либо совсем не читаются, либо возвращаются в сильно измененном виде, легко опознаваемом по отсутствию правильной синхропоследовательности в их заголовке. Наконец, с адреса 13308 чтение вновь продолжается без каких-либо проблем.

Судя по всему, мы имеет дело с двухсессионным диском и "сумеречная зона" между сессиями есть не что иное, как области Lead-Out/Lead-In. Накинув два сектора на область Post-gap (при условии, что она записана с соблюдением стандарта), получаем, что LBA-адрес последнего значимого сектора первой сессии составляет: 2057 или, в пересчете на абсолютные единицы, 00 минут, 29 секунд и еще 32 фрейма. Соответственно, LBA-адрес первого сектора второй сессии равен: 13308 + 150 (Pre-gap) = 13458 или 3 минуты, 1 секунда, 33 фрейма. Конечно, если исследуемый диск содержит большое количество ошибок, то его анализ значительно усложняется, т. к. физические дефекты на сектором уровне могут выглядеть точно так же, как области Lead-In/Lead-Out, конечно, при том условии, что дефективные области имеют соответствующую протяженность — а это вряд ли.

Отбросив сектора, расположенные в зонах Pre-gap и Post-gap (т. е. 150 секторов от конца первой читаемой области и ровно столько же от начала следующей), мы должны объединить их в один файл, используя для этой цели любой файловый копировщик (например, штатную команду MS-DOS `copy file_1 /b + file_2 image.img`). Остается прочитать "сырой" TOC SCSI/ATAPI командой `READ TOC` (opcode: `43h`, format: `2h`) и записать его в файл IMAGE.CCD в соответствии с синтаксисом CloneCD. Как альтернативный вариант — можно воспользоваться CCD-файлом, сформированным программой Alcohol 120%, предварительно скорректировав Pre-gap Mode (как уже сказано ранее, Alcohol 120% определил его неправильно, перепутав Mode 1 с Mode 2). Согласно стандарту, режим сектора задается пятнадцатым, считая от нуля, байтом его заголовка. Если этот байт равен одному (что, собственно, и наблюдается в нашем случае), то и Mode-сектор будет 1, но не 2.

При условии, что все сделано правильно, после записи собственноручно сформированного образа диска, мы получаем практически идентичный ори-

гинал. Просто? Да проще простого! И написать автоматический копировщик, автоматизирующий наш труд, можно буквально за несколько часов! Если чтение "сырых" секторов с диска представляет для вас проблему, воспользуйтесь исходными текстами утилит ASPI32.raw/SPTI.raw, как раз такое чтение и осуществляющих.

Так что искажение TOC — не очень-то надежный прием защиты от копирования, как ни крути. Правда, от обычных пользователей, вооруженных программой CloneCD или Alcohol 120%, он все-таки спасает, а больше от защиты зачастую и не требуется.

Пример реализации защиты на программном уровне

Покажем теперь, как такая защита может быть реализована на программном уровне. Самое простое, что можно сделать, — отправить приводу команду "сырого" чтения TOC: `READ TOC` (opcode: `43h`, format: `2h`) и сравнить возвращенный ею результат с эталоном. Какие именно поля TOC защита будет проверять — это ее личное дело. По минимуму достаточно проверить количество сессий и стартовый адрес искаженного трека. По максимуму можно контролировать весь TOC целиком. Естественно, от побайтового сравнения контролируемого TOC с оригиналом настоятельно рекомендуется воздержаться, т. к. это неявно закладывает защиту на особенности микропрограммной прошивки читающего привода. Стандарт ничего не говорит том, в каком порядке должно возвращаться содержимое TOC, и потому его бинарное представление может варьироваться от привода к приводу (хотя на практике такого и не наблюдается). Грамотно спроектированная защита должна анализировать только те поля, к содержимому которых она привязывается явно.

Демонстрационный пример, приведенный в листинге 6.8, как раз и иллюстрирует технику корректной привязки к TOC. Разумеется, явная проверка целостности TOC может быть элементарно обнаружена хакером и выкинута из программы как ненужная, поэтому не стоит копировать этот демонстрационный пример один к одному в свои программы. Лучше используйте значения полей TOC как рабочие константы, жизненно необходимые для нормальной работоспособности программы, — в этом случае сличение "паспортов с лицами" будет не столь наглядным. Естественно, явная проверка оригинальности диска все равно обязана быть, но ее основная цель отнюдь не защитить программу от взлома, а довести до сведения пользователя, что проверяемый диск, с точки зрения защиты, не является лицензионным.

Листинг 6.8. Демонстрационный пример простейшей защиты, привязывающейся к искаженному TOC и не позволяющей себя копировать

```
/*-----------------------------------------------------------------------------
 *
 *                    crack me 9822C095h
```

```
 *                    ==================
 *
 *    демонстрация техники привязки к искаженному TOC'у; для работы программе
 * требуется лазерный диск, прожженный соответствующим образом
 *
-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <windows.h>
#include "CD.h"
#include "SPTI.h"
#include "ASPI32.h"

// параметры защищенного диска, которые мы будем проверять
//-------------------------------------------------------
#define _N_SESSION        2              // кол-во сессий
#define _TRACK            1              // номер проверяемого трека
#define _TRACK_LBA        0x6B124        // стартовый LBA-адрес трека _TRACK

// параметры программы
//--------------------
#define MAX_TRY           3              // мак. кол-во попыток чтения TOC'a
#define    TRY_DELAY      100            // задержка между попытками
#define MAX_TOC_SIZE      (2352)         // максимальный размер TOC'a


main(int argc, char **argv)
{
    long a, real_len, try = 1;           // основные переменные
    unsigned char TOC[MAX_TOC_SIZE];     // сюда будет читаться TOC
    unsigned char CDB[ATAPI_CDB_SIZE];   // SCSI CDB-блок для SCSI/ATAPI-устройств

    // TITLE
    fprintf(stderr,"crackme 9822C095 by Kris Kaspersky\n");

    if (argc <2)
    {
         fprintf(stderr,"USAGE:crackme.9822C095h.exe drive\n");
         fprintf(stderr,"\tdrive - \\\\.\\X: or Trg.Lun\n");
         return -1;
    }

    // инициализация буферов
    memset(CDB, 0, ATAPI_CDB_SIZE); memset(TOC, 0, MAX_TOC_SIZE);
```

```
    // готовим CDB-блок
    CDB[0] = 0x43;                      // READ TOC
    CDB[2] = 0x2;                       // RAW TOC
    CDB[6] = 0;                         // номер первой сессии
    CDB[7] = HIBYTE(MAX_TOC_SIZE);      // размер...
    CDB[8] = LOBYTE(MAX_TOC_SIZE);      //             ...буфера

    // читаем TOC
    while(1)
    {
         // посылаем CDB-блок SCSI/ATAPI-устройству
         a = SEND_SCSI_CMD(argv[1], CDB, ATAPI_CDB_SIZE, NO_SENSE,
                                 TOC, MAX_TOC_SIZE, SCSI_DATA_IN);
         if (a == SCSI_OK) break;    // TOC успешно прочитан, рвем когти

         // произошла ошибка. что ли привод не готов?
         Sleep(TRY_DELAY);              // выдерживаем паузу
         if (try++ == MAX_TRY)          // макс. кол-во попыток уже вышло?
              { fprintf(stderr,"-ERR: can not read TOC\x7\n"); return -1;}
    }

    // TOC прочитан, приступаем к его анализу
    //--------------------------------------

    // проверка кол-ва сессий
    if ((TOC[3] - TOC[2]) != (_N_SESSION-1))
            {fprintf(stderr, "-ERR: not original CD\n");return -1;}

    // проверка стартового LBA-адреса трека _TRACK
    //--------------------------------------------
    real_len = TOC[0]*0x100L+TOC[1];    // определение реальной длины TOC'a
    for (a = 4; a < real_len; a+=11)    // перебор всех entry
    {
         if (TOC[a+3] == _TRACK)        // это наш трек?
              if ((((TOC[a+4]*60L)+TOC[a+5])*75L)+TOC[a+6] != _TRACK_LBA)
                  {fprintf(stderr, "-ERR: not original LBA\n");return -1;}
         else
              break;
    }

    // это оригинальный диск!
    printf("Hello, original CD\n");
}
```

Предлагаемая защита не копируется программой CloneCD (т. к. она создает всего одну сессию вместо ожидаемых двух), но легко обходится программой Alcohol 120%, которая хоть и помещает на место первой секции "непотребный мусор", зато вполне корректно воссоздает оригинальный TOC.

Для усиления защиты мы можем попытаться не только проверять обе сессии на существование, но и контролировать целостность их содержимого. Разумеется, не обязательно "перелопачивать" каждую из секций целиком. Достаточно выбрать несколько ключевых секторов, желательно имеющих по возможности уникальное содержимое. "Постойте! — воскликнет внимательный читатель. — Разве автор не предостерегал нас о последствиях такой проверки?! Ведь никто не может гарантировать, что на оборудовании пользователя эти сектора вообще прочтутся!" "Что ж, — отвечу я. — Закладываться на *читабельность* секторов, действительно, категорически не рекомендуется, но вот контролировать *успешно просчитавшиеся* сектора можно и нужно!" То есть: если ключевые сектора не читаются, то все хорошо и нет никаких поводов считать диск нелицензионным — это просто у конечного пользователя оборудование такое (в смысле "кривое"). Другое дело, если чтение секторов прошло без ошибок, но вместо ключевых данных в них оказалось нечто совсем иное. Вот тогда, действительно, проблема не в оборудовании, а в диске.

Усиленный вариант защиты уже не копируется программой Alcohol 120% (т. к. вместо оригинального содержимого первой сессии она помещает на диск какой-то "дикий мусор"), но может быть скопирован вручную по методике, описанной ранее. К тому же, привязка к искаженному TOC элементарно "отламывается" в отладчике/дизассемблере. Как? Сейчас отвечу. Так что дальнейшее совершенствование защиты практически полностью бессмысленно. От "простых смертных" пользователей мы уже защитились, а от хакеров мы не сумеем защититься все равно (во всяком случае, не этим способом). В любом случае, более "продвинутые" защиты — тема отдельного разговора.

Полная нейтрализация защиты

Получить работоспособную копию защищенного диска — это только полдела. Законченный взлом подразумевает как минимум восстановление искаженного TOC и отвязку защиты от диска. Другими словами, корректно взломанный диск должен копироваться на любом оборудовании, любыми штатными копировщиками и при этом не должен ни с чем конфликтовать.

Процесс отвязки обычно начинается с анализа геометрии диска на предмет выявления реально используемых сессий/секторов. Искаженные сессии обычно не несут никакой полезной нагрузки — для защиты важен лишь сам факт их существования, но отнюдь не содержимое, которое на некоторых приводах вообще не доступно. А какие существуют способы проверки секций на существование? На "железном" уровне способ один — компьютер посылает приводу SCSI/ATAPI команду `READ TOC` и получает в ответ полный каталог содержимого. На программном уровне взаимодействие с аппаратурой

чаще всего осуществляется посредством ASPI/SPTI-интерфейса, реже — непосредственно через порты ввода-вывода. Некоторые защиты предпочитают действовать через драйвер привода CD-ROM, полагая, что этот способ более цивилизован и менее опасен.

В любом случае у хакера есть два пути — либо локализовать команду чтения TOC в машинном коде (обычно это делается установкой точки останова на функцию `SendASPI32Command`) и попытаться переписать код программы так, чтобы она корректно работала с любым TOC, либо путем перехвата функции `CreateFile/DeviceIoContorl` внедрить "шпиона", отслеживающего весь проходящий через него поток SCSPI/ATAPI-команд и, в случае чтения TOC защищенного диска, возвращающего защите подложные данные. Первый путь более надежен, а значит, и более практичен. Рассмотрим его поподробнее.

Фиктивный трек в настоящем треке

Не зная броду, больше шансов утопиться.

Народная мудрость

Тот факт, что диски с данными адресуются исключительно на секторном уровне, дает большой простор для извращений с раскладкой треков, — ни сам привод, ни операционная система не обращают на это обстоятельство ни малейшего внимания, но сбивает с толку подавляющее большинство копировщиков, включая копировщиков защищенных дисков, пытающихся скопировать диск именно по трекам, а не по секторам. Еще больший эффект дает размещение фиктивных треков в служебных областях, которые либо вовсе не могут быть скопированы приводом, либо завязаны на малоизвестных и редко используемых структурах, присутствие которых копировщики предпочитают не замечать. Но для начала разберемся, как организованы стандартные треки и как все это хозяйство ухитряется работать.

По соображениям экономии места служебные структуры лазерных дисков содержат минимум необходимой информации и длина треков нигде явным образом не хранится. В грубом приближении она вычисляется путем вычитания стартового адреса текущего трека от стартового адреса следующего трека (стартового адреса выводной области диска — если текущий трек в сессии последний). Сами же стартовые адреса хранятся в *оглавлении диска* (листинг 6.9), так называемом *TOC* (*Table Of Contents*).

Листинг 6.9. Пример содержимого оглавления диска в "сыром" виде с комментариями

```
session number
 | ADR/control
 |  | TNO
 |  |  | point
 |  |  |  | AM:AS:AF
```

```
 |  |  |  |  |  |  | zero
 |  |  |  |  |  |  |  | PM:PS:PF
01 14 00 A0 00 00 00 00 01 00 00 ← номер первого трека первой сессии диска
01 14 00 A1 00 00 00 00 02 00 00 ← номер последнего трека первой сессии диска
01 14 00 A2 00 00 00 00 00 1D 21 ← адрес выводной области первой сессии диска
01 14 00 01 00 00 00 00 00 02 00 ← стартовый адрес трека N1
01 14 00 02 00 00 00 00 00 11 00 ← стартовый адрес трека N2
02 14 00 A0 00 00 00 00 03 00 00 ← номер первого трека второй сессии диска
02 14 00 A1 00 00 00 00 03 00 00 ← номер последнего трека второй сессии диска
02 14 00 A2 00 00 00 00 03 18 17 ← адрес выводной области второй сессии диска
02 14 00 03 00 00 00 00 03 01 21 ← стартовый адрес трека N3
```

Между концом области Lead-In и стартовым адресом первого трека каждой сессии расположена *область предзазора*, иначе называемая областью *Pre-gap*, протяженностью в 150 секторов, формально принадлежащая первому треку и по стандартам "Красной" и "Желтой" книг (базовые стандарты для аудиодисков и дисков с данными соответственно) не содержащая никаких полезных данных и на штампованных дисках CD-ROM обычно заполненная нулями. Тип области предзазора совпадает с типом относящегося к ней трека, и она сконструирована по его образу и подобию. А это значит, что для треков, записанных в MODE1, MODE2 FORM1 и MODE2 FORM2 область предзазора оказывается совсем не пустой. Как минимум, она содержит корректные заголовки секторов, а как максимум — заголовки секторов, контрольную сумму, корректирующие коды Рида-Соломона и прочую служебную информацию.

Листинг 6.10. Сектор из области Pre-gap аудиотрека (слева) и трека с данными (справа)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 FF FF FF FF FF FF FF FF FF FF 00 00 00 02 01
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
…                                               | …
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 69 A0 A7 82 CA 8A 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 CA 65 65 BC AF D9 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 A7 5B BD 72 88 0A 92 23 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 3D 90 90 48 AD D8
```

Между концом последнего трека и выводной областью каждой сессии расположена *область постзазора*, иначе называемая областью *Post-gap*, протяженностью от 150 и более секторов, формально принадлежащая последнему треку и, аналогично области Pre-gap, не содержащая никаких данных. Тип области постзазора такой же, как и у предшествующего ей трека.

Если за треком одного типа следует трек другого типа (например, MODE1 сменяется на MODE2 или аудиотреки чередуются с треками данных), такие треки разделяются *переходной областью* (transition area) протяженностью по меньшей мере в 350 секторов. Первые 150 секторов занимает область Post-gap предшествующего трека, а остальные 200 секторов принадлежат *расширенной* (Extended) pre-gap области последующего трека. Расширенная область предзазора состоит из двух частей, занимающих по 50 и 150 секторов соответственно. Первые 50 секторов сохраняют тип предшествующего им трека, а оставшиеся 150 секторов представляют собой обычную область предзазора.

Треки данных идентичного типа могут располагаться как вплотную друг к другу, так и разделяться переходными областями. Однако некоторые копировщики (в частности, Ahead Nero) ошибочно полагают, что переходные области между соседними треками присутствуют всегда, и в порядке собственной инициативы пропускают последние ~350 секторов каждого трека. Поэтому диски без переходных областей (или с укороченной переходной областью) такими копировщиками копируются некорректно, несмотря на свое полное соответствие стандарту.

Заметим, что здесь указаны лишь минимально допускаемые по стандарту размеры переходных областей, а их предельная длина практически ничем не ограничена. Размеры переходных областей нигде в явном виде не хранятся, и для определения их границ необходимо проанализировать субканальные данные. Конкретно — содержимое поля INDEX Q-канала подкода. Нулевое значение соответствует Pre-gap (или, применительно к аудиодискам — паузе), любое другое — действительному сектору трека или области Post-gap. Таким образом, область постзазора ничем не отличается от предшествующего ей трека, и копировщик не в состоянии определить ее длину, и наличие Post-gap распознается лишь по косвенным признакам, а именно по отсутствию информации в пользовательской части последних секторов трека. Грамотно спроектированный копировщик должен копировать содержимое всех сессий диска целиком — от первого до последнего принадлежащего им сектора,

не пытаясь анализировать раскладку треков, ибо она может быть произвольным образом искажена (адресация дисков с данными идет исключительно на секторном уровне, и треки в ней не участвуют, а потому искажение их атрибутов вполне допустимо и не приводит ни к каким возмущениям со стороны операционной системы). К сожалению, подавляющее большинство копировщиков (включая и копировщики защищенных дисков) негласно закладываются на стандартные размеры переходных областей и крайне чувствительны к их искажениям. Обратите внимание (листинг 6.11), что второй трек начинается с адреса 465h, что соответствует абсолютному адресу 00:11h:00, приведенному в листинге 6.9; адрес начала Pre-gap, равный 3CFh, отстоит от стартового адреса трека ровно на 96h (150) секторов, следовательно, данная область Pre-gap полностью соответствует стандарту.

Листинг 6.11. Определение длины области Pre-gap по субканальным данным

```
                        ++- номер трека
                        !!  ++- index
03CC:00 15 00 0C 01 14 01 01 00 00 03 CC 00 00 03 CC
03CD:00 15 00 0C 01 14 01 01 00 00 03 CD 00 00 03 CD
03CE:00 15 00 0C 01 14 01 01 00 00 03 CE 00 00 03 CE ← конец post-gap первого трека
03CF:00 15 00 0C 01 14 02 00 00 00 03 CF 00 00 00 96 ← начало pre-gap второго трека
03D0:00 15 00 0C 01 14 02 00 00 00 03 D0 00 00 00 95
03D1:00 15 00 0C 01 14 02 00 00 00 03 D1 00 00 00 94
…
0462:00 15 00 0C 01 14 02 00 00 00 04 62 00 00 00 03
0463:00 15 00 0C 01 14 02 00 00 00 04 63 00 00 00 02
0464:00 15 00 0C 01 14 02 00 00 00 04 64 00 00 00 01 ← конец pre-gap второго трека
0465:00 15 00 0C 01 14 02 01 00 00 04 65 00 00 00 00 ← начало второго трека
0466:00 15 00 0C 01 14 02 01 00 00 04 66 00 00 00 01
0467:00 15 00 0C 01 14 02 01 00 00 04 67 00 00 00 02
```

Однократно записываемые и перезаписываемые лазерные диски используют Pre-gap для хранения такой экзотической и малоизвестной структуры данных, как *TDB (Track Descriptor Block — блок описания трека)*, содержащей сведения о режиме записи, размере одного пакета и т. д. Стандарт предписывает "прожигать" блок описания трека в режиме пакетной записи и режиме *TAO (Track At Once — по треку за раз)*, однако большинство программ "прожига" (включая уже упомянутый Ahead Nero) "прожигают" TDB во всех доступных режимах записи, включая DAO. В листинге 6.12 представлен пример TDB с диска, прожженного Ahead Nero (диск записан в XA MODE2 FORM1, поэтому пер-

вый байт пользовательской области начинается со смещения 17h, а не 10h, как это происходит в MODE1).

Листинг 6.12. Пример TDB с диска, "прожженного" Nero

```
000:00 FF FF FF FF FF FF FF FF FF FF 00 00 00 05 02              ♣☻ ; sector head
010:00 00 00 00 00 00 00 00 54 44 49 01 50 01 01 01        TDI☺P☺☺☺ ; TDT-блок \
020:01 80 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 ☺A              ; TDU-блок /
TDB
030:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
040:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
050:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
…
810:00 00 00 00 00 00 00 00 C3 0C 2E 82 00 00 00 00         ├♀.B     ; к   к
820:00 00 00 00 00 00 00 00 93 78 85 F5 60 F5 F5 F5         УхЕї`їїї ; о   о
830:F5 0B AA AA AA 00 00 00 00 00 00 00 00 00 00 00 ї♂ккк            ; р   д
840:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                  ; р   ы
850:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                  ; е
860:00 00 00 00 00 00 00 00 00 00 00 00 00 00 58 14               X¶ ; к   Р
870:72 9B 00 00 00 00 00 00 00 00 00 00 00 00 C7 3C rЫ            ╟< ; т   И
880:CC F4 30 F4 F4 F4 F4 8B 55 55 55 00 00 00 00 00 ╠╜ї0ї╜╜╜JUUU      ; и   Д
890:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                  ; р   А
8A0:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                  ; у
8B0:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                  ; ю   С
8C0:00 00 00 00 9B 18 5C 19 00 00 00 00 00 00 00 00     Ы↑\↓         ; щ   О
8D0:00 00 00 00 00 00 72 9B E5 94 71 47 E6 48 00 00        rЫхФqGцН ; и   Л
8E0:D1 00 F3 15 CC F5 2B 2C B1 AF F6 51 41 80 E0 F2 ╤ єS╠ї+,▒пЎQAAрЄ ; е   О
8F0:23 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 #@               ;     М
900:00 00 00 00 00 00 00 00 00 00 5C 19 54 03 75 4A           \↓T♥uJ ;     О
910:7D 50 00 00 7B 00 0C BF 93 AB D5 AD 24 2E 42 51 }P  { ♀┐Ул╓н$.BQ ;     Н
920:4E 0D 6E CF 77 04 00 00 00 00 00 00 00 00 00 00 N♪n╧w♦            ;     А
```

Блок описателя трека занимает один сектор, начинаясь с первого байта его пользовательской части, и дублируется во всех секторах второй половины Pre-gap данного трека. На структурном уровне он стоит из двух частей: *таблицы описателя трека* и одного или двух *модулей описателей трека*, сокращенно обозначаемых как *TDT* (Track Descriptor Table) и *TDU* (Track Descriptor Unit) соответственно. Дешифровка TDT в представленном листинге 6.12: длина Pre-gap — 150 секторов, данный TDB относится только к первому треку, следом за блоком TDT идет один-единственный TDU, описывающий текущий трек; дешифровка TDU: тип записи — непрерывная запись.

Таблица описателя трека начинается со специальной сигнатуры: TDI (`54h 44h 49h`), что расшифровывается как *Track Descriptor Identification* (*идентификатор описателя трека*), следующие два байта хранят заявленную длину Pre-gap области, записанную в BCD-формате. Поле Type of Track Description Unit (табл. 6.1) указывает на количество модулей описания трека (сокращенно TDU от *Track Description Unit*), начинающихся непосредственно за концом блока TDB. Единичное значение соответствует одному-единственному модулю, относящемуся к текущему треку. Нулевое значение указывает на наличие двух модулей, следующих друг за другом, первый из которых описывает атрибуты предшествующего трека, а текущий трек специфицируется вторым модулем.

Поля Lowest Track Number и Highest Track Number, записанные в BCD-формате, содержат наименьший и наибольший номера треков, описанных в данном TDB, и используются главным образом в режиме пакетной записи для определения режимов предпочтительной записи. Во всех остальных случаях следить за корректностью этих полей необязательно.

Первый байт модуля описателя трека содержит BCD-номер трека, который он, собственно, и описывает. Следующий за ним байт специфицирует метод записи и может принимать следующие значения:

- `00000000b` — непрерывная запись (аудио-трек);
- `10010000b` — непрерывная запись (всего лишь один пакет);
- `10010000b` — инкрементная запись с пакетами переменной длины;
- `10010001b` — инкрементная запись с пакетами фиксированной длины.

Поле Packet Size действительно только в режиме инкрементной записи пакетов постоянной длины — в этом случае оно содержит размер одного пакета, измеряемых в секторах. Иначе здесь должно находится FF FF FFh.

Таблица 6.1. *Структура блока описателя трека с одним модулем описателя трека на конце*

Байт	Содержимое
0	"T"
1	"D"
2	"I"
3	Pre-gap length
4	
5	Type of Track Description Unit
6	Lowest Track Number
7	Highest Track Number

Таблица 6.1 *(окончание)*

Байт	Содержимое
8+00	Track Number
8+01	Write Method of the Track
8+02	Packet Size
8+03	
8+04	
8+05	Reserved
8+06	
8+07	
8+08	
8+09	
8+10	
8+11	
8+12	
8+13	
8+14	
8+15	
8+16	

Большинство копировщиков защищенных дисков (и Alcohol 120%/CloneCD в том числе) в отношении переходных областей ведут себя чрезвычайно некорректно и никогда не копируют область Pre-gap первого трека, либо оставляя ее "непрожженной" (Alcohol 120%), либо забитой нулями (CloneCD). Последующие же переходные области копируются вполне нормально.

Все переходные области, за исключением Pre-gap первого трека первой сессии диска, свободно доступны на секторном уровне и не вызывают никаких проблем при чтении. Но область Pre-gap первого трека первой сессии — особенная. Поскольку логический адрес первого значимого сектора диска принят за ноль (и это адрес первого сектора первого трека), то предшествующая ему область Pre-gap целиком лежит в отрицательных адресах. Это не вызывает никаких затруднений у команды `READ CD MSF`, принимающей в качестве аргументов абсолютные адреса, однако при использовании `READ CD` уже требуется совершенно иная система преобразования адресов (отрицательные LBA-адреса привод в упор не понимает). Она описана

в стандарте, однако разработчики копировщиков не всегда обращают на нее внимание, а может быть, им просто лень "топтать" кнопки? Кто знает? Но, как бы там ни было, первую область Pre-gap никто из них не читает, что позволяет нам использовать ее для хранения ключевой информации (на штампованных и дисках CD-R/RW) или же привязываться к конкретному TDB (на дисках CD-R/RW).

Сектор с адресом 00:00:00 (первый сектор Pre-gap) по стандарту читаться не обязан, т. к. у привода еще отсутствуют субканальные данные и он вынужден некоторое время заниматься их накоплением. На практике же, однако, штампованные и однократно записываемые диски CD-ROM/CD-R в зависимости от их качества и модели привода начинают читаться где-то со второго—десятого сектора, а до этого идут сплошные ошибки. С перезаписываемыми дисками ситуация обстоит намного хуже, и они зачастую содержат нечитабельные сектора даже в середине области Pre-gap!

Таким образом, для защиты лазерного диска от несанкционированного копирования мы можем использовать следующие приемы:

- размещать соседние треки вплотную, без переходных областей (такой диск не копируется Ahead Nero, но копируется Alcohol 120% и CloneCD);
- разместить в области Pre-gap первого трека диска ключевую информацию (такой диск копируется Ahead Nero, но не копируется ни Alcohol 120%, ни CloneCD);
- создать фиктивный трек в подлинном треке или в переходной области подлинного трека (такой диск не копируется Ahead Nero, но копируется CloneCD);
- разместить фиктивный трек в области Pre-gap первого трека (такой диск не копируется вообще ничем).

Добавление фиктивного трека приводит к искажению длины первого трека, т. к. теперь она вычисляется путем вычитания стартового адреса первого — подлинного — трека от стартового адреса второго — фиктивного — трека минус размер области Post-gap первого трека и Pre-gap второго (рис. 6.7). Допустим, мы имеем диск с одним треком (рис. 6.7, *а*) и добавляем в TOC фиктивную запись о втором, реально не существующем треке; как следствие этого, длина первого трека уменьшается на `sizeof(TRACK2) + sizeof(post-gap) + sizeof(pre-gap)`, причем между треком номер один и треком номер два образуется "дыра" в `sizeof(post-gap) + sizeof(pre-gap)` байт (рис. 6.7, *б*), которая штатными копировщиками не копируется! Поскольку номера треков в адресации дисков с данными вообще не участвуют, операционной системе по-прежнему доступно все содержимое исходного трека, включая и области Pre-gap и Post-gap, образовавшиеся на границе настоящего и фиктивного треков. Другими словами, диск будет нормально читаться на любом оборудовании и под любой операционной системой, но скопировать его смогут лишь те копировщики, которые копируют содержимое Pre-gap

и Post-gap, что по стандарту они делать не обязаны, т. к. с официальной точки зрения эти области не содержат ничего интересного. Как следствие — скопированный диск будет содержать "дыру" в 300 секторов, заполненных нулями. Такая "рана" способна угробить любой файл, а то и несколько файлов сразу!

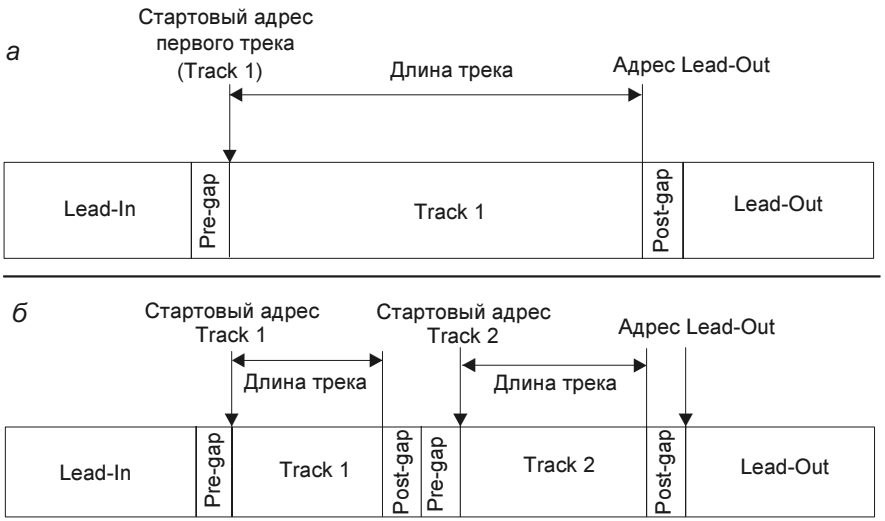


Рис. 6.7. Длина трека определяется как разность стартовых адресов следующего трека и стартового адреса самого этого трека минус размер области Post-gap

Еще большие перспективы открывает создание фиктивных треков в буферных областях Post-gap и Pre-gap. Длина образовавшегося трека, вычисленная по алгоритму `PreGap_len = min(&Lead-Out, &NexTrack - 150) - &MyTrack - 150`, становится резко отрицательной, дезориентируя такие копировщики, как Ahead Nero, CDRWin, Blind Write и даже самого Alcohol 120%. Причем если первые три копировщика дипломатично отказываются копировать защищенный диск, завершая свою работу вежливым сообщением об ошибке, то дезориентация Alcohol 120% сопровождается конкретным обрушением последнего.

А вот CloneCD копирует такие диски вполне корректно! На первый взгляд, это полностью обессмысливает данный защитный механизм (действительно, кому нужна защита, копирующаяся хотя бы одним-единственным широко распространенным копировщиком?). Однако не торопитесь выносить окончательное решение! Среди защитных механизмов существуют и такие, что легко копируются Alcohol 120%, но жестоко "обламывают" CloneCD. Совмещение нескольких защитных механизмов на одном диске равносильно

объединению разрозненных штатов (княжеств) в единую державу, стойчески сопротивляющуюся натиску злобных полчищ врагов. В этом свете, действия по размещению фиктивных треков в областях Pre-gap и Post-gap становятся весьма перспективной и к тому же чрезвычайно неконфликтной технологией защиты, претендующей на широкое распространение. Достаточно заманчивые перспективы, не так ли?

Итак, задача-минимум: добавить в файл IMAGE.CCD еще один трек, соответствующим образом скорректировав все связанные с ним поля. Эта, элементарная на первый взгляд, миссия на самом деле требует внесения в файл большого количества изменений. По меньшей мере, должны быть проделаны следующие операции:

- количество `TocEntries` должно быть увеличено на единицу;
- поле `PMin`, принадлежащее `point 0A1h`, также должно быть увеличено на единицу;
- необходимо добавить новый `Entry` с подложным треком (для простоты можно скопировать `Entry` подлинного трека, слегка изменив его стартовый адрес);
- номера всех последующих треков должны быть увеличены на единицу (т. е. все последующие `point'ы` такие, что `64h > point > 00h`, должны быть перенумерованы, при этом предполагается, что диск содержит менее `63h` треков, т. к. максимальное количество треков жестко ограничено сверху);
- номера всех последующих `Entry` должны быть увеличены на единицу;
- если на диске имеется больше, чем одна сессия, то номера треков и `point'ов A0h/A1h` последующих сессий должны быть увеличены на единицу (`point A0h` первой сессии увеличивать не нужно);
- в карту треков должен быть добавлен фиктивный трек, а номера всех последующих треков перенумерованы.

Для пущей важности (в смысле для лучшего соответствия стандарту) было бы нелишне скорректировать субканальные данные фиктивного трека, увеличив содержимое полей TNO (Track Number) каждого из них на единицу. Эта информация содержится в Q-канале подкода, который вместе со всеми остальными каналами находится в файле IMAGE.SUB. Каждый `0Dh + 60h * N` байт файла содержит поле TNO сектора N (строго говоря: жесткого соответствия между субканальными данными и секторами нет, поэтому эта формула весьма приблизительна). После внесения всех необходимых изменений, контрольная сумма каждой 16-байтовой субканальной секции должна быть пересчитана заново, в противном случае защищаемый диск перестанет работать. Для этой цели можно воспользоваться функцией `CalcSubChannelCRC` из динамической библиотеки newtrf.dll, входящей в состав прожигателя Ahead Nero.

Листинг 6.13. Пример секции субнанальных данных, выделенное полужирным шрифтом поле содержит номер текущего трека

```
0003060C:  41 02 01 00 00 06 00 03 │ 01 39 63 8A 00 00 00 00   A☻☺ ♠ ♥☺9cK
```

Если вам лень возиться с субканальными данными — ничего не трогайте и оставьте все как есть — в процессе чтения диска с данными номера треков никак реально не используются. Вот аудиодиски — другое дело. Там поле TNO используется для индикации текущего воспроизводимого трека и (иногда) для переключения между треками.

С другой стороны, наличие некорректных полей TNO существенно усиливает защищенность диска, поскольку лишь немногие из копировщиков способны копировать субканальные данные, да и у тех эта опция по умолчанию отключена (у CloneCD отключена точно).

Фиктивный трек в области данных подлинного трека

Самое простое (и, с точки зрения совместимости — самое надежное) это расположить фиктивный трек в области данных подлинного трека, выбрав стартовый адрес фиктивного трека так, чтобы до начала следующего трека оставалось не менее 350 секторов, которые пойдут на покрытие областей Post-gap/Pre-gap (для последнего трека сессии достаточно и 150 секторов, поскольку здесь не нужно тратиться на Pre-gap). Между стартовым адресом подлинного трека и стартовым адресом фиктивного трека также должно помещаться по меньшей мере 350 секторов, занимаемых постзазором подлинного и предзазором фиктивного трека. Нарушение этого правила приводит к усилению защиты, но и порождает определенные побочные эффекты, о которых мы поговорим позднее (см. *разд. "Фиктивный трек в Post-gap подлинного трека" и "Фиктивный трек в Pre-gap подлинного трека" этой главы*).

Сейчас же нас больше всего интересует, как именно добавить новое Entry в файл IMAGE.CCD, не нарушив его работоспособности. Алгоритм создания фиктивного трека, подробно рассмотренный в предыдущем разделе, все еще представляет собой абстрактную теорию, достаточно далеко отстоящую от практики. В ходе осуществления творческого замысла на нашем пути могут встретиться различные трудности, и нам придется сразиться с ними или просто обойти их.

Давайте рассмотрим самый сложный случай — многосессионный диск. Правила хорошего тона обязывают нас скорректировать не только содержимое той сессии, к которой добавляется новый трек, но и атрибуты всех последующих сессий, поскольку нумерация треков на диске — сквозная. Частая ошибка начинающих — перенумеровать треки. Перенумеровать-то они перенумеруют, а вот переустановить указатели на первый и последний трек каждой сессии — забывают. Как следствие этого, защищенный диск перестает

читаться совсем или читается неправильно (конкретное поведение зависит от "темперамента" привода, на который этот диск попадет).

Конкретный пример добавления нового трека в уже существующий приведен далее в листинге 6.14. По понятным причнам его наглядность оставляет желать лучшего (все-таки печатная книга — это вам не утилита WinDiff), но это все же лучше, чем совсем ничего. Первая и третья колонки, залитые серым цветом, содержат оригинальные значения редактируемого файла, вторая и четвертая — измененные. Непосредственно сами изменения выделены полужирным шрифтом.

Листинг 6.14. Создание фиктивного трека – трека номер два; все изменения выделены полужирным шрифтом, значение оригинальных полей залито серым цветом (нечетные колонки)

```
[CloneCD]
Version=3

[Disc]
TocEntries=12
Sessions=2
DataTracksScrambled=
0
CDTextLength=0

[Session 1]
PreGapMode=1
PreGapSubC=0

[Session 2]
PreGapMode=1
PreGapSubC=0

[Entry 0]
Session=1
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
```

```
[CloneCD]
Version=3

[Disc]
TocEntries=13
Sessions=2
DataTracksScrambled=0

CDTextLength=0

[Session 1]
PreGapMode=1
PreGapSubC=0

[Session 2]
PreGapMode=1
PreGapSubC=0

[Entry 0]
Session=1
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
```

```
[Entry 6]
Session=1
Point=0xc1
ADR=0x05
Control=0x04
TrackNo=0
AMin=4

ASec=120
AFrame=96
ALBA=26946
Zero=0
PMin=0
PSec=0
PFrame=0
PLBA=-150

[Entry 7]
Session=2
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
```

```
[Entry 7]
Session=1
Point=0xc1
ADR=0x05
Control=0x04
TrackNo=0
AMin=4

ASec=120
AFrame=96
ALBA=26946
Zero=0
PMin=0
PSec=0
PFrame=0
PLBA=-150

[Entry 8]
Session=2
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
```

Zero=0	Zero=0
PMin=1	PMin=1
PSec=0	PSec=0
PFrame=0	PFrame=0
PLBA=4350	PLBA=4350
[Entry 0]	[Entry 0]
Session=1	Session=1
Point=0xa0	Point=0xa0
ADR=0x01	ADR=0x01
Control=0x04	Control=0x04
TrackNo=0	TrackNo=0
AMin=0	AMin=0
ASec=0	ASec=0
AFrame=0	AFrame=0
ALBA=-150	ALBA=-150
Zero=0	Zero=0
PMin=1	PMin=1
PSec=0	PSec=0
PFrame=0	PFrame=0
PLBA=4350	PLBA=4350
[Entry 1]	[Entry 1]
Session=1	Session=1
Point=0xa1	Point=0xa1
ADR=0x01	ADR=0x01
Control=0x04	Control=0x04
TrackNo=0	TrackNo=0
AMin=0	AMin=0
ASec=0	ASec=0
AFrame=0	AFrame=0
ALBA=-150	ALBA=-150
Zero=0	Zero=0
PMin=1	**PMin=2**
PSec=0	PSec=0
PFrame=0	PFrame=0
PLBA=4350	**PLBA=-1**
[Entry 2]	[Entry 2]
Session=1	Session=1
Point=0xa2	Point=0xa2

Zero=0	Zero=0
PMin=2	**PMin=3**
PSec=0	PSec=0
PFrame=0	PFrame=0
PLBA=8850	**PLBA=-1**
[Entry 8]	**[Entry 9]**
Session=2	Session=2
Point=0xa1	Point=0xa1
ADR=0x01	ADR=0x01
Control=0x04	Control=0x04
TrackNo=0	TrackNo=0
AMin=0	AMin=0
ASec=0	ASec=0
AFrame=0	AFrame=0
ALBA=-150	ALBA=-150
Zero=0	Zero=0
PMin=2	**PMin=3**
PSec=0	PSec=0
PFrame=0	PFrame=0
PLBA=8850	**PLBA=-1**
[Entry 9]	**[Entry 10]**
Session=2	Session=2
Point=0xa2	Point=0xa2
ADR=0x01	ADR=0x01
Control=0x04	Control=0x04
TrackNo=0	TrackNo=0
AMin=0	AMin=0
ASec=0	ASec=0
AFrame=0	AFrame=0
ALBA=-150	ALBA=-150
Zero=0	Zero=0
PMin=3	PMin=3
PSec=24	PSec=24
PFrame=23	PFrame=23
PLBA=15173	PLBA=15173
[Entry 10]	**[Entry 11]**
Session=2	Session=2
Point=0x02	**Point=0x03**

ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01
Control=0x04	Control=0x04	Control=0x04	Control=0x04
TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0
AMin=0	AMin=0	AMin=0	AMin=0
ASec=0	ASec=0	ASec=0	ASec=0
AFrame=0	AFrame=0	AFrame=0	AFrame=0
ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150
Zero=0	Zero=0	Zero=0	Zero=0
PMin=0	PMin=0	PMin=3	PMin=3
PSec=29	PSec=29	PSec=1	PSec=1
PFrame=33	PFrame=33	PFrame=33	PFrame=33
PLBA=2058	PLBA=2058	PLBA=13458	PLBA=13458
[Entry 3]	[Entry 3]	[Entry 11]	**[Entry 12]**
Session=1	Session=1	Session=2	Session=2
Point=0x01	Point=0x01	Point=0xb0	Point=0xb0
ADR=0x01	ADR=0x01	ADR=0x05	ADR=0x05
Control=0x04	Control=0x04	Control=0x04	Control=0x04
TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0
AMin=0	AMin=0	AMin=4	AMin=4
ASec=0	ASec=0	ASec=54	ASec=54
AFrame=0	AFrame=0	AFrame=23	AFrame=23
ALBA=-150	ALBA=-150	ALBA=21923	ALBA=21923
Zero=0	Zero=0	Zero=1	Zero=1
PMin=0	PMin=0	PMin=22	PMin=22
PSec=2	PSec=2	PSec=14	PSec=14
PFrame=0	PFrame=0	PFrame=34	PFrame=34
PLBA=0	PLBA=0	PLBA=99934	PLBA=99934
	[Entry 4]	[Entry 5]	**[Entry 6]**
	Session=1	Session=1	Session=1
	Point=0x02	Point=0xc0	Point=0xc0
	ADR=0x01	ADR=0x05	ADR=0x05
	Control=0x04	Control=0x04	Control=0x04
	TrackNo=0	TrackNo=0	TrackNo=0
	AMin=0	AMin=162	TrackNo=0
	ASec=0	ASec=128	ASec=128
	AFrame=0	AFrame=140	AFrame=140
	ALBA=-150	ALBA=288590	ALBA=288590
	Zero=0	Zero=0	Zero=0
	PMin=22	PMin=97	PMin=97

	PSec=0	PSec=27	PSec=27
	PFrame=0	PFrame=21	PFrame=21
	PLBA=-1	PLBA=-11604	PLBA=-11604
[Entry 4]	**[Entry 5]**	[TRACK 1]	[TRACK 1]
Session=1	Session=1	MODE=1	MODE=1
Point=0xb0	Point=0xb0	INDEX 1=0	INDEX 1=0
ADR=0x05	ADR=0x05		
Control=0x04	Control=0x04		**[TRACK 2]**
TrackNo=0	TrackNo=0		**MODE=1**
AMin=2	AMin=2		**INDEX 1=0**
ASec=59	ASec=59		
AFrame=33	AFrame=33	[TRACK 2]	**[TRACK 3]**
ALBA=13308	ALBA=13308	MODE=1	MODE=1
Zero=3	Zero=3	INDEX 1=0	INDEX 1=0
PMin=22	PMin=22		
PSec=14	PSec=14		
PFrame=34	PFrame=34		
PLBA=99934	PLBA=99934		

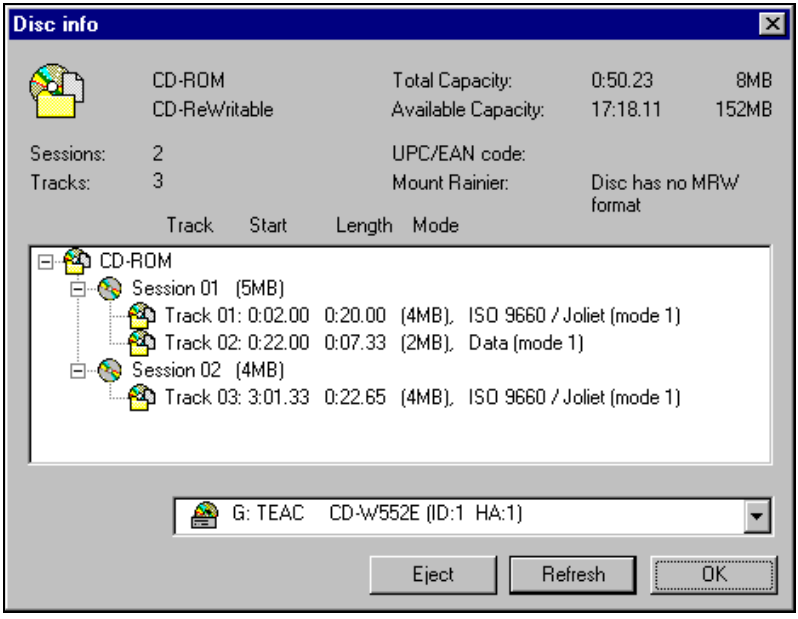


Рис. 6.8. В первой сессии находятся два трека, первый нормальный, второй — "рукотворный" фиктивный

Сохранив отредактированный файл IMAGE.CCD на диск, "залейте" полученный образ на болванку с помощью CloneCD или Alcohol 120%. Убедитесь, что защищенный диск нормально обрабатывается операционной системой и что к двум прежним трекам диска добавился еще один (рис. 6.8).

Попытка копирования защищенного диска штатными копировщиками (такими например, как Stomp Record Now или Ahead Nero) как будто бы проходит успешно, но при ближайшем рассмотрении между первым и вторым треком обнаруживается "дыра" из 300 секторов, заполненных нулями. Что ж! Этого следовало ожидать! Штатные копировщики слишком буквально понимают тот пункт стандарта, что утверждает, будто бы области Post-gap и Pre-gap не содержат никаких данных.

Изменяется и субканальная информация (рис. 6.15). Копировщики, не выполняющие фактического чтения Q-канала подкода копируемого диска и самостоятельно воссоздающие его с нуля, исправно маркируют субканальные данные порядковыми номерами соответствующих им треков, что позволяет легко отличить оригинал от его грубой копии.

Листинг 6.15. Демонстрация изменения субканальной информации при копировании диска

```
# читаем TOC защищенного диска для определения стартового адреса 2-го трека
$toc TEAC 0
00 14 01 00 00 00 00 00
00 14 02 00 00 00 02 A3 # стартовый адрес 2-го трека = 2A3h или 675 в дес. нотац.
00 14 03 00 00 00 34 92
00 14 AA 00 00 00 3B 45

# читаем субканальные данные из 2-го трека защищенного диска
$seek_and_Q TEAC 675
seek CD-ROM & read Q-subcode by KK
LBA - 02A3: 00 15 00 0C 01 14 01 01 00 00 02 A3 00 00 02 A3
# поле TNO 2-го трека содержит ^^ номер один

# копируем диск Ahead Nero/Easy CD Creator/Record Now!
# или CloneCD/Alcohol 120% без чтения субканальной информации

#  читаем субканальные данные из 2-го трека диска-копии
$seek_and_Q TEAC 675
seek CD-ROM & read Q-subcode by KK
LBA - 02A3: 00 15 00 0C 01 14 02 01 00 00 02 A3 00 00 00 00
# поле TNO 2-го трека содержит ^^ номер два
# субканальная информация изменилась!
```

Забавно, но MP3-файл, расположенный на оригинальном диске, даже будучи жестоко продырявленным, воспроизводится вполне нормально, негромко "булькая" в месте своего "ранения", что, конечно, неприятно, но все-таки не смертельно. Конечно, чем больше фиктивных треков содержит защищенный диск и чем теснее эти треки расположены друг к другу, тем сильнее несанкционированный дубликат будет отличаться от оригинала. Копия диска, нашпигованного фиктивными треками "по самую макушку", уже не играет, а свистит и булькает, вызывая у доморощенных пиратов смесь злости с недоумением. Вот так и рождаются легенды о неизбежном падении качества при копировании лазерных дисков. Разумеется, речь идет исключительно о дисках с MP3 или видеодисках. Диски с данными гораздо более ранимы, и если "дыра", оставленная копировщиком, попадет на исполняемый файл и/или архив, с вероятностью, близкой к единице, он окажется полностью выведеным из строя.

Впрочем, стойкость защиты данного типа очень и очень невелика и такой диск вполне успешно копируется копировщиками Alcohol 120% и CloneCD (конечно, при условии, что опция чтения субканальных данных включена), которые, судя по всему, вообще игнорируют стартовые адреса треков и читают всю читабельную область каждой сессии от конца Lead-In до начала Lead-Out целиком. Что ж! Тем хуже для тех, кто применяет эту защиту для затруднения копирования своих программ. Что же касается хваленого Blind Write, то он вообще не копирует такой диск, вылетая по "экскепшену" в (цитирую) "read engine". Чем ему не понравился фиктивный трек — загадка!?

Фиктивный трек в Post-gap подлинного трека

Разместить фиктивный трек в середине настоящего (как это и показано на рис. 6.8) — неинтересно. Лучше разместить фиктивный трек целиком в области Post-gap подлинного трека. Тогда, при попытке вычисления длины фиктивного трека, все копировщики сойдут с ума. Вспомним, что длина всякого нормального трека по стандарту равна: `min(&Lead-Out, &NexTrack - 150) - &MyTrack - 150`. Если расположить начало трека так, чтобы `min(&Lead-Out, &NexTrack - 150) < (&MyTrack - 150)`, то его расчетная длина окажется отрицательной и многие из копировщиков просто не поймут, что с таким треком вообще следует делать. К тому же, большинство копировщиков хранят длину треков в переменных типа `unsigned long`, а потому небольшое по модулю отрицательное число, ошибочно интерпретируемое процессором как беззнаковое, превращается в очень и очень большое положительное, и для записи "содержимого" фиктивного трека потребуется около 4 Гбайт свободного места на диске и еще столько же — на прожигаемой болванке.

Обратимся к `pointA2h`, хранящему адрес выводной области, и скопируем принадлежащие ему `PMin`, `PSec` и `PFrame` в соответствующие поля фиктивного

сектора, уменьшив `PFrame` последнего на некоторое значение (о том, как создаются фиктивные сектора, мы уже говорили, см. *два предыдущих раздела этой главы*).

Фиктивный трек в Pre-gap подлинного трека

Размещение фиктивного трека в области Pre-gap первого подлинного трека приводит к довольно интересным результатам, обсуждению которых не грех посвятить отдельный раздел. На первый взгляд, такая защита полностью аналогична предыдущей, с той лишь разницей, что теперь исказится адрес первого, а не второго трека. Да, это так, но лишь отчасти! Область Pre-gap первого трека — особенная. Мало того, что по стандарту она вообще не доступна для чтения (хотя некоторые приводы вроде бы ухитряются ее читать), так еще и LBA-адрес ее начала измеряется отрицательным числом! Вспомним, что адреса LBA связаны с абсолютными адресами следующим соотношением:

```
LBA = ((Min * 60) + Sec) * 75 + Frame - 150,
```

где `150` и есть `sizeof(pre-gap)`.

Абсолютный стартовый адрес первого нормального трека по стандарту должен быть равен 00:02:00 (что соответствует LBA-адресу `0h`), абсолютный стартовый адрес первой области Pre-gap — 00:00:00 (что соответствует LBA-адресу `-96h` и `-150` в десятичной нотации). Даже если разработчик копировщика использовал для хранения адресов знаковые переменные, это все равно ничего не меняет, поскольку аргументы команд `READ` и `READ CD` всегда представляют собой беззнаковые числа! К тому же, размещение второго трека в области Post-gap первого приводит к тому, что стартовый адрес второго трека становится меньше стартового адреса первого, к чему подавляющее большинство копировщиков просто не готово.

Скопировать содержимое первой области Pre-gap (в которой расположен фиктивный трек) — нельзя, да, собственно, и не нужно. Но всякий ли копировщик об этом знает? Если только его разработчики заранее не предусмотрели обработку такой ситуации, копировщик в зависимости от типа используемой им адресации либо выдаст ошибку чтения (абсолютная адресация), либо совершит очень далекое перемещение головкой по "сумасшедшему" LBA-адресу (LBA-адресация без проверки корректности адресов), либо же просто не будет знать, что ему с этим отрицательным адресом делать (LBA-адресация с проверкой корректности адресов). Забегая вперед, отметим, что с защитой данного типа (кодовое наименование "Шакал") способен справиться один лишь CloneCD.

Используя файл IMAGE.CCD, оставшийся от предыдущих экспериментов, давайте переместим начало фиктивного трека по абсолютному адресу 00:01:00, как это показано в листинге 6.16.

Листинг 6.16. Фиктивный трек в Post-gap подлинного трека, расположенный по адресу 00:01:00

```
[Entry 4]
Session=1
Point=0x02
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
PMin=00
PSec=01
PFrame=0
PLBA=-1
```

При открытии отредактированного файла IMAGE.CCD, копировщик CloneCD неправильно вычисляет длину первого трека (см. листинг 6.17), однако на "прожиге" болванки это обстоятельство никак не сказывается.

Листинг 6.17. CloneCD выдает неправильную информацию о длине первого трека

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 2
Track 1: Данные Mode 1, размер: 4.294.967.124 Кбайт
Track 2: Данные Mode 1, размер: 4899 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 3: Data, размер: 3939 Кбайт
```

Проверка показывает, что защищенный таким способом диск нормально читается на приводах NEC и TEAC, а ASUS "видит" лишь первый трек первой сессии диска, поэтому закладываться на вторую и все последующие сессии — неразумно да и небезопасно для своего здоровья (разъяренные пользователи при случае и побить могут).

При попытке скопировать защищенный диск штатными копировщиками, последние ведут себя довольно странно. Stopm Record Now! и Ahead Nero вообще отказываются читать такой диск, ругаясь на *Invalid Disk* и *Imvalid Track Mode* соответственно.

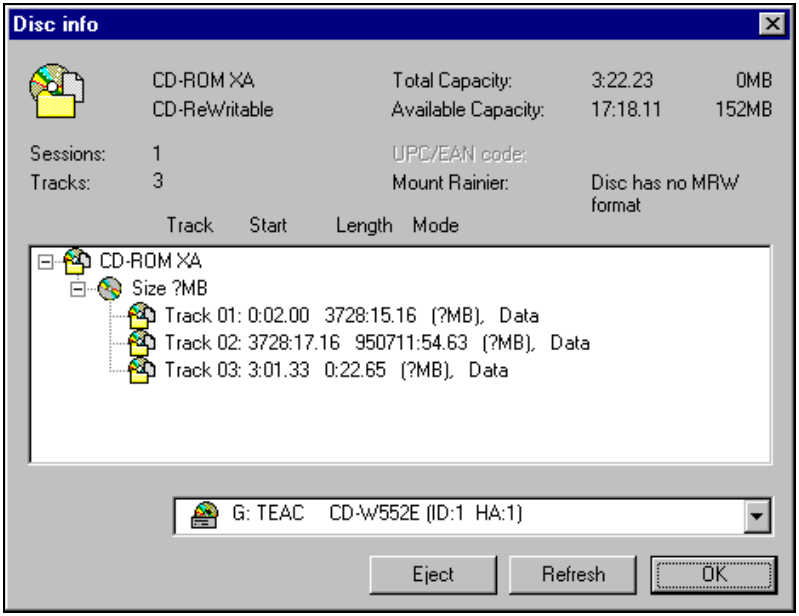


Рис. 6.9. Ahead Nero, встретив фиктивный трек в Pre-gap подлинного трека, запутался настолько, что не смог определить длину ни одного из треков и некорректно определил адрес второго — фиктивного трека

Столкнувшись с фиктивным треком в области Pre-gap, Ahead Nero совершенно дезориентируется и при попытке определения геометрии диска впадает в грубые ошибки (рис. 6.9). Ну, то что длина первого трека определяется неправльно — это нас и не удивляет, но вот насколько же нужно быть тупым, чтобы не суметь определить атрибуты всех остальных треков — это уже интересно! Стартовый адрес второго трека, равный 3728:17:16, со всей своей очевидностью указывает на то, что в качестве базового типа адресации Ahead Nero использует беззнаковые LBA-адреса, оперативно переводя их в MSF при необходимости. Поскольку беззнаковый LBA-адрес начала второго трека представляет собой очень большое положительное число, разница между стартовым адресом Lead-Out и стартовым адресом фиктивного трека вновь оказывается отрицательна, что окончально запутывает Ahead Nero, приводя его к катастрофически неверному результату. О причинах же неудачного определения типа третьего трека остается только

гадать. Наверное, это как-то связано с неправильным определением количества сессий: Ahead Nero увидел всего лишь одну сессию из двух. Только не спрашивайте меня почему, я этого все равно не знаю, а обращаться за разъяснениями в службу технической поддержки мне лень, да и не подписывался я сообщать разработчикам о "ляпах" в их программе. Пускай лучше думают головой, чем гонятся за деньгами.

Попытка сканирования поверхности диска утилитой Ahead Nero CD Speed на предмет поиска поврежденных секторов (**~Extra → ScanDisk**) приводит к тому, что окно программы просто "слетает". Тесты, такие как CPU Usage, Spin Up/Dows, также останавливаются с сообщением об ошибке. Таким образом, *размещение фиктивного трека в области Post-gap подлинного трека может служить эффективным средством борьбы с утилитами, определяющими качество диска, позволяя тем самым продавать дефектные диски под видом хороших.* Нет, это отнюдь не призыв. Напротив, это грустная констатация факта, что мир, в котором мы живем, не иделен и доверять нельзя никому и ничему. Ладно, не будем вдаваться в лирику, а лучше попробуем скопировать защищенный диск Alcohol 120%.

Если только флажок **Пропуск ошибок чтения** не был заблаговременно установлен, то программа Alcohol 120%, прервав чтение диска на 13%, невнятно ругнется на "Illegal Mode For This Trak" и предложит удалить незавершенные файлы. Как бы сказал чукча из анедота про оленей: "Тенденция, однако!"

Чтение диска с пропуском ошибок также не дает никакого результата. Достигнув сектора 2056 (предпоследний сектор в Post-gap подлинного трека), Alcohol 120% со всего маху "врезается" в область Lead-Out, широко "раскинув мозгами" (в смысле — разбросав их по всей территории).

Фиктивный трек в Lead-Out

Фиктивный трек, размещенный в области Lead-Out, не препятствует нормальному чтению диска, но серьезно озадачивает штатных копировщиков. Во-первых, длина фиктивного трека, вычисляемая как разность стартовых адресов Lead-Out и самого этого трека минус `sizeof(post-gap)`, в данном случае выражается отрицательным числом, а мы уже знаем, какую головную боль для копировщиков представляет всякая отрицательная величина! Во-вторых, содержимое области Lead-out в силу его недоступности на секторном уровне очень легко принять за bad sector'a со всеми вытекающими отсюда последствиями. В-третьих, предпоследний сектор области Post-gap подлинного трека (условно называемый нами *X-сектором*) по загадочным причинам не обрабатывается ни командой `READ CD`, ни командой `SEEK`, ни командой `READ HEADER`.

Обратимся к `point A2h`, хранящему адрес выводной области, и скопируем принадлежащие ему `PMin`, `PSec` и `PFrame` в соответствующие поля фиктивного сектора, увеличив `PSec` последнего на единицу или любую другую величину, находящуюся в границах области Lead-Out.

Пропустив традиционное ругательство программы CloneCD на ненормальную длину второго трека, мы "заливаем" модифицированный нами образ на диск CD-R/CD-RW. Как и в предыдущем примере, приводы NEC и TEAC "видят" все доступные им сессии, а ASUS только первую из них, поэтому на вторую и последующие сессии мы не должны закладываться.

Копировщики Stop Record Now! и Ahead Nero откажутся копировать такой диск, а **Disk info**, возвращенная последним, окажется некорректной (рис. 6.10). Конкретно: копировщик Ahed Nero не сможет определить MODE и длину третьего трека. Странно! Третий трек вообще не имеет никакого отношения к первым двум и для определения его атрибутов достаточно просто прочитать TOC.

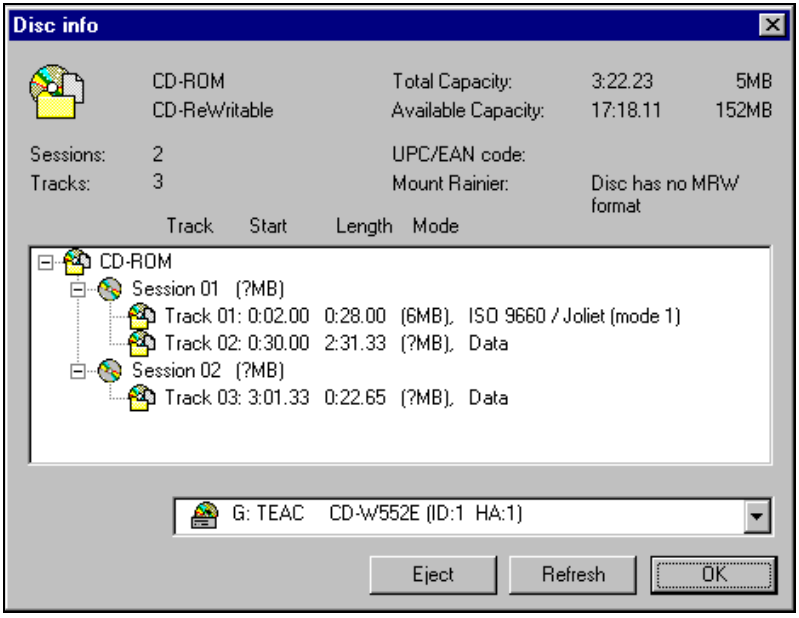


Рис. 6.10. Ahead Nero неправильно определил длину второго — фиктивного — трека и, что удивительно, не справился с определением длины и mode третьего трека, лежащего совсем в другой сессии

Попытка копирования защищенного диска с помощью программы Alcohol 120% ни к чему хорошему не приводит. Если флажок **Пропуск ошибок чтения** был сброшен, то на экране появляется сообщение "Ошибка: [05/64/00] – Illegal Mode For This Track" и процесс создания образа аварийно прерывается (рис. 6.11).

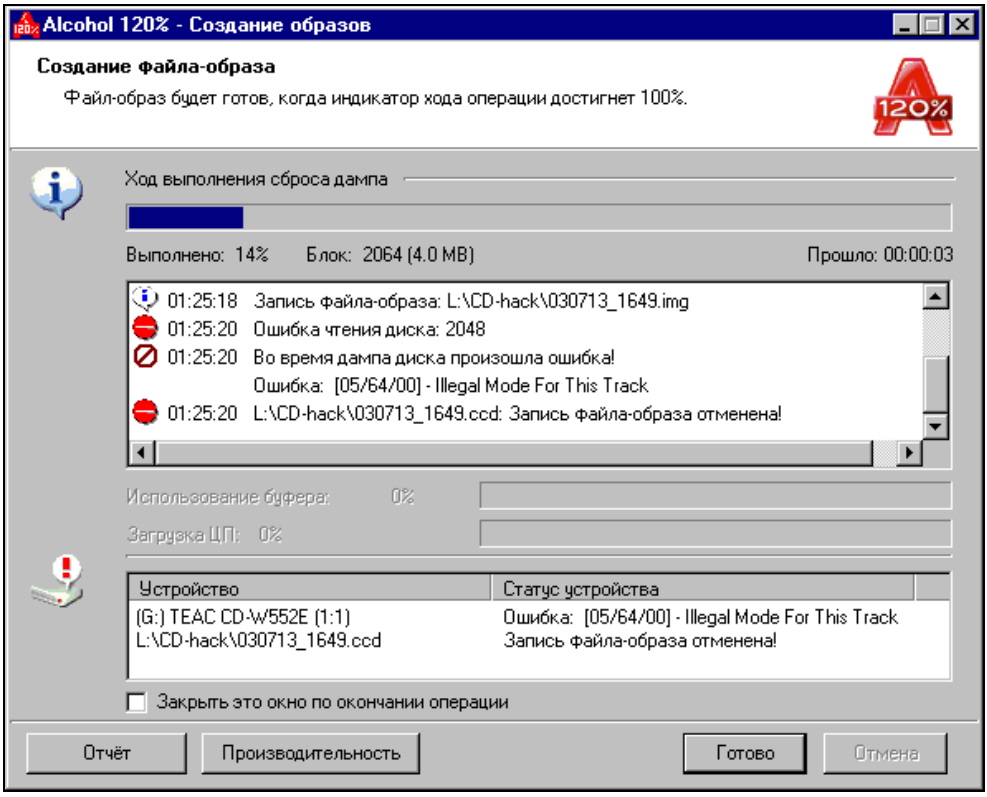


Рис. 6.11. Alcohol 120% без пропуска ошибок

Листинг 6.18. Реакция Alcohol 120% на фиктивный трек в области Lead-Out (режим пропуска ошибок выключен)

```
01:25:17 Информация о процессоре:
         Pentium III (0.18 um) 256KB OnDie L2 Cache (736 MHz)
01:25:17 Дамп диска: (G:) TEAC CD-W552E (1:1)
01:25:18 Режим чтения: Режим RAW
01:25:18 Информация об источнике:  Сессия: 2, Трек: 3, Длина: 29.6 MB / 003:22:23
01:25:18 Запись файла-образа: L:\CD-hack\030713_1649.img
01:25:20 Ошибка чтения диска: 2048
01:25:20 Во время дампа диска произошла ошибка!
01:25:20 Ошибка:  [05/64/00] - Illegal Mode For This Track
01:25:20 L:\CD-hack\030713_1649.ccd: Запись файла-образа отменена!
```

Если же режим пропуска ошибок чтения был включен (листинг 6.19), то создание образа как будто бы проходит успешно, пускай не без "пулеметной очереди" сбойных секторов, начинающихся с адреса 2058 и заканчивающихся адресом 2172 (очевидно, что эти сектора принадлежат области Pre-gap/Post-gap фиктивного трека, тщетно "пытаемой" Alcohol 120%).

Листинг 6.19. Реакция Alcohol 120% на фиктивный трек в области Lead-Out (режим пропуска ошибок включен)

```
01:32:11 Информация о процессоре:
         Pentium III (0.18 um) With 256 KB On-Die L2 Cache  (736MHz)
01:32:11 Дамп диска: (G:) TEAC CD-W552E (1:1)
01:32:12 Режим чтения: Режим RAW , Пропуск ошибок чтения
01:32:12 Информация об источнике:  Сессия: 2, Трек: 3, Длина: 29.6 MB / 003:22:23
01:32:12 Запись файла-образа: L:\CD-hack\030713_1649.img
01:32:21 Ошибка чтения диска: 2056
01:32:21 Ошибка чтения диска: 2058
01:32:21 Ошибка чтения диска: 2059
01:32:21 Ошибка чтения диска: 2060
…
01:32:22 Ошибка чтения диска: 2169
01:32:22 Ошибка чтения диска: 2170
01:32:22 Ошибка чтения диска: 2171
01:32:22 Ошибка чтения диска: 2172
01:32:32 L:\CD-hack\030713_1649.ccd: Запись файла-образа завершена!
01:32:32 Дамп диска завершен!
```

На что он надеялся — неизвестно, но образ, созданный Alcohol 120%, оказывается неработоспособным и выдает ошибку `Sector not fount. Cannot read folder contents` (рис. 6.12).

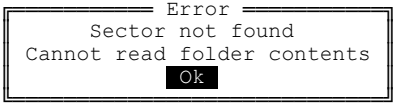


Рис. 6.12. Диск, скопированный Alcohol 120%, становится нечитаемым, и при попытке просмотра его содержимого операционная система начинает жутко материться

Зато CloneCD копирует такой диск без каких-либо ошибок, и его дубликат оказывается вполне работоспособен, однако он все-таки отличается от оригинала. Чтение X-сектора командой `READ CD` по-прежнему возвращает ошибку (листинг 6.20), но вот позиционирование головки командой `SEEK` и чтение

заголовка командой READ HEADER (листинг 6.21) выполняется нормально (напоминаем, что X-сектор оригинального диска не обрабатывался ни одной из этих команд!).

Листинг 6.20. Попытка чтения X-сектора оригинального диска командой READ CD дает ошибку Sense Key == 3 MEDIUM ERROR (выше), дубликат диска, полученный CloneCD, ведет себя аналогично (ниже)

```
>cd_raw_read.exe 1.1 2056 1
-ERR:F0 00 03 00 00 00 00 0A 00 00 00 00 11 00

>cd_raw_read.exe 1.1 2056 1
-ERR:F0 00 03 00 00 00 00 0A 00 00 00 00 11 00
```

Листинг 6.21. Попытка чтения заголовка оригинального диска командой READ HEADER дает ошибку Sense Key == 5 ILLEGAL REQUEST (выше), в то время как дубликат диска, полученный CloneCD, обрабатывается нормально (ниже)

```
>read.header.exe 1.1 2056 2056
READ HEADER (44h) SCSI/ATAPI commad demo by KK
-ERR:f0 00 05 00 00 00 00 0a 00 00 00 00 64 00

>read.header.exe 1.1 2056 2056
READ HEADER (44h) SCSI/ATAPI commad demo by KK
LBA:0808h --> MSF:00:1D:1F (MODE-1 [L-EC symb])
```

Попытка позиционирования на X-сектор оригинального диска командой SEEK, с последующим чтением субканальной информации командой READ SUBCHANALL, приводит к тому любопытному эффекту (листинг 6.22): привод не диагностирует ошибку, но и не перемещает оптическую головку, возвращая с командой READ SUBCHANALL субканальные данные с места ее предыдущего позиционирования (выше). Дубликат диска, полученный программой CloneCD, напротив, никак не препятствует позиционированию головки на X-сектор, успешно возвращая субканальные данные последнего (ниже).

Листинг 6.22. Попытка позиционирования на X-сектор оригинального диска командой SEEK, с последующим чтением субканальной информации командой READ SUBCHANALL (выше). Дубликат диска, полученный CloneCD (ниже)

```
>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 05 83 00 00 05 83
```

```
>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 08 08 00 00 08 08
```

Таким образом, *помещение фиктивного трека в область Lead-Out вкупе с обработкой X-сектора командами* `READ HEADER` *и* `SEEK/READ` `SUBCHANALL` *позволяет надежно отличить оригинальный диск от его копии* (защита с кодовым наименованием *"Волк"*).

Как скопировать "Волка"? При записи отредактированного образа на диск, CloneCD выдаст следующую информацию о его геометрии (листинг 6.23).

Листинг 6.23. Встретив диск с фиктивных треков в области Post-gap, CloneCD неправильно вычисляет его длину (выделена полужирным шрифтом)

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 2
Track 1: Данные Mode 1, размер: 299397 Кбайт
Track 2: Данные Mode 1, размер: 4294672626 Кбайт
```

По скромному мнению CloneCD, длина второго трека составляет целых 4 294 672 626 Кбайт или *4 Тбайт*! К счастью, это ничуть не мешает прожигу диска, и если все было сделано правильно, то защищенный диск должен нормально обрабатываться операционной системой.

А вот Ahead Nero (рис. 6.13) (и подавляющее большинство остальных копировщиков) поведет себя иначе, наотрез отказываясь "переваривать" фиктивный трек и аварийно завершая свою работу с криком "Illegal track mode" (неверный тип трека) или что-то типа того.

Копировщик CRWin по не совсем понятным причинам отказывается определять тип фиктивного трека (рис. 6.14) (хотя тип трека явно прописан в заголовках каждого из принадлежащему ему секторов) и препятствует его select'у, то есть, попросту говоря, выделению курсором, без которого трек извлекаться ни за что не будет. Извлечение первого — вполне нормального трека — также прерывается сообщением об ошибке.

А теперь попытаемся скопировать защищенный диск программой Alcohol 120%. Если флажок пропуска ошибок не был заблаговременно установлен, то Alcohol 120%, невнятно "ругнувшись" на `Illegal Mode For This Track`, просто прервет чтение диска в самом начале снятия образа (листинг 6.24).

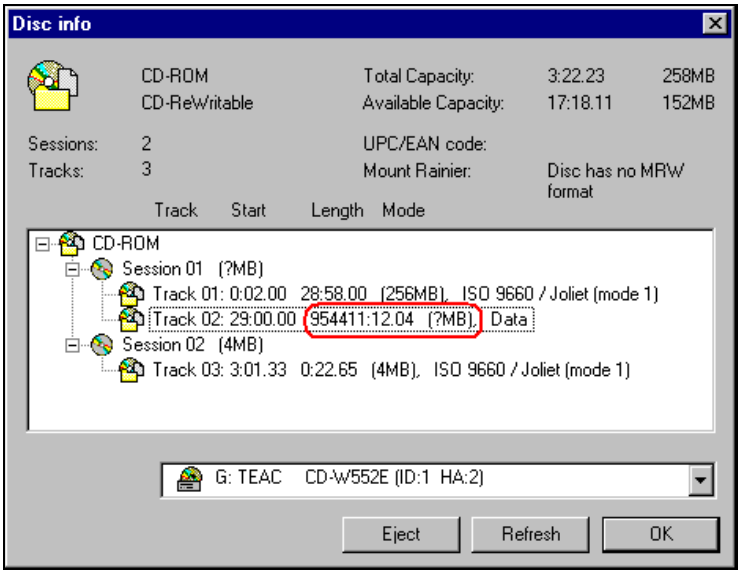


Рис. 6.13. Информация, выдаваемая Ahead Nero при анализе геометрии защищенного диска, чудовищная ошибка определения длины фиктивного трека препятствует его нормальному копированию

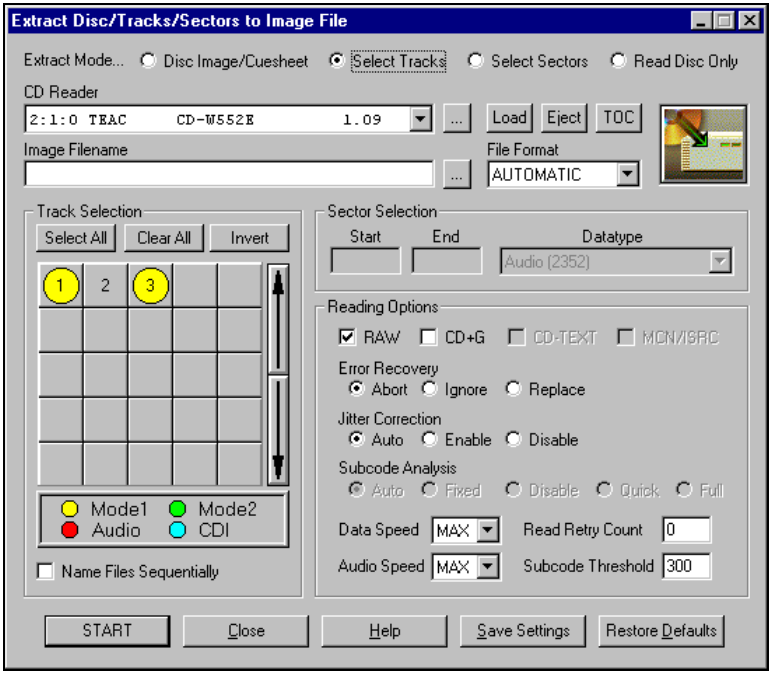


Рис. 6.14. Копировщик CDRWin отказывается определить тип фиктивного трека

Листинг 6.24. Alcohol 120% без пропуска ошибок

```
02:40:38 Информация о процессоре: Pentium III (0.18um) 256KB OnDie L2Cache (736MHz)
02:40:38 Дамп диска: (G:) TEAC CD-W552E (1:1)
02:40:46 Режим чтения: Режим RAW
02:40:46 Информация об источнике:  Сессия: 2, Трек: 3, Длина: 29.6 MB / 003:22:23
02:40:53 Запись файла-образа: L:\CD-hack\030713_1649.img
02:40:58 Ошибка чтения диска: 2048
02:40:58 Во время дампа диска произошла ошибка!
02:40:58 Ошибка:  [05/64/00] - Illegal Mode For This Track
02:40:58 L:\CD-hack\030713_1649.ccd: Запись файла-образа отменена!
```

Хорошо, флажок **Пропуск ошибок чтения** установлен. Пытаемся скопировать защищенный диск опять. Благополучно достигнув X-сектора (LBA-адрес которого составляет 2056), программа Alcohol 120% выдает сообщение об ошибке чтения и со всего маху "врезается" в выводную область, обдавая нас "пулеметной очередью" сбойных секторов (листинг 6.25). Достигнув 100%, он по непонятным причинам продолжает процесс чтения и дальше. Затем, видимо испугавшись, что его могут неправильно понять, блокирует кнопку **Отмена** и "зависает". То есть не то чтобы совсем "зависает" ("пулеметная очередь" сбойных секторов "раскаленными гильзами" по-прежнему продолжает устилать экран), но и остановить разбушевавшегося Alcohol 120% становится невозможно. Приходится его "убивать". Тоже мне, культура программирования!

Листинг 6.25. Alcohol 120% с пропуском ошибок (при достижении 100% он "зависает", но продолжает сообщать о сбойных секторах)

```
09:52:22 Информация о процессоре:
         Pentium III (0.18 um) 256KB OnDie L2 Cache (736MHz)
09:52:22 Дамп диска: (G:) TEAC CD-W552E (1:1)
09:52:29 Режим чтения: Режим RAW , Пропуск ошибок чтения
09:52:29 Информация об источнике:  Сессия: 2, Трек: 3, Длина: 29.6 MB / 003:22:23
09:52:38 Ошибка чтения диска: 0
09:52:39 Запись файла-образа: L:\CD-hack\030713_1649.img
09:52:53 Ошибка чтения диска: 2056
09:52:57 Ошибка чтения диска: 2057
09:52:57 Ошибка чтения диска: 2058
…
09:53:01 Ошибка чтения диска: 2707
09:53:01 L:\CD-hack\030713_1649.ccd: Запись файла-образа отменена!
09:53:01 Дамп диска отменён!
```

Третья попытка копирования защищенного диска начинается с перезапуска Alcohol 120% и установки флажка быстрого пропуска ошибочных блоков. На этот раз Alcohol 120% уже не виснет, но и не копирует, отказывать читать весь диск целиком (листинг 6.26). По мнению программы Alcohol 120%, сбойные сектора начинаются с первого (ну, в смысле нулевого) LBA-адреса до последнего. Чудеса, да и только!

Листинг 6.26. Alcohol 120% с быстрым пропуском ошибок

```
02:52:18 Информация о процессоре:
         Pentium III (0.18 um) With 256 KB On-Die L2 Cache  (736MHz)
02:52:18 Дамп диска: (G:) TEAC CD-W552E (1:1)
02:52:25 Режим чтения: Режим RAW , Быстрый пропуск ошибочных блоков
02:52:25 Информация об источнике:  Сессия: 2, Трек: 3, Длина: 29.6 MB / 003:22:23
02:52:25 Ошибка чтения диска: 0
02:52:26 Ошибка чтения диска: 1
02:52:26 Ошибка чтения диска: 2
02:52:26 Ошибка чтения диска: 3
```

Фиктивный трек, совпадающий с подлинным треком

Создание фиктивного трека, совмещенного с подлинным треком, приводит к тому, что длина первого трека обращается в ноль. Почему в ноль, а не в отрицательное число, ведь длина подлинного трека в данном случае равна: `&Track2 - &Track1 - sizeof(post-gap) - sizeof(pre-gap)`, что в конкретных цифрах выглядит так: `00:02:00 - 00:02:00 - 00:02:00 - 00:02:00 == -00:04:00` или в переводе в LBA-адреса: − 300.

Все дело в том, что адрес **00:02:00** — особенный. Трек, начинающийся с этого адреса, не имеет Pre-gap (вернее, не позволяет обрабатывать свою область Pre-gap нормальным образом), и потому вычисление фактической длины такого трека происходит в отдельной ветке программы, которая оказывается достаточно интеллектуальной для того, чтобы догадаться не вычитать `sizeof(post-gap)` из нулевого значения. Особенности реализаций конкретных копировщиков сейчас нам неинтересны, достаточно лишь знать, что подавляющее большинство из них вычисляют длину первого и всех последующих за ним треков вполне корректно (правда, от проблем копирования защищенного диска это их все равно не избавляет).

Внедренение X-сектора в подлинный трек (см. защиту типа "Шакал" в *разд. "Фиктивный трек в Pre-gap подлинного трека" этой главы*) приводит к следующему побочному эффекту: попытка чтения X-сектора командой `READ CD` возвращает ошибку, позиционирование головки на X-сектор командой `SEEK` проходит без ошибок, но и без перемещения самой головки. Чтение заголовка X-сектора командой `READ HEADER` вновь возвращает ошибку.

Забегая вперед, отметим, что копия диска, полученная с помощью CloneCD, читает заголовок и выполняет позиционирование головки без ошибок (головка при этом действительно перемещается), благодаря чему дубликат диска элементарно отличается от оригинала.

При открытии оригинального образа защищенного диска, CloneCD обнаруживат лишь второй — фиктивный — трек в первой сессии, "в упор не видя" настоящего (листинг 6.27). К счастью, несмотря на это, "прожиг" диска проходит успешно.

Листинг 6.27. Реакция CloneCD на фиктивный трек, совпадающий с настоящим треком

```
ИНФОРМАЦИЯ О СЕССИИ 1:

Размер сессии: 4726 Кбайт
Число треков: 2
Track 2: Данные Mode 1, размер: 4726 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 3: Данные Mode 1, размер: 3939 Кбайт
```

Просмотр геометрии диска подтверждает существование двух треков с одинаковыми стартовыми адресами, причем длина первого трека действительно равна нулю, а не отрицательному числу (рис. 6.15). Еще интереснее то, что тип первого трека определен как "ISO 9660/Joliet", что, мягко говоря, не совсем соответствует истине. Идентификатор типа файловой системы ISO 9660 содержится в шестнадцатом, а Joliet — в семнадцатом секторе трека. При условии, что длина трека равна нулю (а это так и есть!), возникает вполне законный вопрос: какое отношение эти сектора имеют к первому треку? Сдается мне, что процедура идентификации типа файловой системы вообще не проверяет диапазон доступных адресов на переполнение!

Приводы NEC и TEAC оказались достаточно интеллектуальными, чтобы суметь прочитать такой диск, а вот ASUS, к своему стыду, увидел лишь первую сессию. Следовательно, на вторую и все последующие сессии закладываться нельзя.

Попытка копирования защищенного диска программой Alcohol 120% традиционно проходит неуспешно (если режим попуска ошибок чтения не включен, Alcohol 120% вообще не хочет копировать такой диск, а в режиме пропуска ошибок чтения врезается в область Lead-Out и входит в дурной цикл, не позволяя себя прервать иначе, чем принудительным завершением процесса).

Копировщик CloneCD, как уже говорилось, обрабатывает фиктивный трек вполне нормально, но удаляет "Шакал" из области Post-gap, "благодаря" чему дубликат диска легко отличается от оригинала.

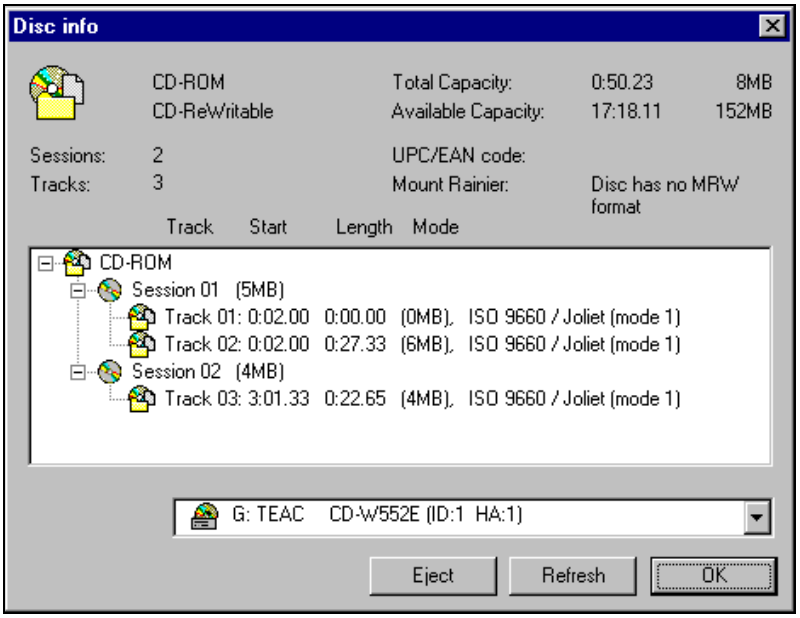


Рис. 6.15. Первый и второй треки имеют идентичные стартовые адреса, в результате чего длина первого трека обращается в ноль

Искажение нумерации треков

Согласно стандартру ECMA-130, информационные треки должны нумероваться последовательно, начиная от единицы и кончая последним треком диска *("Track Numbers 01 to 99 shall be those of the Information Tracks in the User Data area. Consecutive Information Tracks shall be numbered consecutively. The first Information Track of the user Data area of a disk shall have Track Number 01")*. Чувство здравого смысла разработчиков аппаратно-программного обеспечения придерживается такого же мнения, и потому считается, что всякая система может закладываться на то, что за треком номер один следует либо трек с номером два, либо область Lead-Out (трек с номером `AAh`). Но ведь нумерация треков может быть легко искажена с таким расчетом, чтобы за первым треком располагался девятый или даже еще один "первый" трек!

Испытания показывают, что подавляющее большинство приводов и копировщиков на перенумерацию треков реагируют крайне неадекватно, порой вообще отказываясь распознавать такой диск или же отображая треки с данными как аудио. Неудивительно, что копирование искаженных дисков

вызывает большие проблемы. Даже такие "зубры", как CloneCD и Alcohol 120%, оказываются не в состоянии разобраться с разметкой защищенного диска, и полученные копии оказываются чудовищно искажены или же вовсе не работоспособны.

Теоретически, диск с нарушенной нумераций треков должен копироваться без проблем, поскольку номера треков в абсолютной адресации вообще не участвуют и при работе с диском на секторном уровне копировщику достаточно лишь прочитать все содержимое диска от первого читаемого сектора до последнего, даже не подозревая о существовании треков вообще. Тем не менее, на практике все обстоит иначе, и подавляющее большинство копировщиков копируют диск по трекам, а не по секторам. Причем алгоритмы анализа TOC'а зачастую тупы до ужаса и не способны справиться даже с очевидными искажениями. Любые отклонения нумерации треков от нормальной вполне записываются на диск тем же CloneCD (за исключением трека, начинающегося с номера ноль, но об этом мы поговорим позже), но для чтения искаженного диска понадобится программа поумнее. Из всех известных мне программ на это способен лишь мой собственный копировщик (ну или посекторное копирование диска "вручную"), так что для защиты дисков эта методика — самое то!

Однако, чтобы защищенный диск не вызывал никаких конфликтов с оборудованием легальных пользователей, следует действовать очень осторожно, ни в коем случае не прикасаясь к нумерации треков первой сессии (т. к. зачастую это ведет к полной нечитаемости диска, подробности вы найдете далее в одноименном *разд. "Диск, начинающийся не с первого трека" этой главы*). Приятное исключение составляет лишь создание фиктивного трека с номером подлинного трека, — защита этого типа не конфликтует ни с каким доступным мне оборудованим и, судя по всему, не должна конфликтовать ни с каким оборудованием вообще, однако полной уверенности в этом у меня нет. Нумерацию треков второй сессии можно изменять более или менее безболезненно. В худшем случае привод просто не увидит треки второй сессии, но первая сессия будет доступна ему целиком.

Для изменения нумерации треков достаточно лишь изменить номер point'a, соответствующий оригинальному номеру искажаемого трека и скорректировать значение поля `PMin` point'a `A1h`, хранящего номер последнего трека диска (если этого не сделать, то мы получим защиту типа *"некорректный номер последнего трека"*). Так же следует исправить раскладку треков, содержащуюся в конеце CCD-файла. Следующий пример (листинг 6.28) демонстрирует, как создать разрыв между вторым и третьим треками, увеличив номер последнего с трех до девяти. Нечетные колонки, залитые серым цветом, — оригинальное содержимое CCD-файла, справа приведены измененные колонки (непосредственно сами изменения выделены полужирным шрифтом).

Листинг 6.28. Образование разрыва между вторым и третьим треками

[Entry 8]	[Entry 8]	[Entry 11]	[Entry 11]	[TRACK 1]	[TRACK 1]
Session=2	Session=2	Session=2	Session=2	MODE=1	MODE=1
Point=0xa1	Point=0xa1	Point=0x03 →	**Point=0x09**	INDEX 1=0	INDEX 1=0
ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01		
Control=0x04	Control=0x04	Control=0x04	Control=0x04	[TRACK 2]	[TRACK 2]
TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0	MODE=1	MODE=1
AMin=0	AMin=0	AMin=0	AMin=0	INDEX 1=0	INDEX 1=0
ASec=0	ASec=0	ASec=0	ASec=0		
AFrame=0	AFrame=0	AFrame=0	AFrame=0	[TRACK 3] →	**[TRACK 9]**
ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150	MODE=1	MODE=1
Zero=0	Zero=0	Zero=0	Zero=0	INDEX 1=0	INDEX 1=0
PMin=3 →	**PMin=9**	PMin=3	PMin=3		
PSec=0	PSec=0	PSec=1	PSec=1		
PFrame=0	PFrame=0	PFrame=33	PFrame=33		
PLBA=8850	**PLBA=-1**	PLBA=13458	PLBA=13458		

Поскольку большинство дисков с данными имеют лишь по одному треку на каждую сессию, то для перенумерации треков в границе одной сессии обычно приходится прибегать к созданию фиктивного трека в настоящем. Эта методика уже рассматривалась ранее (см. *разд. "Фиктивный трек в настоящем треке" этой главы*), и здесь мы не будем повторяться.

Некорректный стартовый номер первого трека

Искажение стартового номера первого трека — достаточно честный и стойкий прием защиты. Подавляющее большинство приводов вполне уверенно "заглатывают" диски, нумерация треков которых начинается с цифры, отличной от единицы. Предположим, что диск начинается с трека номер два.

Вернемся к оригинальному образу защищаемого диска и отредактируем файл IMAGE.CDD следующим образом (короче говоря, мы сдвинем номера всех треков на единицу, не забывая о том, что номер первого и последнего трека каждой сессии хранится в pointer'ах `0xA0` и `0xA1` соответственно и для корректной защиты диска они также должны быть модифицированы) (листинги 6.29—6.31). Серой заливкой выделены оригинальные значения, без заливки — измененные. Непосредственно сами изменения отмечены стрелкой и полужирным шрифтом.

Листинг 6.29. Изменение номера первого трека

[Entry 0]	[Entry 0]	[Entry 1]	[Entry 1]	[Entry 3]	[Entry 3]
Session=1	Session=1	Session=1	Session=1	Session=1	Session=1

Point=0xa0	Point=0xa0	Point=0xa1	Point=0xa1	**Point=0x1 ⇒**	**Point=0x2**
…	…	…	…	…	…
PMin=1 ⇒	**PMin=2**	**PMin=1 ⇒**	**PMin=2**	PMin=0	PMin=0
PSec=0	PSec=0	PSec=0	PSec=0	PSec=2	PSec=2
PFrame=0	PFrame=0	PFrame=0	PFrame=0	PFrame=0	PFrame=0

Листинг 6.30. Изменение номера второго трека

[Entry 7]	[Entry 7]	[Entry 8]	[Entry 8]	[Entry 10]	[Entry 10]
Session=2	Session=2	Session=2	Session=2	Session=2	Session=2
Point=0xa0	Point=0xa0	Point=0xa1	Point=0xa1	**Point=0x2 ⇒**	**Point=0x3**
…	…	…	…	…	…
PMin=2 ⇒	**PMin=3**	**PMin=2 ⇒**	**PMin=3**	PMin=0	PMin=0
PSec=0	PSec=0	PSec=0	PSec=0	PSec=2	PSec=2
PFrame=0	PFrame=0	PFrame=0	PFrame=0	PFrame=0	PFrame=0

Листинг 6.31. Изменение карты

[TRACK 1]⇒	[TRACK 2]	[TRACK 2]⇒	[TRACK 3]
MODE=1	MODE=1	MODE=1	MODE=1
INDEX 1=0	INDEX 1=0	INDEX 1=0	INDEX 1=0

Внимание

В параметрах профиля обязательно должен быть установлен флажок **Не восстанавливать субканальные данные**, в противном случае данные о номерах треков в Q-канале подкода не будут соответствовать TOC, что чревато ухудшением совместимости.

Запишем измененный образ на диск и из рекордера выпрыгнет вполне симпатичный защищенный диск, нумерация треков которого начинается с цифры два (рис. 6.16). Такие приводы, как ASUS или TEAC, покажут все его треки, нормально отобразят оглавление диска средствами операционной системы и уж тем более без колебаний согласятся работать с защитой на секторном уровне. Правда, NEC обламывается с таким диском по полной программе — при первой же попытке обращения (не важно, на секторном уровне она производится или нет), привод, издавая негромкие щелкающие звуки, впадает в дурной ступор, из которого его выводит только кнопка Eject. Увы, совместимость с нестандартным оборудованием у защит этого типа достаточно далека от идеала.

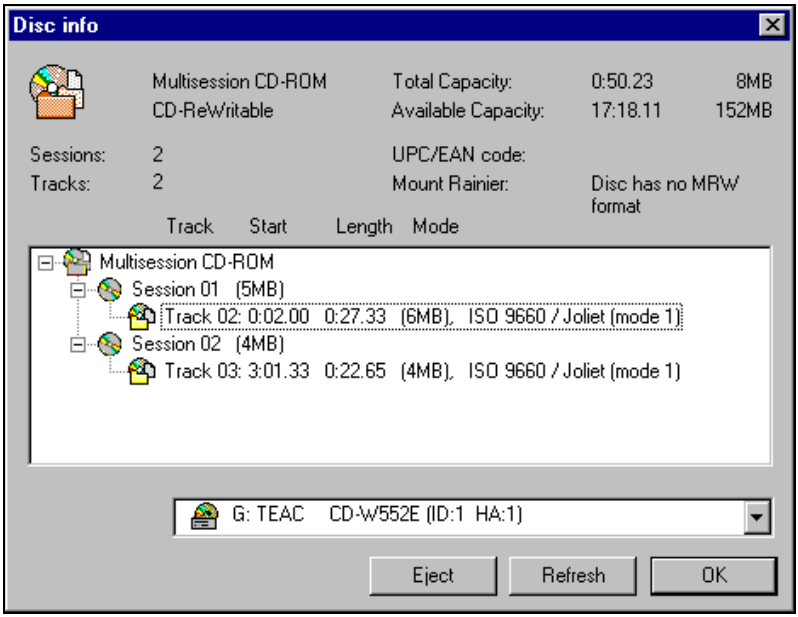


Рис. 6.16. Нумерация треков диска начинается с цифры 2

Посмотрим, какова окажется ее стойкость к копированию. Штатные копировщики (Stomp Record Now, Ahead Nero) отказываются признать такой диск хорошим лазерным диском и даже не пытаются его скопировать! Alcohol 120% из-за какой-то внутренней ошибки просто выдает сообщение о критической ошибке доступа и "отлетает" в мир иной, не реагируя ни на мышь, ни на клавиатуру. Зато CloneCD копирует такой диск вполне успешно (правда, по непонятной причине пропускает point `0xC1`, хранящий копию ATIP, и это обстоятельство можно с лихвой использовать для привязки к оригинальному диску).

Тот факт, что защиту не копируют текущие версии Alcohol 120%, еще не дает нам поводов для самодовольства, ибо в данном случае речь идет не о концептуальном архитектурном ограничении копировщика, а о досадной программистской ошибке. В любой момент ее могут исправить, и тогда наша защита окажется совсем беззащитной! И, чтобы ее усилить, придется удалиться от буквы стандарта на довольно приличное расстояние, совершив рассогласование номеров каждого из треков с номерами pointer'ов `0xA0` и `0xA1`. Пусть pointer'ы `0xA0` и `0xA1` первой сессии указывают на 1, а pointer'ы `0xA0` и `0xA1` второй сессии — на 2, в то время как нумерация самих треков начинается с двойки.

Такой диск на высоком уровне не способен читать ни один из известных мне приводов (даже ASUS "видит" лишь первую сессию), однако на секторном уровне с ним работать вполне можно (но только не на NEC'е!), поэтому

такая защита подходит лишь для двух дисковых программ. Первый диск — обычный, второй — защищенный, и чтение содержащихся на нем данных осуществляется непосредственно самим приложением на секторном уровне.

Копировщики защищенных дисков уже не справляются с такой защитой: и для Alcohol 120% и CloneCD диск представляется абсолютно пустым. Alcohol 120% не видит ни одной из сессий, а CloneCD на чистейшем русском языке признается, что

```
ИНФОРМАЦИЯ О CD В ДИСКОВОДЕ:
Невозможно получить информацию о CD.  Диск пуст?
```

Но мы-то с вами знаем, что диск не пуст! И в его копировании нет ничего невозможного! При условии, что используемый вами привод вообще поддерживает такие диски, достаточно лишь проанализировать TOC, который, как уже неоднократно говорилось, можно прочитать командой `READ TOC` (opcode: `0x43`; format `0x2`) и "сграбить" содержимое всех треков на секторном уровне, после чего остается лишь сформировать образ для "прожига" диска. Вообще удивительно, что копировщики защищенных дисков до сих пор не научились копировать такие диски автоматически, — ведь ничего трудного в этом нет.

Дважды одинаковый трек

Стойкая, несложная в реализации, хорошо совместимая с оборудованием и при всем этом в высшей степени элегантная защита. Все, что нам требуется сделать, так это изменить номер второго трека на первый, в результате чего на диске образуются два трека с идентичными номерами, но совершенно различным содержимым (не забудьте также скорректировать point'ы `0xA0` и `0xA1` и принудительно отключить восстановление субканальных данных!).

Защищенный диск нормально читается на всех известных мне приводах, однако не копируется ни одним доступным мне копировщиком (за исключением моих собственных копировщиков, разумеется). Два "первых" трека смотрятся довольно забавно (рис. 6.17) и не создают никаких проблем ни для самого привода, ни для операционной системы. Микропрограмма приводов обычно ищет лишь первый трек, который по наивному мнению некоторых разработчиков всегда имеет номер "один", а номера всех остальных — игнорирует. Операционная система, а точнее, файловая система лазерных дисков, также адресует треки не по их номерам, а по абсолютным секторным адресам, поэтому номера всех треков, кроме первого, как бы выпадают из поля зрения компьютера. Разумеется, сказанное относится только к дискам с данными.

Теоретически, копирование таких дисков не должно вызывать каких-либо проблем. Действительно, содержимое треков прекрасно читается на секторном уровне, номера треков в абсолютной адресации вообще не участвуют, и грамотно спроектированному копировщику всего-то и остается: прочитать

TOC и извлечь с диска содержимое всех его треков, невзирая на их номера. Конечно, если недальновидный разработчик решил помещать данные трека N в ячейку массива с индексом N, то наличие двух (или более) треков с идентичными номерами разобьют этот хлипкий алгоритм в пух и прах.

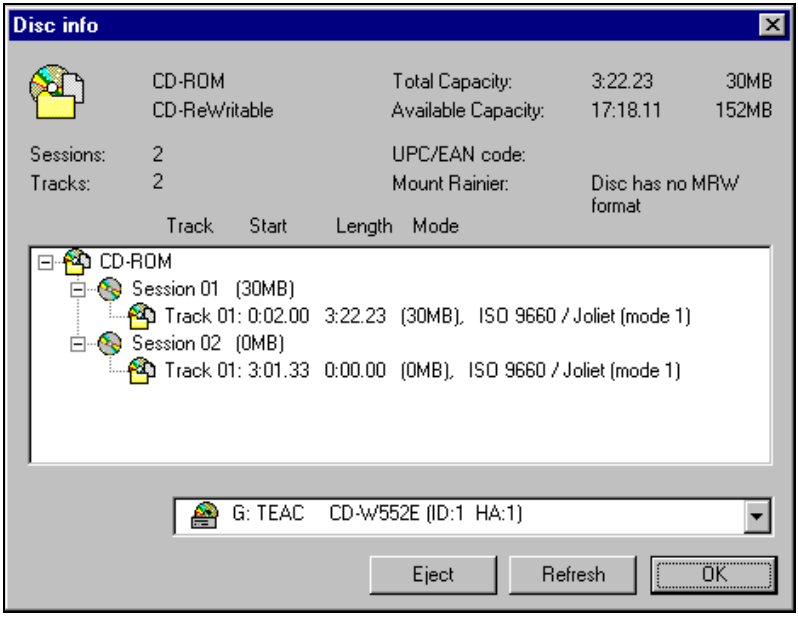


Рис. 6.17. Защищенный диск, содержащий два первых трека

Штатные копировщики со всем естеством своей штатной натуры копировать такие диски наотрез отказываются, ссылаясь на "Invalid disk" (Stomp Record Now!) и (Ahead Nero). Alcohol 120% достаточно корректно справляется с анализом геометрии диска (листинг 6.32), однако здесь не обходится без двух "но".

Листинг 6.32. Данные о защищенном диске (слева) и данные его образа, созданного Alcohol 120% (справа)

```
ИНФОРМАЦИЯ О ФАЙЛЕ-ОБРАЗЕ:

Число сессий: 2
Занято на диске: 34850 Кбайт
Секторов: 15173
Время: 03:22:23 (мин:сек:кадр)
```

```
ИНФОРМАЦИЯ О ФАЙЛЕ-ОБРАЗЕ:

Число сессий: 2
Занято на диске: 34850 Кбайт
Секторов: 15173
Время: 03:22:23 (мин:сек:кадр)
```

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 1
Track 1: Данные Mode 1,
размер: 4726 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 1: Данные Mode 1,
размер: 3939 Кбайт
```

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 1
Track 1:Данные Mode 1,
размер: 4726 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 1:Данные Mode 1,
размер: 3939 Кбайт
```

Во-первых, карта треков, составленная Alcohol 120%, состоит сплошь из одних ошибок и "не лезет ни в какие ворота". Вместо двух треков, Alcohol 120% занес сюда только один, зато продублировал индексы, окончательно запутавшись в их номерах и значениях (листинг 6.33).

Листинг 6.33. Карта треков защищенного диска (слева) и карта диска, составленная Alcohol 120% (справа)

```
[TRACK 1]
MODE=1
INDEX 1=0

[TRACK 1]
MODE=1
INDEX 1=0
```

```
[TRACK 1]
MODE=1
INDEX 0=-13608
INDEX 1=-13458
```

Во-вторых, запись образа защищенного диска выполняется довольно необычным образом. Индикатор прогресса по мере достижения 100% отметки не останавливает свой бег, как это бывает обычно, а упорно продолжает двигаться вперед. Затем, словно спохватившись, прогресс сбрасывается в нуль и процесс записи начинается сначала (вероятно, это начинает "прожигаться" второй первый трек). К его концу привод начинает жалобно стучать, тем не менее запись завершается без ошибок, вот только попытка прочесть оглавление записанного диска крепко "завешивает" привод и вызывает целую симфонию повизгивающих звуков. Короче говоря, Alcohol 120% облажался на все свои 120%.

Копировщик CloneCD уже на этапе анализа защищенного диска выдает полную лажу, теряя второй трек вместе с принадлежащей ему сессией. К тому же, по непонятной для простых смертных причине CloneCD уменьшает объем занятого пространства примерно на 3900 Кбайт (34850 Кбайта против 30911 Кбайт), также уменьшается и число секторов. Несмотря на то, что защищенный диск использует все 15173 сектора, CloneCD видит лишь

13458 из них. Куда девались остальные 1688 секторов — непонятно. И это ведь информация, полученная анализом TOC, который честно сообщает оригинальный объем и оригинальное количество секторов! Ладно бы, емкость диска уменьшилась на величину потерянного трека (это более или менее понятно), так ведь нет! От диска отрезается совершенно произвольная величина!!! Да, впрочем, смотрите сами (листинг 6.34).

Листинг 6.34. Данные о защищенном диске (слева) и данные его образа, созданные CloneCD (справа)

```
ИНФОРМАЦИЯ О ФАЙЛЕ-ОБРАЗЕ:

Число сессий: 2
Занято на диске: 34850 Кбайт
Секторов: 15173
Время: 03:22:23 (мин:сек:кадр)

ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 1
Track 1: Данные Mode 1,
размер: 4726 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 1: Данные Mode 1,
размер: 3939 Кбайт
```

```
ИНФОРМАЦИЯ О CD В ДИСКОВОДЕ:

Число сессий: 1
Занято на диске: 30911 Кбайт
Секторов: 13458
Время: 02:59:33 (мин:сек:кадр)

ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 30911 Кбайт
Число треков: 1
Track 1:Данные Mode 1,
размер:30911 Кбайт
```

Но это еще только цветочки! При попытке копирования диска CloneCD реально считывает с него лишь первые 15%, а затем врезается в огромный массив нечитаемых (с его точки зрения!) секторов. Начало этого массива (или, можно сказать, даже материкового щита) в аккурат совпадает с адресом области Lead-Out, прямым текстом записанной в TOC. Зачем CloneCD понадобилось читать область Lead-Out — непонятно. Короче, события развиваются почти по Пушкину: "Уж полночь близится, а Германа все нет". В смысле — проходит час, за ним другой, а CloneCD во всю "молотит" нечитаемые сектора из области Lead-Out и, судя по всему, прекращать это занятие и не собирается (рис. 6.18).

Хорошо! Ни CloneCD, ни Alcohol 120% по отдельности с защищенным диском не справляются. Но, может быть, их симбиоз хоть что-то даст?! Вспомним, Alcohol 120% достаточно корректно снял образ копируемого диска, вот только "прожечь" его бедолага никак не смог. CloneCD, напротив, "прожигает"

такие диски на ура (ведь именно с его помощью мы защищенный диск и создали!), но неумело фотографирует образ. А что если снять образ диска Alcohol 120% и подсунуть его CloneCD?! Да, при этом еще надо не забыть исправить карту треков на оригинальный вариант, который в силу своей тривиальности восстанавливается без труда. Впрочем, это можно и не делать, т. к. все равно ничего у нас не получится. Записать-то Alcohol 120% образ копировщик CloneCD запишет, но вот прочитать не сможет. После долгого стука головкой и натужного рева моторов, привод вывалит на экран содержимое одной лишь первой сессии. "А где же вторая?" — спросите вы. Чтобы выяснить это, достаточно заглянуть в IMAGE-файл, сформированный Alcohol 120%. Побайтовое сравнение секторов обоих треков покажет, что их содержимое идентично, т. е. второй трек затирает собой первый!

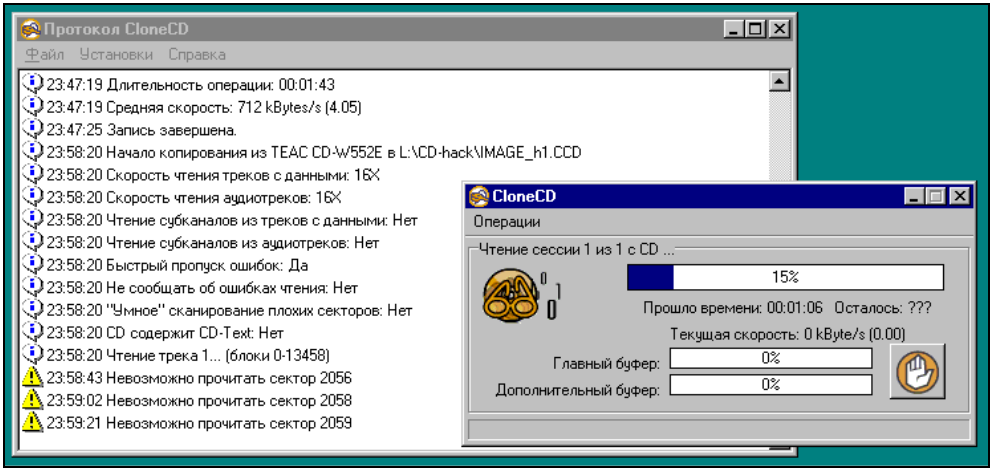


Рис. 6.18. CloneCD начинает читать сектора из области Lead-Out, страшно "удивляясь" факту их полной нечитаемости

Таким образом, предложенная защита действительно не копируется распространенными копировщиками защищенных программ, однако это еще не означает, что такой защищенный диск вообще нельзя скопировать в принципе! Еще как можно!!! Давайте, используя любую из утилит чтения секторов на "сыром" уровне, извлечем содержимое второго трека и запишем его в IMAGE-файл (найти, где начинается второй трек, можно по абсолютным адресам в его заголовке, достаточно открыть файл в HEX-редакторе и отыскать последовательность "`00 FF FF FF FF FF FF FF FF FF FF FF 00 00 02 00`", т. е. синхропоследовательность плюс адрес первого сектор 00:00:02 — первое вхождение придется на действительно первый трек, а следующее — на второй трек, затертый первым треком).

Проверка показывает, что на этот раз таким трудом скопированный диск работает безотказно, так что стойкость защиты отнюдь не так велика, как

хотелось бы разработчикам лицензируемых программ. Тем не менее, с пользователями, вооруженными типовым хакерским инструментарием, защита все же справляется и ввиду своей неплохой совместимости с оборудованием она в скором времени может стать весьма популярной.

Некорректный номер последнего трека

Искажение номера последнего трека (указатель `A1h` в TOC) подавляющее большинство приводов воспринимает крайне болезненно, поэтому данный способ защиты носит скорее академический, чем практический характер (по типу: так защищать свои диски не надо). Тем не менее, при условии соблюдения определенных предосторожностей пользоваться этой методикой все-таки можно.

Итак, берем уже известный нам образ диска IMAGE.CCD, находим в нем строку `Point=0xA1`, расположенную в сессии 1, и изменяем значение принадлежащего ей поля `PMin` с единицы на двойку. Тем самым мы заставляем привод считать, что номер последнего трека первой сессии равен двум, а не одному, как это имеет место быть в действительности.

```
[Entry 1]
Session=1
Point=0xa1
    …
PMin=1
PSec=0
PFrame=0
```

```
[Entry 1]
Session=1
Point=0xa1
    …
PMin=2
PSec=0
PFrame=0
```

При записи измененного образа на диск CloneCD выдает вполне корректную информацию о количестве треков, косвенно свидетельствуя о том, что он вообще не анализирует значения указателя `A1h`, а номер последнего трека определяет путем анализа указателей `01h`—`99h`. Последний встретившийся point и будет последним номером трека данной сессии.

Внимание

Point с наибольшим номером не обязательно является последним треком, т. к. нумерация треков может быть умышленно искажена с целью затруднения копирования диска.

Защищенный таким образом диск (точнее и правильнее было бы сказать "изуродованный таким образом диск") приводом ASUS показывает лишь свою первую сессию. NEC видит оба трека, но ввиду разбушевавшейся фантазии своих электронных цепей ошибочно распознает их тип как AUDIO, однако проигрывать их наотрез отказывается (жаль, а то бы мы услышали неповторимую симфонию Шума и Скрежета). Привод TEAC не опознает такой диск вообще. Короче, все указывает на то, что микропроцессорная

начинка приводов ведет себя совсем не так, как CloneCD, и вместо подсчета количества треков, напрямую считывает содержимое указателя `A1h`. И, если он оказывается искажен, поведение привода становится в высшей степени неадекватно. В общем, это плохая защита, и мы ее рассматривать не будем.

Гораздо лояльнее приводы относятся к искажению номера последнего трека второй сессии (в том смысле, что изуродованная вторая сессия никак не мешает чтению первой). Давайте в порядке свободного эксперимента возьмем оригинальную копию IMAGE.CCD и совершим над ней следующую "хирургическую" операцию (изменим содержимое поля `PMin`, принадлежащего point'у `A1h` второй сессии, с двойки на единицу, пытаясь убедить привод в том, что номер последнего трека второй сессии оканчивается на единицу, хотя можно выбрать и любое другое значение, например три или восемь):

```
 [Entry 8]
Session=2
Point=0xa1
    …
PMin=2
PSec=0
PFrame=0
```

```
 [Entry 8]
Session=2
Point=0xa1
    …
PMin=1
PSec=0
PFrame=0
```

Приводы ASUS и TEAC показывают лишь первую сессию искаженного диска, вторя же недоступна даже не секторном уровне. Привод NEC также видит только одну сессию — первую, но великодушно позволяет считывать вторую на секторном уровне, тем не менее на щедрость подобного рода нам закладываться нельзя, поскольку подавляющее большинство остальных приводов не столь благородны, как NEC.

Попробуем скопировать защищенный диск, предварительно изучив его геометрию с помощью любой подходящей программы (например, Ahead Nero). Смотрите (рис. 6.19), Ahead Nero видит оба трека диска, но второй — искаженный — трек представляется ему абсолютно пустым, что в общем-то недалеко от истины, т. к. на секторном уровне этот трек все равно не доступен. Попытка копирования диска тем же Ahead Nero приводит к копированию лишь первой его сессии, и хотя скопированный диск формально вполне работоспособен, защита может разгадать подмену элементарным анализом TOC, проверяя количество сессий, имеющихся на диске, и атрибуты второго трека. На диске, скопированном Ahead Nero, они, разумеется, не совпадут.

Копировщик CloneCD также обламывается с копированием. Во-первых, он теряет способность корректно распознавать границы сессии и при достижении конца первой сессии упорно пытается прочесть содержимое области Lead-Out на секторном уровне. Правда, процесс "обмолачивания" дефективных секторов протекает достаточно быстро, и пользователь даже не успевает заскучать. Вторая сессия по понятным причинам остается не скопированной,

и CloneCD корректирует TOC, выбрасывая из него всякие упоминания о последней (это во-вторых). Как следствие: TOC скопированного диска будет кардинально отличаться от TOC-оригинала, и защите не составит большого труда обнаружить факт несанкционированного копирования. Alcohol 120% также копирует только первую сессию.

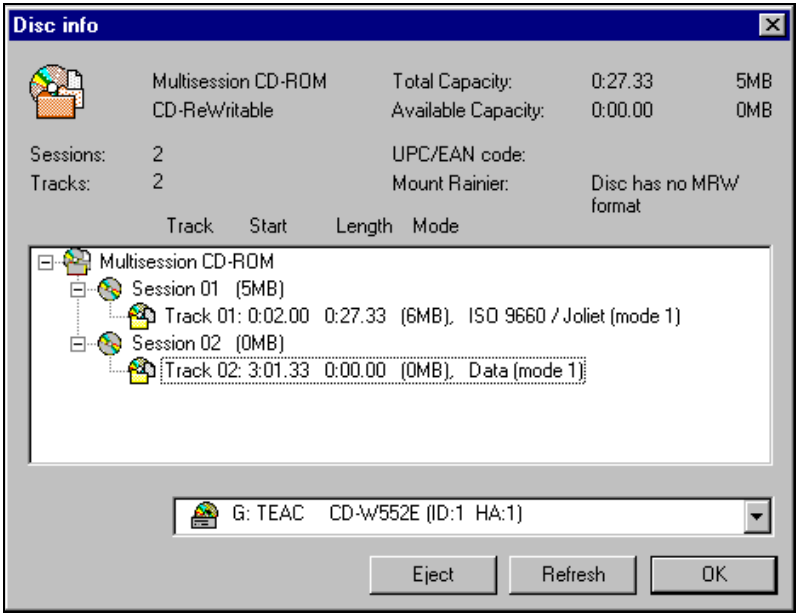


Рис. 6.19. Реакция Ahead Nero на некорректный номер последнего трека

Может ли такой диск быть скопирован вручную? Разумеется! Достаточно только один к одному переписать оригинальный TOC, не внося в него никаких изменений, и "залить" на диск содержимое первой сессии.

Разрыв в нумерации треков первой сессии

Эту ненормальную защиту мы рассмотрим лишь теоретически — на тот случай, если вам попадет диск, первая сессия которого разорвана каким-то моральным уродом и по причине конфликта с вашим оборудованием читается неправильно или, что более вероятно, не читается вообще. На этот случай в уголовном кодексе имеется статья об умышленном вредительстве, а с точки зрения прав потребителя — продажа товара заведомо не соответствующего соответствующим спецификациям — незаконна, и потому нейтрализация защит подобного типа на юридическом языке зовется не "взломом", а ремонтом товара, осуществляемого непосредственно самим покупателям. В общем, как говорится — на всякую хитрую гайку найдется свой хакер с характерным пивным животом.

Изучение "повадок" защиты мы начнем с создания своеобразной "лабораторной крысы" или тестового стенда — это уж как вам будет угодно. Извлечем из архива CCD-файл, сохранившийся после наших экспериментов по созданию фиктивного трека во второй сессии (вы ведь храните все свои CCD-файлы, правда?) и отредактируем его так, как показано в листинге 6.35. Номер второго трека заменен на девятый, соответственно, номер третьего трека второй сессии заменен на десятый для обеспечения корректной стыковки двух сессий.

Листинг 6.35. Создание разрыва в нумерации треков первой сессии

[Entry 1]	[Entry 1]	[Entry 3]	[Entry 3]	[Entry 8]	[Entry 8]
Session=1	Session=1	Session=1	Session=1	Session=2	Session=2
Point=0xa1	Point=0xa1	Point=0x02 →	**Point=0x09**	Point=0xa0	Point=0xa0
ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01
Control=0x04	Control=0x04	Control=0x04	Control=0x04	Control=0x4	Control=0x4
TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0
AMin=0	Amin=0	AMin=0	AMin=0	AMin=0	AMin=0
ASec=0	Asec=0	ASec=0	ASec=0	ASec=0	ASec=0
AFrame=0	AFrame=0	AFrame=0	AFrame=0	AFrame=0	AFrame=0
ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150
Zero=0	Zero=0	Zero=0	Zero=0	Zero=0	Zero=0
PMin=2 →	**Pmin=9**	PMin=3	PMin=3	PMin=3 →	**PMin=10**
PSec=0	Psec=0	PSec=1	PSec=1	PSec=0	PSec=0
PFrame=0	PFrame=0	PFrame=33	PFrame=33	PFrame=0	PFrame=0
PLBA=8850	**PLBA=-1**	PLBA=13458	PLBA=13458	PLBA=8850	**PLBA=-1**
[Entry 9]	[Entry 9]	[Entry 11]	[Entry 11]	[TRACK 1]	[TRACK 1]
Session=1	Session=1	Session=1	Session=1	MODE=1	MODE=1
Point=0xa1	Point=0xa1	Point=0x03 →	**Point=0x010**	INDEX 1=0	INDEX 1=0
ADR=0x01	ADR=0x01	ADR=0x01	ADR=0x01		
Control=0x04	Control=0x04	Control=0x04	Control=0x04	TRACK 2]	**[TRACK 9]**
TrackNo=0	TrackNo=0	TrackNo=0	TrackNo=0	MODE=1	MODE=1
AMin=0	Amin=0	AMin=0	AMin=0	INDEX 1=0	INDEX 1=0
ASec=0	Asec=0	ASec=0	ASec=0		
AFrame=0	AFrame=0	AFrame=0	AFrame=0	[TRACK 3] →	**[TRACK 10]**
ALBA=-150	ALBA=-150	ALBA=-150	ALBA=-150	MODE=1	MODE=1
Zero=0	Zero=0	Zero=0	Zero=0	INDEX 1=0	INDEX 1=0
PMin=3 →	**Pmin=10**	PMin=6	PMin=6		
PSec=0	Psec=0	PSec=1	PSec=1		
PFrame=0	PFrame=0	PFrame=33	PFrame=33		
PLBA=8850	**PLBA=-1**	PLBA=26958	PLBA=26958		

Проблемы начинаются уже с записи искаженного образа на диск. CloneCD не только не смог определить длину второго (ныне девятого) трека, чем

нас — бывалых экспериментаторов — действительно трудно удивить, но и неправильно интерпретировал выводную область первой сессии, приняв ее за самостоятельный трек с данными, имеющий номер 170 (или AAh в шестнадцатеричной системе исчисления). Самое интересное, что point, указывающий на Lead-out, в TOC явным образом не хранится, т. е. трека с номером 170 в оглавлении диска просто нет, и добыть его можно лишь путем чтения Q-канала подкода из самой области Lead-Out! Но если CloneCD явным образом обращается к Lead-Out, то почему принимает его за самостоятельный трек?

Впрочем, на качество "прожига" диска этот глюк никак не влияет, и потому мы можем смело пропустить его мимо ушей.

Листинг 6.36. Крайне неадекватная реакция CloneCD на разрыв треков

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 4726 Кбайт
Число треков: 9
Track 1: Данные Mode 1, размер: 4823 Кбайт
Track 9: Data, размер: 4294967200 Кбайт
Track 170: Data, размер: 4294962570 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:
Размер сессии: 3939 Кбайт
Число треков: 1
Track 10: Data, размер: 3939 Кбайт
```

Действительно, CloneCD крайне неадекватно среагировал на разрыв треков (листинг 6.30), "свихнувшись" до такой степени, что Lead-Out-трек (AAh или 170 в десятичной нотации) был воспринят им как трек с данными, причем его размер девятого и "стосемидесятого" треков "взлетел до небес". "Защищенный" таким образом диск, будучи вставленным в TEAC, вообще не распознается приводом. После долгих морганий индикатора активности, сопровождаемых надрывнем скрежетом головки, привод останавливает мотор и сообщает, что он не видит здесь никакого диска. Зато NEC обнаруживает целых девять аудиотреков, но при попытке проиграть хоть один из них возвращается сообщение об отсутствии диска. Лучше всех показал себя привод ASUS, пусть и потерявший вторую сессию, но по крайней мере нормально обрабатывающий содержимое первой из них.

Можно ли прочитать такой диск, не будучи при этом владельцем привода ASUS или аналогичного ему? Да, можно, — достаточно лишь подсунуть приводу диск с заведомо исправным TOC'ом и, дождавшись его опознавания, осуществить "горячую" смену дисков "на лету". Такой прием позволяет

скопировать само содержимое диска, но не его TOC. Для чтения TOC нам все-таки потребуется обратиться к услугам привода ASUS или любого другого, распознающего искаженный диск и не препятствующий работе с ним на секторном уровне. Если же у вас нет подобного привода, то не отчаивайтесь, все равно чтение TOC — это абсолютно бессмысленная операция. Границы треков и сессий мы можем восстановить и так (вы ведь не забыли о существовании Q-канала подкода?), а искаженные номера треков нам и даром не нужны, ибо взлом защиты предполагает восстановление упорядоченной нумерации, которая бы опознавалась всеми приводами без исключения. А потому за треком номер один должен идти трек с номером два, за треком номер два должен идти трек номер три, и не важно, что там написано в TOC! Не всему написанному можно верить. На заборах тоже многое пишут. Но ведь защитный механизм может элементарным образом привязываться к TOC, читая его содержимое и сравнивая с оригиналом! Если так, а чаще всего это именно так и есть, мы должны перехватить обращение к команде `READ TOC` (опкод 43h) и модифировать сравнивающий код так, чтобы он больше ничего не сравнивал. Подробнее о том, как именно это сделать, можно прочитать, в частности, в книгах "Техника и философия хакерских атак" и "Фундаментальные основы хакерства" Криса Касперски.

Разрыв в нумерации треков второй сессии

Хорошо, "ломать" нумерацию треков первой сессии —"саксь и маст дай", но вот реакция приводов на разрыв второй сесии уже не такая психованная, и, если предпринять все необходимые предосторожности, вероятность конфликта с оборудованием будет сведена к разумному минимуму, которым можно смело пренебречь (читай — ущерб от несанкционированного копирования "голых" дисков намного превышает убытки от возврата "нечитающихся" дисков, правда, среди пользователей встречаются и такие, что за каждый конфликтующий диск грозятся подать на вас в суд или просто оторвать вам голову).

Опираясь на опыт, приобретенный в процессе предыдущих экспериментов, мы без труда разорвем вторую сессию, изменив номер третьего трека на девятый (естественно, вы можете выбрать и другой номер, но только помните, что четные номера приносят несчастье).

При записи искаженного образа на диск CloneCD неправильно определяет номер последнего трека разорванной сессии, ошибочно принимая выводную область за самостоятельный трек (листинг 6.37). Действительно, CloneCD некорректно определил количество треков второй сессии, ошибочно посчитав, что их восемь (в то время как правильный ответ: два), также трек с номером 170, представляющий собой трек Lead-Out, ошибочно интерпретируется как трек с данными (совмещенные треки). Однако на качество "прожига" диска это никак не влиянет.

Листинг 6.37. CloneCD некорректно определяет количество треков второй сессии

```
ИНФОРМАЦИЯ О СЕССИИ 1:

Размер сессии: 4726 Кбайт
Число треков: 1
Track 1: Данные Mode 1, размер: 4726 Кбайт

ИНФОРМАЦИЯ О СЕССИИ 2:

Размер сессии: 3939 Кбайт
Число треков: 8
Track 2: Данные Mode 1, размер: 1722 Кбайт
Track 9: Данные Mode 1, размер: 2216 Кбайт
Track 170: Data, размер: 4294932446 Кбайт
```

"Полевые" испытания защищенного диска показывают следующие результаты. Приводы NEC и TEAC "видят" лишь первую сессию диска, а вторая оказывается недоступной даже на секторном уровне, однако команды SEEK, READ SUBCHANNAL и READ HEADER исполняются успешно. Если бы аналогичным образом вели себя все приводы, — разработчику защиты ничего бы не стоило разместить в Q-канале подкода второй сессии ключевую метку или просто проверить Q-канал "разорванной" сессии на читаемость. Такие копировщики, как Alcohol 120% и CloneCD, просто не увидят разорванной сессии, а если даже и увидят, то не смогут скопировать ее содержимое, возвращаемое, как уже говорилось, не в отдельном канале, а вместе с основным потоком данных. При условии, что разорванная сессия не доступна на секторном уровне (а это действительно так), на несанкционированных копиях диска ее просто не окажется, и команды SEEK, READ SUBCHANNAL и READ HEADER возвратят ошибку, позволяя тем самым отличить копию от оригинала.

Увы, некоторые приводы (ASUS в частности) не дают к разорванной сессии никакого доступа вообще, "благодаря" чему оригинальный диск ошибочно опознается защитой как копия. Так что закладываться на вторую сессию ни в коем случае нельзя! Впрочем, эти меры предосторожности практически не ослабляет защиту, поскольку даже целая сессия такого диска все равно не копируется.

Просмотр геометрии диска с помощью копировщика Ahead Nero показывает, что последний не только не может определить длины треков разорванной сессии (которые, по его мнению, не имеют никакой длины вообще), он катастрофически неправильно отображает их номера. Истинные номера

треков, записанные в TOC'е, на экран вообще не выводятся, замещаясь на последовательные порядковые номера (рис. 6.20). Таким образом, трек номер девять представляется как трек с номером три. В некотором смысле это может быть и верно, но вот для корректного копирования защищенного диска одних лишь порядковых номеров оказывается недостаточно.

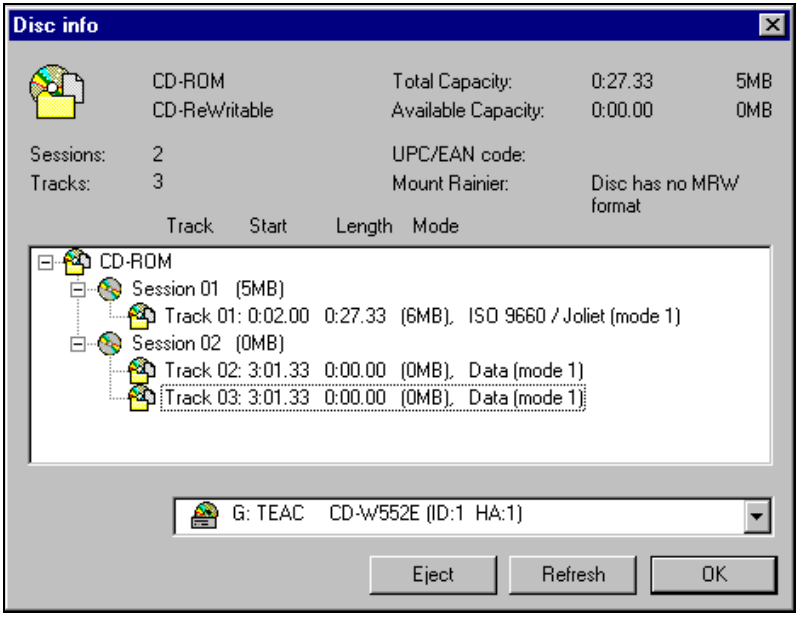


Рис. 6.20. Ahed Nero неправильно отображает номера треков

Копировщик CloneCD видит лишь первую сессию защищенного диска и, судя по всему, даже и не подозревает о существовании второй (листинг 6.38). Как следствие — разорванная сессия вообще не копируется и в TOC'е диска-копии отсутствует всякое упоминание о ней. Таким образом, для проверки лицензионной чистоты диска защитному механизму достаточно лишь считать TOC и сравнить его с эталонным TOC'ом оригинала.

Листинг 6.38. CloneCD неправильно отображает номера треков

```
ИНФОРМАЦИЯ О CD В ДИСКОВОДЕ:
Число сессий: 1
Занято на диске: 30911 Кбайт
Секторов: 13458
Время: 02:59:33 (мин:сек:кадр)
```

```
ИНФОРМАЦИЯ О СЕССИИ 1:
Размер сессии: 30911 Кбайт
Число треков: 3
Track 1: Данные Mode 1, размер: 30911 Кбайт
```

Ладно, с разорванной сессией все более или менее понятно. Ну не рассчитывали создатели CloneCD на такие извращения, ну не додумались до того, что нумерация треков может быть коварно измена. Но ведь и первая сессия защищенного диска также оказалась скопированной неверно (листинг 6.39). Отсюда мораль: несвоевременные санкции, предъявленные к разработчикам, придают "неприятный привкус" программному продукту.

Листинг 6.39. Содержимое TOC оригинального диска (слева) и диска, скопированного CloneCD (справа)

```
[Entry 2]        [Entry 2]        ; адрес выводной области первой сессии оказался
Session=1        Session=1        ; определен неправильно! Clone CD установил его
Point=0xa2       Point=0xa2       ; на адрес начала трека номер два (первого трека
ADR=0x01         ADR=0x01         ; второй сессии), что повлекло за собой искажение
Control=0x04     Control=0x04     ; длины первого трека и породило множество
TrackNo=0        TrackNo=0        ; нечитающихся секторов, расположенных в Lead-Out/
AMin=0           AMin=0           ; Lead-In-областях. Таким образом, для определения
ASec=0           ASec=0           ; подлинности диска вовсе не обязательно читать
AFrame=0         AFrame=0         ; содержимое TOC'a и достаточно всего лишь
ALBA=-150        ALBA=-150        ; определить полную емкость носителя, что можно
Zero=0           Zero=0           ; сделать и штатными средствами операционной системы
PMin=3 →         PMin=0           ; без обращения к интерфейсам ASPI32/SPTI
PSec=1 →         PSec=29          ; альтернативный путь - прочитать содержимое
PFrame=33        PFrame=33        ; TOC командой IOCTL_CDROM_READ_TOC, что также не
PLBA=13458 →     PLBA=2058        ; требует обращения к ASPI32/SPTI
[Entry 10]       [Entry 4]        ; трек номер два, принадлежащий второй сесии,
                                  ; CloneCD
Session=2        Session=1        ; запихнул в конец первой, даже не потрудившись
Point=0x02       Point=0x02       ; сопоставить стартовый адрес последнего со
                                  ; стартовым
ADR=0x01         ADR=0x01         ; адресом выводной области первой сессии. Очевидно,
Control=0x04     Control=0x04     ; что они "волшебным" образом совпадают, вероятно
TrackNo=0        TrackNo=0        ; потому-то выводная область и принимается за
                                  ; самостоятельный трек
AMin=0           AMin=0           ; попытка чтения содержимого второго трека ни к чему
ASec=0           ASec=0           ; хорошему не приводит, а третий
```

```
                                ; (ну то есть девятый)
AFrame=0        AFrame=0        ; трек вообще оказался потерян, поэтому копия диска,
ALBA=-150       ALBA=-150       ; полученная с помощью CloneCD отличается от
                                ; оригинала,
Zero=3          Zero=3          ; как небо от земли и защите не будет стоит большого
PMin=3          PMin=3          ; труда обнаружить факт несанкционированного
                                ; копирования
PSec=1          PSec=1          ; даже без обращений ко второй сессии! Вот такой он,
PFrame=33       PFrame=33       ; CloneCD! А еще претендует на звание дома высокой
PLBA=13458      PLBA=13458      ; культуры и быта, тьфу, на звание защищенного
                                ; копира!
```

Alcohol 120% с копированием такого диска справляется не в пример лучше, однако полученная с его помощью копия имеет по меньшей мере одно существенное отличие от оригинала. Номер разорванного трека непроизвольным образом меняется с девяти на три, то есть Alcohol 120% автоматически восстанавливает поврежденную нумерацию треков на правильную. Но ведь искаженная нумерация треков нам как раз и нужна! Защитный механизм, привязывающийся к TOC'у, просто забракует такой диск, обозвав его несанкционированной копий! (Забавно, но содержимое point'а `A1h`, указывающего на номер последнего трека диска, остается неизменным и по-прежнему равно девяти, то есть восстанавливая TOC, Alcohol 120% все равно восстановил его не корректно).

Таким образом, для привязки к оригинальному TOC'у, защитному механизму незачем считывать его содержимое в "сыром" виде, и можно вполне обойтись штатными средствами операционной системы, поскольку искажения, вносимые копировщиками в TOC, настолько велики, что в прямом смысле слова видны невооруженным глазом.

Тем не менее, стойкость защит данного типа достаточно невелика. Хакеру ничего не стоит вручную отредактировать образ диска, снятый Alcohol 120%, вернув разорванному треку его законный номер, равный девяти или иному другому числу (конкретное значение легко узнать с помощью утилиты CD_READ_TOC или аналогичной ей).

Диск, начинающийся не с первого трека

"Ну и запросы у вас...", —
"сказала" база данных и "повисла".

Требование стандарта начинать нумерацию треков с единицы вкупе с наличием point'а `A0h`, хранящего номер первого трека, выглядит несколько странным, если не сказать — избыточным. Мне могут возразить, что point `A0h` имеет смысл для второй и всех последующих сессиий, но я резонно отвечу:

во-первых, номер первого трека каждой сесссии равен point'у `A1h` предыдущей сессии плюс единица, во-вторых, номер первого трека всякой сессии — это наименьший номер трека из всех треков, принадлежащих данной сессии. Так что point'ы `A0h` и `A1h` все равно избыточны и предназначены исключительно для быстрого определения номера первого и последнего треков, без анализа всего TOC целиком, что для хлипких микропроцессоров первых аудиоплееров было действительно актуальным.

Но вот интересно — анализируют ли современные приводы эти point'ы или молчаливо закладываются на номер первого трека по умолчанию (чего стандарт, кстати, деать не запрещает)? Попробуем это выяснить, создав диск, начинающийся не с первого трека, а, например, сразу с трека номер два. Используя свой предыдущий опыт, мы это сделаем без труда. Достаточно лишь изменить point `0x01` на point `0x02`, соответствующим образом перенумеровав все point'ы последующих треков (если они есть), изменить `[TRACK 1]` на `[TRACK 2]` и перенумеровать все последующие треки (если они есть), наконец, увеличить point `A1h` первой сессии, а также point'ы `A0h` и `A1h` всех последующих сессий на единицу. Если этого не сделать, оставив point'ы прозябать по умолчанию, то такой диск не будет читаться приводом NEC вообще, TEAC'у он будет доступен лишь на секторном уровне и даже ASUS увидит только первую сессию, да и то лишь после долгих раздумий и сексапильных подергиваний головкой (так что ASUS — это "круто", правда в других отношениях он ведет себя весьма нервно). При попытке копирования такого диска посредством CloneCD, последний скажет, что диск пуст (хотя это и нет так), зато при очистке диска пишет "диск пуст: нет". Alcohol 120% на таком диске также ничего в упор не видит и, равно как и CloneCD, ничего и не копирует. Если вы уверены, что приводы ваших пользователей способны читать такие диски хотя бы на секторном уровне, вы легко можете создать практически непрошибаемую защиту, которую не скопирует практически ни один копировщик (мой — скопирует, поскольку он вообще не заглядывает в TOC), однако, ввиду потенциальной конфликтности защиты такого типа, на вашем месте я бы этого делать не стал, ну разве что в качестве курсовой работы, которая все равно никому не нужна, так почему бы тогда и не поизвращаться? (Помните историю о том парне, который написал в курсовой: "Тому, кто дочитает до этого места, ставлю ящик пива/водки/шампанского"?)

Диск, начинающийся не с первого трека, но с корректно установленными point'ами `A0h` и `A1h` практически всеми, доступными мне приводами, читается нормально. И все бы хорошо, да вот досада — NEC, под тихие щелкающие звуки, "завешивается" таким диском вплоть до нажатия на Eject. Так что это не очень хорошая защита, тем более, что тот же CloneCD "обламывает" ее по полной программе, создавая вполне корректную и работоспособную копию. Иначе ведет себя Alcohol 120%, выбрасывающий "access violation" (нарушение доступа) и аварийно завершающий свое выполнение.

Таким образом, приводы ASUS и TEAC активно используют point'ы `A0h` и `A1h`, а NEC, судя по всему, рассчитывает на то, что нумерация треков всякого диска обязательно должна начинаться с единицы, что с одной стороны не противоречит Стандарту, а с другой стороны и не соответствует ему, т. к. point `A0h` присутствует в нем не даром и его игнорирование ни к чему хорошему не приводит.

Ситуация с Alcohol 120% до конца не ясна. Ошибка доступа — очевидное следствие грубых алгоритмических ошибок и просчетов проектирования. Скорей всего, Alcohol 120% читает TOC в буфер и последовательно сканирует его на предмет поиска трека номер один, забывая при этом контролировать "вылет" за доступные буферу границы.

А вот CloneCD понимает стандарт правильно, благодаря чему копирует защищенные диски на ура (впрочем, не в обиду будет сказано, такое поведение CloneCD скорее исключение, чем правило, и, как мы уже видели и как увидим далее, редкий диск с искаженным TOC'ом удается скопировать без ошибок).

Диск с нулевым треком

Задумывались ли вы, почему нумерация треков лазерных дисков начинается с единицы, а не с нуля? Ведь говорят, чтобы отличить программиста от простого смертного, достаточно дать ему команду "рассчитайсь!". Нормальный человек скажет "первый" (если он действительно первый) и будет по-своему прав. Программист же сначала уточнит, в какой системе исчисления вести расчет (в двоичной, восьмеричной, шестнадцатеричной), и затем, сделав шаг вперед, гордо скажет: "Нулевой". "Так ведь лазерные диски изначально разрабатывались для пользователей! — ответите вы. — А пользователи более привычны к натуральной, а не позиционной системе исчисления, и потому первый трек должен быть именно первым, но никак не нулевым".

И все же, несмотря на всю убедительность своих доводов, вы будете не правы. Отсчет треков всякого диска начинается не с единицы, а с нуля. Да, нулевой номер зарезервирован за служебным треком (вводной областью диска), и его содержимое недоступно на интерфейсном уровне, но это ничего не меняет! Поле TNO (Track Number) Q-подканала диска области Lead-In равно нулю, следовательно, *с точки зрения привода всякий диск начинается с трека номер ноль.* Электронная начинка привода читает и адресует нулевой трек точно так же, как и любой другой трек диска, сохраняя тем самым прозрачность и упорядоченность принятой системы нумерации. С точки зрения системных программистов, разрабатывающих микропрограммные прошивки, отсчет треков всегда начинается с нуля. С точки же зрения пользователей привода — с единицы. Одним словом, и "волки сыты и овцы целы"!

Атрибуты нулевого трека отсутствуют в TOC, поскольку этот трек как раз и служит для хранения TOC. Давайте задумаемся, что произойдет, если одному

из point'ов подлинного или фиктивного трека мы присвоим значение ноль, то есть, попросту говоря, создадим еще один нулевой трек в пользовательской области диска?

Если помимо внесения подложных данных в TOC мы еще и скорректируем содержимое Q-канала подкода, забив поле TNO нулями, то с точки зрения привода такой трек будет неотличим от вводной области диска и попытка его посекторного чтения будет обречена на провал (хотя некоторые приводы и не такое читают). Субканальные данные нулевого трека теоретически должны быть доступны для чтения командами `SEEK` или `READ SUBCHANNELL`, но никаких гарантий на счет этого у нас нет, поскольку наличие двух подряд идущих областей Lead-In сильно нервирует привод, и его реакция становится совершенно непредсказуемой. Отказ от восстановления субканальных данных мало что меняет. Одно лишь наличие нулевого point'а в TOC'е — событие вполне неординарное и взаимно противоречивое.

Большинство приводов просто свихнутся и откажутся обрабатывать такой диск, совершенно непредсказуемым образом осуществляя чтение его оглавления. В частности, NEC при выполнении команды READ TOC возвращает ошибку, ASUS воспринимает нулевой трек как индикатор завершения TOC, а TEAC, столкнувшись с нулевым треком, начинает очень сильно нервничать и вместо атрибутов всех последующих треков выдает содержимое своих внутренних буферов вместе с мусором, оставшимся от TOC'а предыдущего диска. Короче говоря, *нулевой трек делает лазерный диск практически полностью нечитаемым*.

На этом, собственно, можно было бы и остановиться (кому нужна жутко конфликтная защита, работающая исключительно в лабораторных условиях и крайне нежизнеспособная на практике?!), если бы не одно "НО". Стремительное падение цены на оптические носители позволяет использовать лазерный диск не только для хранения полезной информации, но и в качестве своеобразного ключа.

Весь фокус в том, что наличие нулевого трека на диске никак не препятствует чтению субканальных данных спиральной дорожки, но подавляющее большинство копировщиков (включая Alcohol 120% и CloneCD) скорее зависнут, чем скопируют такой диск! Таким образом, алгоритм работы защитного механизма сводится к "ручному" чтению TOC'а командами `SEEK` или `READ SUBCHANNEL` с последующей проверкой его содержимого на предмет наличия нулевого трека.

И хотя ключевой диск не может содержать никаких других данных кроме собственно самого проверяемого TOC'а — это ли беда? В каком-то смысле это даже достоинство. Пусть на одном лазерном диске, никак не защищенном от копирования, содержится демонстрационная версия программы, свободно доступная и для скачивания через Интернет. Чтобы она превратилась в полноценную полнофункциональную версию, пользователь должен

вставить в привод ключевой диск, полученный от регионального дилера или переданный непосредственно самим разработчиком по почте (собственно, не обязательно постоянно держать ключевой диск в приводе, защита может запомнить флаг регистрации и в реестре, запрашивая ключевой диск лишь изредка — на тот случай, если пользователь захочет одолжить его кому-нибудь). Согласитесь, это гораздо надежнее ключевого файла или регистрационного номера, которым ничего не стоит поделиться с другом или выложить в Интернет, а учитывая, что субканальные данные диска могут хранить не только ключ, но и исполняемый (интерпретируемый) код, обеспечивающий полнофункциональность зарегистрированной программы, становится ясно, что если у хакера нет ни одной полностью работоспособной копии защищенного приложения, взломать его за разумное время будет просто нереально. Но довольно слов, перейдем к делу, попытавшись перво-наперво создать образ защищенного диска с нулевым треком внутри (листинг 6.40). Как мы сейчас и увидим, оказывается это не так-то просто сделать!

Если просто обнулить point первого трека, то CloneCD наотрез откажется открывать такой образ, ссылаясь на ошибку анализа. На самом деле, этих ошибок по меньшей мере две. Первая — полная неосведомленность CloneCD о потенциальной возможности существования нулевых треков, коих он в упор не видит. Вторая — непростительный оптимизм, закладывающийся на тот факт, что каждая сессия должна содержать в себе по меньшей мере один трек (что в действительности вовсе не факт, а лишь допущение).

Alcohol 120% воспринимает сессию с единственным нулевым треком внутри более благодушно, корректно отображая его номер (листинг 6.40), однако при попытке записи такого образа на диск появляется сообщение "access violation" и копировщик "наглухо повисает", даже не удосужившись аварийно завершить свою работу (рис. 6.21). Вызов "Диспетчера программ" с последующим "умерщвлением разбушевавшегося" процесса не решает проблемы, т. к. лоток диска остается заблокированным и приходится прибегать к помощи утилиты CD.lock.exe для уменьшения счетчика блокировок на единицу.

Листинг 6.40. Alcohol 120% открывает образ защищенного диска вполне успешно, честно отображая нулевой номер первого трека, правда некорректно определяет его длину

```
Тип:        Файл-образ CloneCD
Путь:       L:\CD-hack\
Имя:        IMAGE.CCD
            IMAGE.img
            IMAGE.sub
Размер:     8.81 MB
Сессий:     2
```

```
Треков:    2


Сессия 01:
  Трек 00: Mode 1, Длина: 000000(0 Byte), Адрес: 000000
Сессия 02:
  Трек 01: Mode 1, Длина: 000000(0 Byte), Адрес: 013458
```

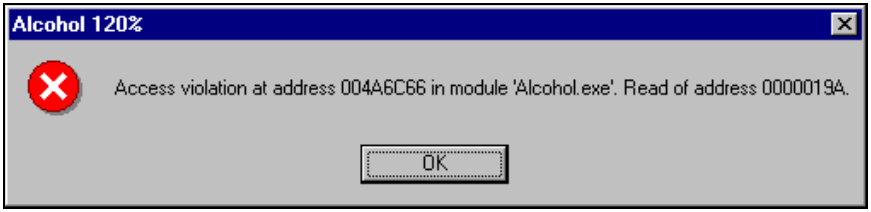


Рис. 6.21. Реакция Alcohol 120% на попытку записи образа диска с единственным нулевым треком внутри первой сессии

Постойте, но ведь это же настоящая золотая жила! Диск с единственным нулевым треком внутри первой сессии не то что не копируется, он даже и не "прожигается"! Даже если хакер каким-то неизвестным науке способом снимет с защищенного диска правильный дамп, ему будет нечем этот дамп записать!!! Правда, и разработчику защищаемого приложения ключевой диск нечем записать тоже, ну, разве что отважиться на разработку собственной программы "прожига". Естественно, изготовить штампованные CD с нулевым треков вообще не проблема, но этот путь доступен далеко не всем (индивидуальным программистам он недоступен точно).

Компромиссным вариантом защиты становится добавление в искажаемую сессию хотя бы единственного ненулевого трека. Такой диск может быть изготовлен с помощью того же CloneCD, и корпеть над написанием собственной "прожигалки" в этом случае не надо, что есть плюс. Однако коль скоро для создания оригинального диска используется утилита массового распространения, процесс создания несанкционированных дубликатов значительно упрощается. Хакеру достаточно снять с диска корректный дамп, а все остальное — это забота копировщика CloneCD. Необходимость разрабатывать специализированный софт для взлома при этом отпадает. Одним словом, все, что легко защищается, легко и ломается. Впрочем, квалифицированных хакеров не так уж и много, и для предотвращения "утекания" своей продукции нам с вами достаточно добиться некопируемости ключевого диска распространенными копировщиками в автоматическом режиме. И, как мы увидим в дальнейшем, защита данного типа полностью удовлетворяет этому требованию.

Процесс подготовки защищенного диска не лишен определенных тонкостей. Создание фиктивного трека с нулевым номером не вызывает особых трудностей, но вот вопрос: где его разместить? В первой, а, может быть, лучше во второй сессии? До подлинного трека или после? Поскольку вводная область первой сессии недоступна для чтения на субканальном уровне, то прочитать TOC первой сессии вручную нельзя. Нельзя и закладываться на команду READ TOC, поскольку, как уже было сказано ранее, ее корректное выполнение не гарантируется. Вводные области второй и всех последующих сессий свободно доступны на субканальном уровне, и ручное чтение хранимого ими TOC'а все-таки возможно. Конкретная позиция нулевого трека внутри сессии особой роли не играет, и нулевой трек может быть с одинаковым успехом размещен как до ненулевого трека, так и после него.

Только, пожалуйста, не забывайте о необходимости коррекции point'а `A0h`, хранящего номер "первого" трека всякой сессии. Если его значение оставить без изменений, то образ диска запишется без каких-либо препирательств со стороны CloneCD, но никаких упоминаний о нулевом треке в TOC'е прожженного диска не окажется! Точно так же ведет себя и Alcohol 120%. Чтобы этого избежать, значение point'а `A0h` той сессии, к которой вы добавляете нулевой трек, должно быть сброшено в ноль.

Фрагмент отредактированного CCD-файла приведен в листинге 6.41.

Листинг 6.41. Фрагмент CCD-файла с добавленным нулевым треком

```
TocEntries=13    TocEntries=14    ; корректируем количество входов в TOC
[Entry 8]        [Entry 8]        ; это вход не обязательно должен быть восьмым…
Session=2        Session=2        ; …главное, чтобы Session == 2, а Point == A0h
Point=0xa0       Point=0xa0       ; этот Point отвечает на номер первого трека
ADR=0x01         ADR=0x01         ; это служебные поля ADR/Control, описывающие
Control=0x04     Control=0x04     ; режим обработки трека (это трек с данными)
TrackNo=0        TrackNo=0        ; TNO = 0 - это Lead-In область
AMin=0           AMin=0           ; \
ASec=0           ASec=0           ;  +- условный текущий абсолютный адрес
AFrame=0         AFrame=0         ; /
ALBA=-150        ALBA=-150        ; условный текущий LBA-адрес
Zero=0           Zero=0           ; это поле всегда равно нулю
PMin=2 →         PMin=0           ; корректируем номер "первого" трека
PSec=0           PSec=0           ; эти поля не имеют никакого смысла и должны
PFrame=0         PFrame=0         ; быть равны нулю
PLBA=8850        PLBA=8850        ; LBA-"адрес" номера "первого" трека
                 [Entry 11]       ; добавляем еще одно Entry, описывающее нулевой трек
                 Session=2        ; нулевой трек должен быть не в первой сессии
                 Point=0x00       ; номер трека - ноль
```

```
ADR=0x01        ; Sub-channel Q encodes current position data
Control=0x04    ; трек с данными
TrackNo=0       ; это Lead-In
AMin=0          ; \
ASec=0          ;  + - условный абсолютный адрес Lead-In
AFrame=0        ; /
ALBA=-150       ; условный LBA-адрес Lead-In
Zero=0          ; это поле должно быть равно нулю
PMin=3          ; \
PSec=1          ;  + - абсолютный стартовый адрес нулевого трека
PFrame=66       ; /
PLBA=13458      ; LBA-адрес нулевого трека
```

При просмотре геометрии защищенного таким образом диска Ahead Nero выдает приблизительно следующую информацию (рис. 6.22)[1]. То, что он посчитал вторую сессию открытой ("Session is open"), вполне объяснимо, так как созданный нами нулевой трек был ошибочно принят Ahead Nero за вводную область, в результате чего шаткое равновесие между вводными и выводными областями оказалось нарушенным. Между тем, вторая сессия диска все-таки закрыта, и анализ TOC подтверждает это. А не знать, что упоминание о вводных областях никогда в явном виде не присутствует в TOC'е, — глупо. Разобраться, почему нулевой трек оказался приобщен Ahead Nero к первой сессии, несколько сложнее. По-видимому, это грубая алгоритмическая ошибка, которая не делает чести ни самому Ahead Nero, ни его разработчикам.

CloneCD ведет себя аналогичным образом и при попытке копирования защищенного диска на приводах ASUS и TEAC со всего маху врезается в выводную область первой сессии диска. Вторая сессия (с нулевым треком) по причине грубых алгоритмических ошибок полностью выпадает из его поля зрения, и в TOC'е скопированного диска о ней даже и не упоминается. Стартовый адрес выводной области первой сессии также определяется неправильно (CloneCD устанавливает его на стартовый адрес нулевого трека второй сессии). Point'ы `B0h` (стартовый адрес следующей позиции для дозаписи) и `C0h` (стартовый адрес первой вводной области диска) тоже оказываются потерянными. Короче говоря, TOC скопированного диска более чем существенно отличается от TOC'а оригинального, и обнаружить факт несанкционированного копирования не составит никакого труда, да вы их сами сравните (листинг 6.42). TOC оригинального ключевого диска с нулевым треком внутри второй сессии (слева), причем атрибуты нулевого трека

[1] "Приблизительно" — потому что на некоторых приводах Ahead Nero вообще ничего не выдает.

выделены серой заливкой и TOC его копии, полученной с помощью копировщика CloneCD (справа). Все несовпадения выделены полужирным шрифтом.

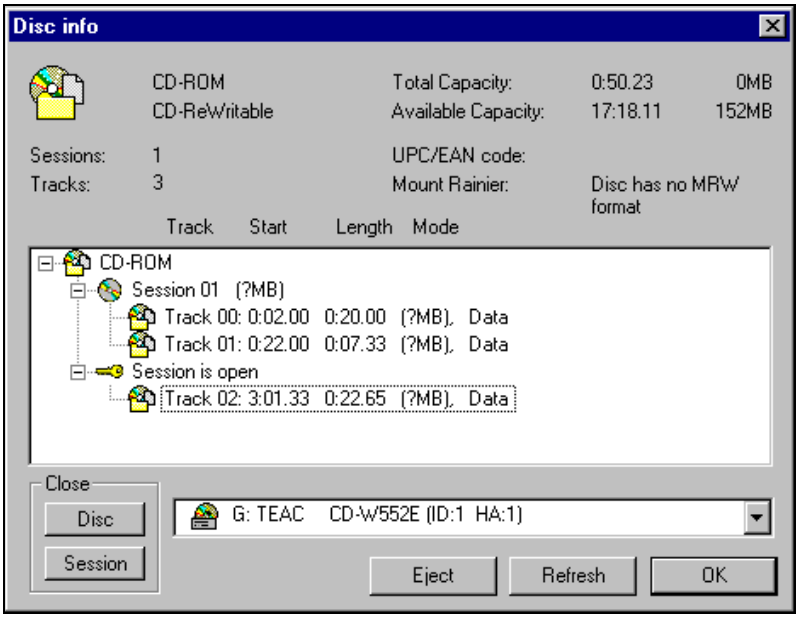


Рис. 6.22. Наличие нулевого трека на диске путает Ahead Nero, вводя его в глубокое заблуждение относительно состояния последней сессии диска. Ahead Nero считает, что вторая сессия открыта, хотя на самом деле это не так

Листинг 6.42. TOC оригинального ключевого диска с нулевым треком внутри второй сессии (слева) и TOC его копии, полученной с помощью CloneCD (справа)

```
01 14 00 A0 00 00 00 00 01 00 00
01 14 00 A1 00 00 00 00 01 00 00
01 14 00 A2 00 00 00 00 00 1D 21
01 14 00 01 00 00 00 00 00 02 00
01 54 00 B0 02 3B 21 03 16 0E 22
01 54 00 C0 A2 C8 E0 00 61 1B 15
02 14 00 A0 00 00 00 00 02 00 00
02 14 00 A1 00 00 00 00 00 00 00
02 14 00 A2 00 00 00 00 03 18 17
02 14 00 00 00 00 00 00 03 01 42
```

```
01 14 00 A0 00 00 00 00 01 00 00
01 14 00 A1 00 00 00 00 01 00 00
01 14 00 A2 00 00 00 00 03 01 42
01 14 00 01 00 00 00 00 00 02 00
```

При попытке копирования защищенного диска на приводе NEC (который, как уже отмечалось, отказывается читать TOC с нулевым треком), копировщик CloneCD удивленно спрашивает: "Диск пуст?" — и вне зависимости от нашего ответа, даже и не пытается приступить к его копированию, вызывая страшный хакерский гнев и раздражение.

Alcohol 120% при попытке копирования защищенного диска просто "виснет", едва лишь успев перед смертью выбросить исключение "access violation", но зато заблокировав лоток привода, так что без уменьшения счетчика блокировок извлечь диск помогает разве что тотальная перезагрузка системы.

В общем, дело — мрак! (С точки зрения "пиратов" конечно.) И мы по праву можем считать, что процедура создания некопируемого ключевого диска завершилась успешно. Теперь недурно бы разобраться: как с полученным диском вообще работать и каким именно образом защитный механизм сможет отличить копию от оригинала.

Первое, что приходит нам в голову, — прочитать "сырой" TOC командой `READ TOC` и проверить наличие трека номер ноль, а для пущей надежности и его атрибуты. Если нулевой трек действительно присутствует в TOC'е и его атрибуты (т. е. поля `Session`, `ADR`, `Control`, `PMin:PSec:Pfarme`) полностью соответствуют эталону, — это оригинальный диск и, соответственно, наоборот. Достоинство такого алгоритма в том, что он очень просто реализуется, укладываясь буквально в десяток строк кода, а недостаток — нестабильность опознавания ключевого диска на определенных моделях приводов. Привод может отказаться читать TOC, и к такому повороту событий защита должна быть заранее готова. Давайте сделаем так: если команда `READ TOC` возвращает ошибку, но диск присутствует в приводе и не препятствует выполнению команды `SEEK`, то это оригинальный диск. Конечно, подобное эвристическое допущение значительно ослабляет защиту, однако для большинства применений ее стойкости будет вполне достаточно.

Однако безошибочное выполнение команды `READ CD` еще не есть свидетельство того, что она выполнена успешно. Ни один известный мне привод, способный читать TOC с нулевым треком, не читает его правильно, а потому защита должна заранее учитывать характер возможных искажений TOC'а и адекватно ему противостоять.

Рассмотрим, например, какой результат возвращает привод TEAC (листинг 6.43). Нулевой трек выделен серой заливкой, а "мусор", следующий за ним, — полужирным шрифтом.

Листинг 6.43. Содержимое TOC ключевого диска, возвращенное приводом TEAC

```
Номер сессии
|  ADR/Control
|  |  TNO
```

```
|  |  |  Point
|  |  |  |  |  AM:AS:AF PM:PS:PF
|  |  |  |  |  |  |  |  |  |  |
01 14 00 A0 00 00 00 00 01 00 00 ; point A0 - номер первого трека сессии 1 в PM
01 14 00 A1 00 00 00 00 01 00 00 ; point A1 - номер последнего трека сессии 1 в PM
01 14 00 A2 00 00 00 00 00 1D 21 ; point A2 - адрес Lead-out сессии 1 в PM:PS:PF
01 14 00 01 00 00 00 00 00 02 00 ; point 01 - стартовый адрес трека 1 в PM:PS:PF
01 54 00 B0 02 3B 21 03 16 0E 22 ; point B0 - позиция дозаписи в AM:AS:AF
01 54 00 C0 A2 C8 E0 00 61 1B 15 ; point C0 - стартовый адрес Lead-In в
                                 ; PM:PS:PF /искж
02 14 00 A0 00 00 00 00 02 00 00 ; point A0 - номер первого трека сессии 1 в PM
02 14 00 A1 00 00 00 00 00 00 00 ; point A1 - номер последнего трека сессии 2 в PM
02 14 00 A2 00 00 00 00 03 18 17 ; point A2 - адрес Lead-out сессии 2 в PM:PS:PF
02 14 00 00 00 00 00 00 03 01 42 ; point 00 - стартовый адрес трека 0 в PM:PS:PF
FB FD 00 FB F4 FB 7A FF FD FD FF ; \          | как видно, встретив нулевой трек,
FB DF 00 FA FD F5 FF BF FB FE FF ;  + - мусор | TEAC вместо осмысленных данных
FE F7 00 FB FF FD FB FF FF F7 FF ; /          | начал изрыгать мусор
```

Что это за подозрительный мусор, расположенный следом за нулевым треком? Это — содержимое внутренних буферов привода, попавшее сюда в результате грубой программистской ошибки в микропрограмме привода (между прочим, тестировалась самая свежая на момент написания этих строк прошивка — 1.09). Небольшое расследование убедительно доказывает, что "мусор" носит неслучайный характер и представляет собой "хвост" TOC'а предыдущего диска.

Давайте, вставив в привод какой-нибудь диск (например, *"Soul Ballet Hit Collection"*), заново сменим его на ключевой и посмотрим, что у нас из этого получится (листинг 6.44).

Листинг 6.44. TOC диска Soul Ballet (слева) и TOC ключевого диска (справа), возвращенные приводом TEAC

```
01 10 00 A0 00 00 00 00 01 00 00          01 14 00 A0 00 00 00 00 01 00 00
01 10 00 A1 00 00 00 00 10 00 00          01 14 00 A1 00 00 00 00 01 00 00
01 10 00 A2 00 00 00 00 48 1C 05          01 14 00 A2 00 00 00 00 00 1D 21
01 10 00 01 00 00 00 00 00 02 00          01 14 00 01 00 00 00 00 00 03 00
01 10 00 02 00 00 00 00 03 35 40          01 54 00 B0 02 3B 21 03 16 0E 22
01 10 00 03 00 00 00 00 08 14 33          01 54 00 C0 A2 C8 E0 00 61 1B 15
01 10 00 04 00 00 00 00 0C 21 0D          02 14 00 A0 00 00 00 00 02 00 00
01 10 00 05 00 00 00 00 10 3A 2D          02 14 00 A1 00 00 00 00 00 00 00
```

```
01 10 00 06 00 00 00 00 16 23 19
01 10 00 07 00 00 00 00 1C 1B 0C
01 10 00 08 00 00 00 00 21 07 49
01 10 00 09 00 00 00 00 25 1F 19
01 10 00 0A 00 00 00 00 2A 01 06
01 10 00 0B 00 00 00 00 2D 2D 00
01 10 00 0C 00 00 00 00 33 29 02
01 10 00 0D 00 00 00 00 39 08 45
01 10 00 0E 00 00 00 00 3F 1E 27
01 10 00 0F 00 00 00 00 43 1E 29
01 10 00 10 00 00 00 00 44 03 15
```

```
02 14 00 A2 00 00 00 00 03 18 17
02 14 00 00 00 00 00 00 03 01 42
09 25 00 1F 00 00 00 00 19 01 10
0A 2A 00 01 00 00 00 00 06 01 10
0B 2D 00 2D 00 00 00 00 00 01 10
0C 33 00 29 00 00 00 00 02 01 10
0D 39 00 08 00 00 00 00 45 01 10
0E 3F 00 1E 00 00 00 00 27 01 10
0F 43 00 1E 00 00 00 00 29 01 10
10 44 00 03 00 00 00 00 15 FF FF
```

Смотрите! Сейчас содержимое TOC'а ключевого диска существенно изменилось. Пускай не все содержимое, но вот хвост изменился точно. Причем последовательность байт "хвоста" ключевого диска соответствует последовательности байт диска *"Soul Ballet"*. Пускай "…09 00 00 00 00 25 1F 19…" и "…09 25 00 1F 00 00 19…" и не совсем тождественные последовательности, но если убрать паразитные нули, мы получим: "…09 25 1F 19…" и "…09 25 1F 19…", которые побайтно равны друг другу. Так что мы действительно имеем дело с ошибкой в прошивке, что не делает чести ни самому приводу, ни его разработчикам.

Привод ASUS ведет себя более корректно (листинг 6.45), просто "обрубая" TOC по нулевому треку, даже в том случае, когда нулевой трек — не последний трек диска, что тоже расценивается как микропрограммная ошибка, пускай и не такая грубая.

Листинг 6.45. TOC ключевого диска, возвращенный приводом ASUS

```
01 14 00 A0 00 00 00 00 01 00 00
01 14 00 A1 00 00 00 00 01 00 00
01 14 00 A2 00 00 00 00 00 1D 21
01 14 00 01 00 00 00 00 00 03 00
01 54 00 B0 02 3B 21 03 16 0E 22
01 54 00 C0 A2 C8 E0 00 61 1B 15
02 14 00 A0 00 00 00 00 02 00 00
02 14 00 A1 00 00 00 00 00 00 00
02 14 00 A2 00 00 00 00 03 18 17
02 14 00 00 00 00 00 00 03 01 42
```

Привод NEC, как уже говорилось ранее, вообще не выдает ничего, кроме ошибки, которую защитный механизм вынужден интерпретировать как

индикатор лицензионности диска, в противном случае разработчик может поплатиться своей головой, которую с удовольствием оторвут легальные пользователи, пытающиеся "скормить" защищенный диск "неправильному" (с точки зрения защиты) приводу.

Тем не менее, преднамеренное ослабление стойкости защиты — это не выход. Уж лучше попробовать прочитать TOC вручную. Это достаточно трудно реализуется на программном уровне, но еще труднее ломается! Если команду `READ TOC` легко и проэмулировать, то воссоздать особенности обработки субканальных данных практически нереально, благодаря чему усиленный вариант защиты с легкостью обойдет все копировщики, эмулирующие виртуальные диски.

Опасаясь охладить ваш воинственный программистский пыл, я все же скажу: правильно считать субканальные данные не так-то просто, как это может показаться на первый взгляд, и одних лишь спецификаций для успешной реализации защитного механизма мало. В них не отображено и доли тех "чудачеств" приводов, с которыми приходится сталкиваться на практике.

- *Первое и самое главное*: абсолютные адреса секторов никоим образом не связаны с "соответствующими" им субканальными данными, хотя бы уже потому, что одна секция субканальных данных "размазана" по нескольким секторам, причем в силу определенных конструктивных особенностей, обработка субканальных данных и данных основного потока осуществляется раздельно, благодаря чему при позиционировании головки на сектор N с последующим вызовом команды `READ SUBCHANNEL`, мы получим субканальные данные не сектора N, а сектора M, лежащего "поблизости" от сектора N. Понятие "поблизости" всяк производитель определяет самостоятельно, зачастую уползая не на одну сотню секторов вперед.
- *Второе*: связка команд `SEEK` и `READ SUBCHANNEL` *неустойчива* и обладает плохой воспроизводимостью результатов. Никто не гарантирует, что позиционирование на сектор N+k приведет к чтению субканальных данных сектора M+k. Привод может возвратить как данные сектора M, так и данные сектора M+i. Также никто не гарантирует, что повторное позиционирование на сектор N приведет к чтению субканальных данных сектора M *(кстати, не забывайте между двумя соседними вызовами команды SEEK выдерживать паузу хотя бы в 1 сек, иначе головка просто не успеет переместиться на новое место и привод как ни в чем не бывало возвратит уже скэшированные субканальные данные с места предыдущего позиционирования)*. Остается опираться лишь на текущие адреса секторов, возвращаемые в самой субканальной информации (поле *"Absolute CD Address"*). Встретив в этом поле адрес "своего" сектора, мы можем быть абсолютно уверенными в том, что эта субканальная информация принадлежит именно "нашему" сектору, а не какому-то сектору еще (листинг 6.46).

Листинг 6.46. Правильная интерпретация субканальной информации

```
LBA - 10D4:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6D
^^^^^^^^^^                      ^  ^^^^^^^^^^^ ^^^^^^^^^^^
    |                           |       |            |
LBA-адрес  сектора, на  который | атрибуты point'a LBA-адрес сектора,
осуществлялось позиционирование |                  чьи субканальные данные
командой SEEK                 point               возвратил привод
```

- *Третье*: конкретный формат субканальной информации определяется не стандартом, а самим приводом и значительно варьируется от одной модели к другой. Наиболее непостоянны в этом смысле вводные и выводные области диска. Стандарт вообще ничего не говорит о возможности их чтения на субканальном уровне, молчаливо полагая, что это никому и не нужно. Вот производители и извращаются в меру своей распущенности и фантазии. Поля абсолютных и относительных адресов могут безо всяких предупреждений меняться местами, а сами адреса могут задаваться как в формате M:S:F, так и в LBA-форме. Значения point'ов `A0h`, `A1h` и `A2h` (номер первого трека, номер последнего трека и адрес области Lead-Out) могут замещаться значениями `64h`, `65h` и `66h` соответственно. Наконец, нестандартные point'ы (в том числе и нулевые point'ы) в субканальных данных зачастую попросту отсутствуют — вместо этого возвращаются данные либо предыдущей, либо последующей секций!

Все это значительно усложняет интерпретацию субканальных данных и поиск в ней нулевых треков, поэтому приходится действовать так: последовательно читая субканальные данные различных секторов диска, дожидаемся того момента, когда номера треков сменятся сначала на `AAh`, а затем и на `00h`, что будет соответствовать переходу головки с области Lead-Out первой сессии на область Lead-In второй сессий. Продолжая читать Lead-In, мы попытаемся определить, в какой закономерности изменяются поля абсолютных и относительных адресов и форму их представления (LBA или M:S:F). Собственно, формат представления определить очень легко. Если младший байт адреса принимает значения больше, чем 75 (`4Bh`), то это, несомненно, LBA и наоборот. Далее — поскольку поля относительных адресов в вводной области диска используются для хранения атрибутов "своего" point'a, то они чрезвычайно сильно отличаются от текущих адресов секторов — тех, на которых и осуществлялось позиционирование. Напротив, поля абсолютных адресов к текущим адресам должны быть достаточно близки.

Остается решить последнюю проблему — что делать, если в субканальных данных Lead-In области нулевого трека попросту не окажется? Не спешите делать вывод о нелицензионности диска — ведь, как уже было сказано ранее, некоторые приводы нестандартные point'ы просто не возвращают. При

этом *абсолютные адреса секторов, хранящих субканальные атрибуты нулевого трека, в считанном TOC'е не будут присутствовать!* Копия диска, полученная любым из существующих на данный момент копировщиков, по данным абсолютным адресам будет содержать атрибуты совершенно других треков, которые привод вполне корректно прочитает и возвратит либо же вовсе откажется позиционировать головку на эту область, выдавая ту или иную ошибку.

Ну, что, парни, слабо реализовать такое? Для облегчения восприятия материала далее будут приведены субкальные данные ключевого диска, возвращенные различными приводами. А подробные комментарии, щедро разбросанные автором, помогут разобраться что к чему (см. листинг 6.47).

Листинг 6.47. Результат чтения субканальной информации из области Lead-In на приводе TEAC; отчетливо просматривается нулевой трек

```
            +internal+  Format
            |  |  |  |  |  ADR/Control
            |  |  |  |  |  |  TNO
            |  |  |  |  |  |  |  Point
            |  |  |  |  |  |  |  |  +- PLBA -+  +- ALBA -+
            |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
LBA - 10D4:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6D ; LBA 10D4 → 116D
LBA - 10D5:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6D ; LBA 10D5 → 116D
LBA - 10D6:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6E ; LBA 10D5 → 116E
LBA - 10D7:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6D ; LBA 10D7 → 116D (!)
LBA - 10D8:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6E ; ("биение" головки)
LBA - 10D9:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6E ; LBA 2292 = 02:00:00
                                                           ; M:S:F
LBA - 10DA:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; LBA FFh - 6Ah = 95h
                                                           ; (149)
LBA - 10DB:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; LBA 149 == MSF 0:0:1
LBA - 10DC:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 74 ; что удивительно, т.к.
LBA - 10DD:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 75 ; последнего трека
                                                           ; должен
LBA - 10DE:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 74 ; быть в PM, но не в PF
LBA - 10DF:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 74 ; учитывайте это!
LBA - 10E0:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; продолжается биение
LBA - 10E1:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; головки: сектора идут
LBA - 10E2:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7B ; не упорядоченно:
                                                           ; 1179, 1179,
LBA - 10E3:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; 117B, 1179, 117A, 117A
LBA - 10E4:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7A ; и нет секторов 1176,
LBA - 10E5:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7A ; 1177 и 178
```

```
LBA - 10E6:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 80 ;
LBA - 10E7:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 80 ;
LBA - 10E8:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 80 ; биение головки
LBA - 10E9:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 7F ;          продолжается
LBA - 10EA:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 80 ;
LBA - 10EB:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81 ;
LBA - 10EC:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; а вот и нулевой трек,
LBA - 10ED:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; он располагается в
                                                           ; секторах
LBA - 10EE:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 86 ; 1185 и 1186 - запомним
LBA - 10EF:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; это обстоятельство!
LBA - 10F0:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8C ; point A0 повторяется…
LBA - 10F1:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8C ; а вместе с ним и все
LBA - 10F2:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B ; остальные point'ы.
LBA - 10F3:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B ; Читая субканальные
                                                           ; данные, дальше мы вновь
LBA - 10F4:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8C ; встретим нулевой трек,
LBA - 10F5:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B ; но уже в других
LBA - 10F6:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B ; секторах. Запомним и
LBA - 10F7:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B ; их для надежности
```

Здесь абсолютные адреса представлены в LBA-форме, причем расхождение между адресом, на который осуществляется позиционирование головки, и адресом, чьи субканальные данные при этом читаются (далее по тексту — дельта) составляет около −400 секторов. Правда, равномерность "часового хода" очень хороша, абсолютные идут кучным пучком строго социалистического характера, хотя TEAC все-таки не без греха и оплошности типа `11:6D`, `11:6E`, `11:6D`, `11:6E` случаются сплошь и рядом. Атрибуты нулевого трека присутствуют в явном виде, что не может не радовать.

А вот привод ASUS ведет себя более "разболтанно" (листинг 6.48).

Листинг 6.48. Результат чтения субканальной информации из Lead-In на приводе ASUS

```
+internal+  Format
|  |  |  |  |  ADR/Control
|  |  |  |  |  |  TNO
|  |  |  |  |  |  |  Point
|  |  |  |  |  |  |  |  +- PLBA -+  +- ALBA -+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
```

```
LBA - 10D3:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; здесь субканальные
LBA - 10D4:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; данные возвращаются
LBA - 10D5:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; еще более
LBA - 10D6:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; беспорядочно, и потому
                                                           ; отделить соседние
LBA - 10D7:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; point'ы друг от друга
LBA - 10D8:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; уже не удается, между
LBA - 10D9:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; тем они лежат по тем
LBA - 10DA:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; же самым ALBA-адресам,
LBA - 10DB:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; что и в предыдущем
LBA - 10DC:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; случае, а потому ALBA-
LBA - 10DD:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6F ; адреса могут служить
LBA - 10DE:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; надежной опорой в
LBA - 10DF:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 73 ; идентификации point'ов,
LBA - 10E0:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; не зависящей от
LBA - 10E1:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; настороения, разбол-
LBA - 10E2:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 79 ; танности и конструк-
LBA - 10E3:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 75 ; тивных особенностей
LBA - 10E4:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7A ; конкретных моделей
                                                           ; приводов, что значи-
                                                           ; тельно упрощает
LBA - 10E5:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 75 ; процедуру проверки
LBA - 10E6:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 80 ; степени лицензионной
                                                           ; "чистоты" анализиру-
LBA - 10E7:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81 ; емого диска…
LBA - 10E8:00 15 00 0C 01 14 00 A1 00 FF FF 6A 00 00 11 75
LBA - 10E9:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7B
LBA - 10EA:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; атрибуты трека 0,
LBA - 10EB:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81
LBA - 10EC:00 15 00 0C 01 14 00 A2 00 00 3B 45 00 00 11 7B
LBA - 10ED:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; обратите внимание, что
LBA - 10EE:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81
LBA - 10EF:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 86 ; они располагаются по
LBA - 10F0:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
LBA - 10F1:00 15 00 0C 01 14 00 00 00 00 34 B3 00 00 11 85 ; тем же LBA-дресам:
                                                           ; 1185h и 1186!
LBA - 10F2:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
```

Здесь: абсолютные адреса также представлены в формате LBA и дельта составляет все те же 400 секторов, однако степень неупорядоченности возвращаемой информации значительно выше и номера секторов начинают потихонечку "плясать" (листинг 6.49) (обратите внимание на поле `ALBA`).

Листинг 6.49. Результат чтения субканальной информации из Lead-In приводом NEC

```
            +internal+  Format
            |  |  |  |  |  ADR/Control
            |  |  |  |  |  |  TNO
            |  |  |  |  |  |  |  Point
            |  |  |  |  |  |  |  |  +- ALBA -+  +- PLBA -+
            |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
LBA - 1171:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; point 64 - это на
LBA - 1172:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; самом деле point A0
LBA - 1173:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; (номер первого трека),
LBA - 1174:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; просто глупый привод
LBA - 1175:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; так нелепо его
LBA - 1176:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; исказил, также
                                                           ; обратите внимание, что
                                                           ; все адреса идут к
LBA - 1177:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; сектору 116E!
LBA - 1178:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00  резкий переход адресов
LBA - 1179:00 15 00 0C 01 14 00 64 00 00 11 6E 00 02 00 00 ; с 116E на 1174 (+6)
LBA - 117A:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00 ; такова дискретность
LBA - 117B:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00 ; SEEK'а привода NEC!
LBA - 117C:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00
LBA - 117D:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00
LBA - 117E:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00
LBA - 117F:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00
LBA - 1180:00 15 00 0C 01 14 00 65 00 00 11 74 00 00 00 00
LBA - 1181:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1182:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1183:00 15 00 0C 01 14 00 02 00 00 11 7F 00 00 34 92 ; 117F - …
LBA - 1184:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1185:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1186:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1187:00 15 00 0C 01 14 00 02 00 00 11 81 00 00 34 92 ; 117F - 1181 диапазон
LBA - 1188:00 15 00 0C 01 14 00 02 00 00 11 7F 00 00 34 92 ; адресов, занятых
LBA - 1189:00 15 00 0C 01 14 00 02 00 00 11 7F 00 00 34 92 ; субканальной
LBA - 118A:00 15 00 0C 01 14 00 02 00 00 11 81 00 00 34 92 ; информацией
LBA - 118B:00 15 00 0C 01 14 00 02 00 00 11 80 00 00 34 92 ; point'а == 2
LBA - 118C:00 15 00 0C 01 14 00 02 00 00 11 81 00 00 34 92
LBA - 118D:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 118E:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; смотрите! резкий пере-
LBA - 118F:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; ход с адреса 1181 на
```

```
LBA - 1190:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; адрес 118B - 10 секто-
LBA - 1191:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; ров пропущено, причем
LBA - 1192:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; это не прсто биение
LBA - 1193:00 15 00 0C 01 14 00 64 00 00 11 8D 00 02 00 00 ; головки - этих секто-
LBA - 1194:00 15 00 0C 01 14 00 64 00 00 11 8D 00 02 00 00 ; ров в субканальных
                                                           ; данных нет вообще!
LBA - 1195:00 15 00 0C 01 14 00 64 00 00 11 8B 00 02 00 00 ; И как раз в них и
LBA - 1196:00 15 00 0C 01 14 00 65 00 00 11 92 00 00 00 00 ; содержатся атрибуты
LBA - 1197:00 15 00 0C 01 14 00 65 00 00 11 92 00 00 00 00 ; трека 0, а раз так, то
LBA - 1198:00 15 00 0C 01 14 00 65 00 00 11 92 00 00 00 00 ; трек 0 все-таки есть
LBA - 1199:00 15 00 0C 01 14 00 65 00 00 11 92 00 00 00 00 ; на диске (иначе бы эти
                                                           ; сектора возвращ.)
```

Здесь: дельта "уползания" составляет порядка 10 секторов, а зачастую даже менее того, однако сама упорядоченность секторов вообще никакая, а нулевых point'ов вообще нет. Сектора с адресами `1185h` и `1186h` (в которых, собственно, и хранятся атрибуты нулевых треков) в наглую отсутствуют — вместо этого привод спозиционировал головку на адреса `118Bh` и `118Dh`, в результате чего количество `64h` point'ов (в "девичестве" — `A0h`) до неприличия возросло. Ко всему прочему, абсолютные адреса секторов по непонятной причине перекочевали в поле относительных адресов, и если бы мы попытались проанализировать субканальную информацию согласно стандарту, у нашей защиты точно бы "съехала крыша".

Итак, несмотря на громоздкость и трудности реализации защитного механизма, он все-таки может быть реализован так, чтобы уверенно отличать ключевой диск на любой из моделей приводов. Но стоит ли овчинка выделки или, говоря другими словами, каким образом защищенный диск можно взломать?

А вот со взломом у нас туго. Да, в принципе такой диск можно скопировать, но только не с помощью CloneCD или Alcohol 120% и не на всех моделях приводов. Для взлома пригодны лишь те приводы, что уверенно читают TOC и возвращают атрибуты нестандартных point'ов, поскольку ко всему этому может привязываться защита. "Постойте! — воскликните вы, — но ведь защита не должна закладываться на доступность TOC и атрибуты нулевого point'а, иначе программа окажется неработоспособна на некоторых моделях приводов! Это верно, однако, если привод соглашается читать TOC — отчего же его не проверить?"

Если привод взломщика не позволяет читать содержимое TOC, взломщик не сможет восстановить оригинальный TOC (ну разве что отдизассемблирует весь защитный механизм целиком) и потому скопированный диск будет работать лишь на его приводе!

Правда при наличии привода, читающего TOC, и "хорошо смазанных подшипников в котелке", копирование защищенного диска осуществляется

очень просто. Достаточно лишь, считав TOC (команда READ TOC), считать и само содержимое диска на субканальном уровне (команды SEEK и READ SUBCHANNEL), а также содержимое основного канала (команда READ CD), после чего остается лишь сформировать CCD-, IMG- и SUB-файлы и с помощью того же копировщика CloneCD записать их на диск. Однако такой взлом "по зубам" далеко не всякому хакеру, а с натиском желторотых пользователей эта защита без труда справится.

Трек с нестандартным номером

Внедрение в TOC фиктивных треков с нестандартными номерами обеспечивает создание стойкой, элегантной и в высшей степени неконфликтной защиты. Но обо всем по порядку. По стандарту номера нормальных треков могут принимать значения в диапазоне от 1 до 99 (4Bh) включительно. Трек номер 0 принадлежит вводной области диска и никогда явным образом не присутствует в TOC (см. *разд. "Диск с нулевым треком" этой главы*). Трек номер AAh (170) принадлежит выводной области диска и также никогда не присутствует в TOC. Треки с номерами A0h, A1h, A2h, B0h, B1h, B2h, B3h, B4h, C1h и C2h используются для хранения служебных данных и интерпретируются особенным образом, специфичным для каждого номера. В частности, стартовый адрес трека A1h содержит в себе номер последнего трека данной сессии, хранящийся в поле PMin.

Приводу запрещается обрабатывать треки с нестандартными ему номерами, и такие треки по стандарту должны игнорироваться. Создавая трек с нестандартным номером, мы можем быть абсолютно уверены в том, что он не "сорвет крышу" приводу и ни в коей мере не нарушит читаемость диска. Правда, существует определенная вероятность, что с течением времени выбранный нами нестандартный номер трека перейдет в разряд стандартных и тогда поведение привода станет непредсказуемым. Однако степень риска крайне невелика — формат носителей CD-ROM не меняется каждый день и уже содержит в себе все необходимое. Навряд ли в него будут добавляться новые point'ы, да еще и совпадающие с нашим. Так что предлагаемая защита вполне соответствует букве и духу стандарта и в категорию "защит, основанных на искаженном TOC'е", она попала чисто случайно, — так уж был скомпонован материал настоящей книги.

Но если защищенный диск вполне корректно обрабатывается приводом, то в чем же трудность его копирования? Дело в том, что о существовании нестандартных треков копировщику не так-то просто узнать. *Даже при чтении TOC'а в "сыром" виде (команда READ TOC формат 2) привод не возвращает атрибуты нестандартных номеров треков*, так что эта "сырость" весьма относительна и дотянуться до них можно лишь на субканальном уровне, читая содержимое области Lead-In командами SEEK и READ SUBCHANNEL (при этом треки с нестандартными номерами должны располагаться не в первой сессии, т. к. ее область Lead-In на интерфейсном уровне недоступна).

Защите будет достаточно лишь просканировать вводную область соответствующей сессии на предмет поиска "своих" номеров треков, и если окажется, что их там нет, то мы имеем дело не с оригиналом, а с копией. Читать субканальные данные вводных областей второй и всех последующих сессий умеют все приводы, правда, некоторые из них делают это очень коряво и "криво". В частности, нестандартный номер `X` может быть замещен номером `64h + (X - A0h)`. Так, например, номер `ABh` интерпретируется приводом NEC как `6Fh`. Следовательно, на выбор нестандартных номеров наложены определенные ограничения и по соображениям совместимости и неконфликтности номера `65h`—`9Fh` лучше не выбирать, т. к. в этом случае привод может ошибочно интерпретировать нестандартный номер трека как стандартный (допустим, мы выбрали номер `66h`, тогда `64h + 66h - A0h == 2Ah` или 42 в десятичной системе исчисления, трек с номером 42 — вполне стандартный трек).

Хорошо, приступаем к нашим традиционным экспериментам. Как говорится, "не откадывай на завтра то, что можно выпить сегодня" (а взлом лазерных дисков — лучший повод для оправдания количества выпитого пива. Кстати, о пиве: если из-под лежачего хакера что-то и течет, то это отнюдь не пиво. Впрочем, не будем отклоняться от темы обсуждения и вернемся к нашим баранам).

Используя CCD-образ, оставшийся от прошлых экспериментов с нулевым треком, изменим строку `Point = 0x00` на `Point = 0xAB` (естественно, вместо `0xAB` вы можете выбрать любое другое значение, но с учетом требований, предъявляемых к нестандартным номерам треков, о которых уже рассказывалось ранее). Что? У вас нет CCD-образа? Тогда сделайте следующее:

1. С помощью своей любимой программы "прожига" подготовьте лазерный диск, содержащий две сессии и два трека — по одному треку в каждой.
2. Снимите образ диска в CloneCD-формате (для чего пригодится не только сам копировщик CloneCD, но и любой другой, совместимый с ним, например, Alcohol 120%).
3. Откройте CCD-файл в текстовом редакторе типа "Блокнот" или нажмите <F4>, находясь в файловом менеджере FAR.
4. Увеличьте значение поля `TocEntries` на единицу.
5. Следом за треком два (`Session = 2`, `Point = 2`) добавьте еще одно `Entry`, чей `Point` был бы равен `ABh` или любому другому выбранному вами значению. Содержимое остальных полей данного `Entry` не критично, и вы вполне можете использовать их для хранения ключевой информации.
6. Увеличьте значения всех последующих `Entry` на единицу, чтобы сохранить преемственность принятой системы нумерации.
7. Нажмите <F2> или <Ctrl>+<S> для сохранения изменений в файле.

Внимание

"Блокнот" под Windows 9x не понимает клавиатурной комбинации <Ctrl>+<S>, а вот его собрат под Windows NT/2000 — понимает.

Как бы там ни было, вы должны получить приблизительно следующий результат (листинг 6.50) (по соображениям экономии места приведен лишь фрагмент).

Листинг 6.50. Нестандартный номер трека в CCD-файле

```
[Entry 12]
Session=2
Point=0xAB
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
PMin=3
PSec=1
PFrame=66
PLBA=13458
```

Записав полученный образ на диск, убедитесь, что он нормально читается всеми, доступными вам приводами и никоим образом ни с чем не конфликтует. А теперь считайте содержимое TOC и посмотрите: присутствует ли там "наш" нестандартный номер трека или нет (листинг 6.51).

Листинг 6.51. Содержимое TOC в "сыром" виде с отсутствующим нестандартным треком

```
KPNC$F:\.PHCK3\src\etc\RAW.CD.READ>CD_RAW_TOC_READ.exe 1.1
RAW TOC READER by Kris Kaspersky
* * * TOC * * *
session number
 | ADR/control
 |  | TNO
 |  |  | point
 |  |  |  | AM:AS:AF
```

```
 |  |  |  |  |  |  | zero
 |  |  |  |  |  |  |  | PM:PS:PF
01 14 00 A0 00 00 00 00 01 00 00
01 14 00 A1 00 00 00 00 01 00 00
01 14 00 A2 00 00 00 00 00 1D 21
01 14 00 01 00 00 00 00 00 02 00
01 54 00 B0 02 3B 21 03 16 0E 22
01 54 00 C0 A2 C8 E0 00 61 1B 15
02 14 00 A0 00 00 00 00 02 00 00
02 14 00 A1 00 00 00 00 02 00 00
02 14 00 A2 00 00 00 00 03 18 17
02 14 00 02 00 00 00 00 03 01 21
02 54 00 B0 04 36 17 01 16 0E 22
```

Хм! Такое впечатление, что нестандартный номер трека вообще не записался на диск. На самом же деле это предположение неверно. Записаться-то он записался, а вот прочитался — едва ли. Привод просто отказался возвращать атрибуты нестандартного номера трека, делая вид, что как будто его и не было. Причем это отнюдь не дефект какой-то одной отдельно взятой модели — так ведут себя все приводы! (Во всяком случае — все, доступные мне, — точно.)

А теперь посмотрим, что даст чтение вводной области второй сессии на субканальном уровне. Естественно, чтобы прочитать субканальные данные вводной области, мы должны знать ее стартовый адрес. Как его определить? Да очень просто. Открываем CCD-файл и ищем `Entry`, чей `Session` равен 1, а `Point` — `A2h` (она содержит стартовый адрес выводной области первой сессии) (листинг 6.52).

Листинг 6.52. Адрес выводной области первой сессии

```
[Entry 2]
Session=1
Point=0xa2
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
```

```
PMin=0
PSec=29
PFrame=33
PLBA=2058
```

Легко видеть, что абсолютный адрес выводной области первой сессии составлет 00:29:33 (или 2058 в LBA-формате). Увеличив его на длину выводной области (*30 сек. или 2250 в LBA-формате*), мы получим абсолютный адрес вводной области второй сессии, который в данном случае равен 00:59:33 или 4308 в LBA-формате.

Что ж! Остается всего лишь запустить утилиту seek_and_Q.exe, *см. прилагающийся к книге компакт-диск*, и проанализировать поток цифр, возникающий на экране (листинг 6.53).

Листинг 6.53. Содержимое TOC, прочитанное на субканальном уровне с нестандартным номером трека внутри, прочитанное на приводе TEAC

```
KPNC$F:\.PHCK3\src\etc\RAW.CD.READ>seek_and_Q.exe 1.1 4308 4444
seek CD-ROM & read Q-subcode by KK
LBA - 10D4:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 6E
…
LBA - 10E9:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 7F
LBA - 10EA:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 85
LBA - 10EB:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81
LBA - 10EC:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 85
LBA - 10ED:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 85
LBA - 10EE:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 86
LBA - 10EF:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
LBA - 10F0:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 86
LBA - 10F1:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
LBA - 10F2:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
```

Ура! Это получилось!!! Нестандартные номера треков все-таки присутствуют в TOC! Ну-ка, посмотрим, как поведут себя другие модели приводов, например, крайне капризный в этом отношении привод NEC (листинг 6.54).

Листинг 6.54. Содержимое TOC, прочитанное на субканальном уровне с нестандартным номером трека внутри, прочитанное на приводе NEC

```
LBA - 1188:00 15 00 0C 01 14 00 66 00 00 11 7A 00 00 3B 45
LBA - 1189:00 15 00 0C 01 14 00 6F 00 00 11 86 00 00 34 B3
LBA - 118A:00 15 00 0C 01 14 00 6F 00 00 11 87 00 00 34 B3
```

```
LBA - 118B:00 15 00 0C 01 14 00 6F 00 00 11 85 00 00 34 B3
LBA - 118C:00 15 00 0C 01 14 00 6F 00 00 11 87 00 00 34 B3
LBA - 118D:00 15 00 0C 01 14 00 6F 00 00 11 85 00 00 34 B3
LBA - 118E:00 15 00 0C 01 14 00 02 00 00 11 80 00 00 34 92
```

Ага! Трека с номером `ABh` здесь все-таки не оказывается, зато присутствует трек с номером `6Fh`, что с учетом преобразования по ранее приведенной формуле дает: `A0h + 6Fh - 64h == ABh`, — как раз то, что нам нужно.

Копирование защищенного диска копировщиком Alcohol 120% и/или CloneCD протекает вполне нормально (никто не виснет, ничего не взрывается и не дымит), однако и нестандартных номеров треков в субканальных данных второй сессии уже не оказывается, и по этим абсолютным адресам содержатся атрибуты уже совсем других треков, смотрите сами (листинг 6.55).

Листинг 6.55. Копия диска уже не содержит нестандартного трека

```
LBA - 10EB:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 7F
LBA - 10EC:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 86
LBA - 10ED:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 86
LBA - 10EE:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 86
LBA - 10EF:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 87
```

Действуя по описанной ранее методике, защитный механизм сможет без труда отличить оригинальный диск от его подлой копии, причем тиражирование оригинальных дисков элементарным образом осуществляется на штатном оборудовании (обычный рекордер плюс копировщик CloneCD или Alcohol 120% — вот и все, что вам нужно). Ничуть не с меньшим успехом защита может использоваться и в производстве штампованных CD, но это уже тема другого разговора.

Памятуя о пословице "чем меньше женщину мы любим — тем больше времени мы спим", перейдем от любви к практике. То есть — ко взлому. Несмотря на высокую стойкость защиты перед существующими на момент написания этих строк автоматическими копировщиками, скопировать оригинальный диск вполне по силам опытному хакеру средней руки. Достаточно лишь запустить анализатор субканальных данных (для этих целей вполне подойдет утилита seek_and_Q) и прочитать "сырой" TOC второй и всех последующих сессий диска (если они есть). Там, среди прочей информации, мы обнаружим уже знакомый нам нестандартный номер трека (листинг 6.56).

Листинг 6.56. Идентификация защищенного диска

```
                                ADR/Control
                                 | TNO
                                 | |  Point
                                 | |  |  +  PLBA  +  +тек. адр.+
                                 | |  |  |        |  |         |
LBA - 10EB:00 15 00 0C 01 14 00 02 00 00 34 92 00 00 11 81
LBA - 10EC:00 15 00 0C 01 14 00 AB 00 00 34 B3 00 00 11 85
LBA - 10EF:00 15 00 0C 01 14 00 A0 00 00 22 92 00 00 11 8B
```

Теперь остается взять CCD-образ, снятый CloneCD с оригинального диска, и добавить в него еще одно `Entry`, несущее в себе трек с нестандартным номером. Легко сказать! Стоит попытаться сделать это, как сразу же возникает масса вопросов. Как перевести субканальные данные в формат, понятный копировщику CloneCD? В какое именно место CCD-файла следует добавлять новый `Entry`? Начнем с ответа на последний вопрос. При просмотре субканальной информации оригинального диска заметно, что трек с нестандартным номером `ABh` следует за треком номер `02h` и предшествует треку `A0h` (см. листинг 6.56). На приводах с разболтанной головкой эту закономерность установить сложнее, т. к. субканальные данные возвращаются в хаотичном порядке и поэтому приходится опираться отнюдь не на те адреса, на которые осуществлялось позиционирование, а на абсолютные адреса, возвращенные в самой субканальной информации. Впрочем, об этом мы уже говорили. Так что открываем CCD-образ, находим в нем местечко между треками `02h` и `A0h` Черт возьми! Если верить CCD-файлу (а точнее, информации, возвращенной приводом по команде READ TOC, на основании которой и создается CCD-файл), то за `Entry` с треком `02h` следует `Entry` с треком `B0h` (стартовый адрес дозаписи), а `Entry` с треком `A0h` (номер превого трека текущей сессии) как раз и есть первый `Entry` данной сессии! И как прикажете это понимать?! А вот как! Стартовый адрес дозаписи в субканальных данных вводных областей вообще не хранится — поэтому-то трека с номером `B0h` тут и нет. Далее. Поскольку TOC каждой сессии дублируется в субканальных данных огромное количество раз, становится понятно, почему за треком `ABh` следует трек `A0h`, — ведь TOC по сути свой как бы "закольцован" и вынужден в угоду надежности и отказоустойчивости "пожирать" свой "хвост".

Таким образом, мы должны найти `Entry` с треком `02h` и добавить вслед за ним `Entry` с треком `ABh`. Поле `Session` должно быть равно 2 (ведь мы читали вводную область второй сессии), поле ADR равно четырем старшим битам поля `ADR/Control`, а после `Contorl` — соответственно, четырем младшим. В нашем случае `ADR/Contorl = 14h`, следовательно, `ADR = 1`, а `Control = 4`. Поле `TrackNo = 0`, поскольку мы находимся в области Lead-In (служебном треке с номером ноль). Поля `AMin`, `ASec`, `AFrame` по идее должны содержать

адрес соответствующих им субканальных данных, однако, по общему соглашению, их принятно считать равными нулю, что соответствует ALBA-адресу в −150 (минус сто пятьдесят). Поле Zero равно нулю по жизни, а поля PMin:PSec:PFrame равны соответствующему им полю PLBA, представленному в формате M:S:F (только помните, что некоторые приводы склонны менять поля PLBA и ALBA местами!). В нашем случае поле PLBA субканальных данных равно 34B3h, что соответствует абсолютному адресу 03:01:66.

Вот, пожалуй, и все. Обобщив весь этот материал, мы получаем следующий симпатичненький Entry, который должен выглядеть приблизительно так, как показано в листинге 6.57.

Листинг 6.57. Формирование Entry для копирования защищенного диска

```
[Entry 12]
Session=2
Point=0xAB
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
PMin=3
PSec=1
PFrame=66
PLBA=13458
```

Теперь, увеличив содержимое поля TocEntries на единицу и перенумеровав все "хвостовые" Entry так, чтобы только что добавленный нами Entry не нарушал их стройной нумерации, запишем отредактированный образ на новую болванку (добавлять поле [TRACK AB] ни в коем случае не надо!).

Если все сделано правильно, то скопированный диск ничем не будет отличаться от оригинального и защита "упадет". Однако взломать ее сможет далеко не всякий. Как вы уже могли убедиться, хакер должен обладать весьма глубокими познаниями в области строения лазерных дисков и особенностями их обработки различными моделями приводов.

Трек с данными, маскирующийся под аудио

Чем отличаются аудиотреки от треков с данными? И что произойдет, если трек с данными пометить как аудиотрек? На первый взгляд, здесь не случится ничего интересного и подопытный трек будет спокойно читаться

командой `READ CD` с той лишь разницей, что автоматическая коррекция ошибок Q- и P-уровней уже не выполняется приводом, но сбойные сектора вполне поддаются восстановлению вручную. Защитный механизм, знающий истинный формат считываемых секторов, такую коррекцию сможет осуществить без труда, а вот программы-копировщики — нет. А потому при многократном перекопировании диска, количество ошибок чтения будет неуклонно накапливаться и на каком-то этапе корректирующих способностей кодов Рида-Соломона окажется недостаточно и очередная по счету копия откажет в работе. Однако при нынешнем качестве оптических носителей при бережном обращении с ними количество ошибок Q- и P-уровней крайне невелико и копии первых трех поколений гарантированно будут читаться даже раздолбанными "узкоглазыми" приводами. Так что хорошей защиты мы из этого не "сварим".

Подобные рассуждения типичны для специалиста средней руки, считающего, что треки с данными отличаются от аудиотреков одним лишь битом поля `ADR/Control` (и это третий, считая от нуля бит), и в то же время удивляющегося, почему его привод "грабит" аудиотреки не вполне корректно. Хорошенько порывшись в спецификациях и отдизассемблировав пару-тройку микропрограмных прошивок, мастера своего дела приходят к выводу, что в обработке аудиотреков и треков с данными присутствует по меньшей мере пять концептуальных различный. Вот они.

- Точное позиционирование на заданный аудиосектор невозможно в принципе, поскольку адреса секторов, "проплывающих" в данный момент над оптической головкой, хранятся не в самом секторе, а в Q-канале подкода, "размазанном" вдоль спиральной дорожки. Упрощенно говоря, субканальные данные из 748 фреймов или 8 секторов (хотя их правильное название — *блоки*) объединяются в *пакеты*, причем каждый такой пакет содержит в себе по четыре точки отсчета, что в идеальном случае соответствует погрешности в ±1 сектор, а на практике погрешность может достигать и больших величин. Стандарт устанавливает предельно допустимый уровень погрешности в ±1 сек (± 75 секторов), однако некоторые приводы показывают значительно худший результат, так, например, мой TEAC "уплывает" вперед аж на ~500 секторов! И у нас нет никакой возможности прочитать сектор с заранее заданным адресом! Такое положение дел, вполне удовлетворяющее запросы аудиотреков, для треков с данными категорически неприемлемо, и потому они вынуждены оснащать свой заголовок специальным полем, содержащим их собственный адрес. Грубая наводка на сектор осуществляется по субканальным данным, а точная — по их заголовку, в результате чего привод всегда считывает именно тот сектор данных, который и был запрошен.
- Понятие "сектора" к аудиотрекам вообще не применимо. В них нет секторов, но есть так называемые *блоки* (blocks), которые состоят из последовательности непронумерованных фреймов, причем границы блоков

категорически не фиксированы и блок имеет право начинаться с любого понравившегося приводу фрейма, расположенного в окрестностях позиционирования. Поэтому при чтении трека с данными в режиме аудио мы не можем быть уверенными, что прочитанные сектора будут начинаться с "головы", а не "хвоста".

- Сектора с данными скремблируются, а аудиосектора — нет, причем необходимость в скремблировании определяется не типом текущего трека, а наличием синхропоследовательности и правильным Mode в его заголовке. *Трек с данными, записанный как аудиотрек, принудительно скремблируется при записи (что "гробит" Mode), но не дескремблируется при его чтении.* Другими словами, с диска читается совсем не то, что на него писалось.
- Некоторые приводы (например, Plextor) осуществляют принудительную аудиокоррекцию записываемых данных, мотивируя это своим стремлением обеспечить более приятное звучание. Неудивительно, что попытка записи трека с данными в режиме аудио приводит к его полному уничтожению.
- Как уже говорилось ранее, аудиотреки не содержат Q- и P-кодов коррекции и потому привод и не корректирует их, что ведет к накоплению ошибок, однако копии первых трех—пяти поколений замечательно читаются и без корректирующих кодов.

Так, теперь понятно, почему задача точного извлечения аудиотреков такая сложная, если не сказать — невозможная. Но как это обстоятельство можно использовать на практике? Давайте запишем трек с данными как аудиотрек и немного поэкспериментируем с ним. Это поможет нам ответить на поставленный ранее вопрос.

Разумеется, непосредственно этого не сделать, и штатные программы "прожига" нашего хитрого замысла просто не поймут, в лучшем случае выдав что-то наподобие "Illegal mode for this track", а в худшем просто обозвав нас дураками. Что ж, придется идти обходным путем. Используя Ahead Nero, Stomp Record Now! или любую другую программу аналогичного назначения, создадим обыкновенный лазерный диск, содержащий один или несколько треков данных. Затем, используя копировщик CloneCD (Alcohol 120%), снимем с диска образ и слегка отредактируем его CCD-файл. Все, что нам надо — найти `Entry`, чей `Point` равен номеру нашего трека, и изменить содержимое поля `Control` на ноль или два (аудитрек без защиты авторских прав и аудиотрек с защитой оных соответственно). Так же следует поменять `MODE` трека с одного на ноль (см. секцию `[TRACK]` листинга 6.58).

Листинг 6.58. Создание образа защищенного диска

```
[Entry 11]          [Entry 11]          [TRACK 1]          [TRACK 1]
Session=2           Session=2           MODE=1             MODE=1
Point=0x02          Point=0x02
```

```
ADR=0x01          ADR=0x01          [TRACK 2]         [TRACK 2]
Control=0x04      Control=0x02      MODE=1            MODE=0
TrackNo=0         TrackNo=0         INDEX 1=0         INDEX 1=0
AMin=0            AMin=0
ASec=0            ASec=0
AFrame=0          AFrame=0
ALBA=-150         ALBA=-150
Zero=0            Zero=0
PMin=3            PMin=3
PSec=1            PSec=1
PFrame=33         PFrame=33
PLBA=13458        PLBA=13458
```

Субканальные данные корректировать не обязательно. Напротив, лучше оставить их в своем неизменном виде. Это запретит проигрывание данного трека в аудирежиме и избавит вас и ваших пользователей от тяжелого психического шока, вызванного прослушиванием "симфонии данных", замаскированных под аудио. Что произойдет, если, скорректировав субканальные данные, проиграть "аудиотрек" на бытовом или компьютерном приводе? Да в общем-то ничего ужасного, но вы когда-нибудь загружали программы с магнитофонной ленты в персональный компьютер типа "Правец-8Д", "Электроника-БК" или "ZX-Spectrum"? А динамик модема во время передачи данных включали? Если да, тогда вы имеете представление о характере звука, который обрушится на ничего не подозревающих пользователей, вставивших такой диск в привод. Так и заикой недолго стать, особенно если регулятор громкости выведен на максимум. Некоторые специалисты утверждают, что существует риск вывода усилителя и колонок из строя, однако, на мой взгляд, подобные слухи беспочвенны и бессмысленны. Ни предельно доступимый уровень выходного сигнала, ни предельно достустимый спектр частот при этом не может быть превышен, и правильно спроектированный усилитель перенесет это испытание без особого вреда для себя. То же самое относится и к колонкам.

Тем не менее, если вы используете режим "цифрового воспроизведения" треков, штатно поддерживаемый Windows 2000, то, независимо от содержимого каналов подкода, треки с данными, помеченные как аудиотреки, воспроизводиться все-таки будут! А некоторые приводы (такие, например, как NEC) воспроизведут эти треки и в аудиорежиме. Что ж, пусть воспроизводят! А на все недовольные вопли ошарашенных пользователей мы ответим, что программа загружает себя через аудиотракт, — как и в былые времена. Ну чем не лекарство от ностальгии?

А теперь прочитаем образ диска с аудиотреком и сличим его содержимое с исходным (см. рис. 6.23). Уверяю вас, результат превзойдет все ваши ожидания! Начнем с побайтового сравнения IMG-файлов, несущих на своих

плечах данные основного канала. Используя любой компаратор (хоть fc.exe из штатной поставки Windows, хоть мой любимый c2u от Профессора Нимнула (Professor Nimnul)), запустим его приблизительно следующим образом: `FC.EXE IMAGE.IMG IMAGE_T.IMG /B > IMAGE.DIF`, где `IMAGE.IMG` — образ, снятый с "нормального" диска, а `IMAGE_T.IMG`, — образ, снятый с диска с "аудиотреком".

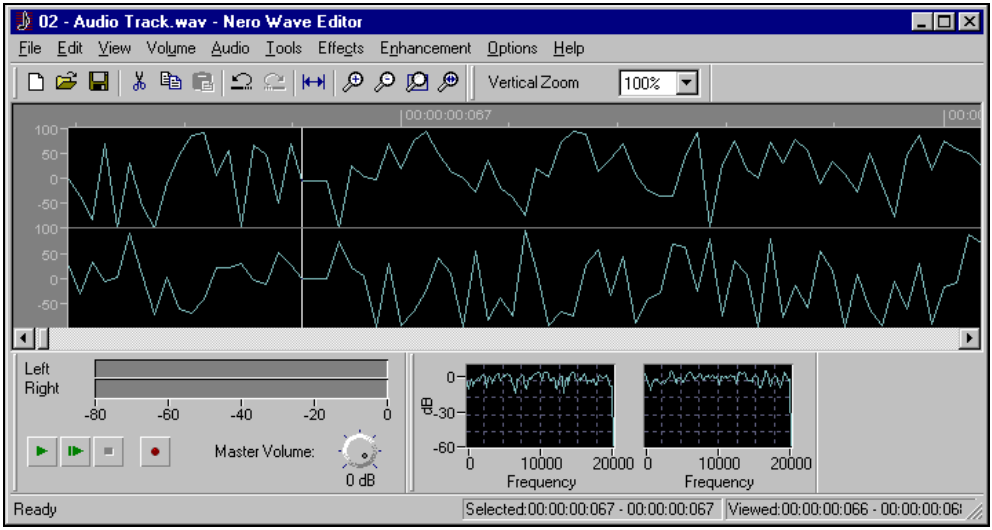


Рис. 6.23. Данные, представленные как аудио

Через несколько минут (утилита fc.exe работает до "ужжжаса" медленно, не то, что c2u) на вашем жестком диске образуется файл IMAGE.DIF с размером в добрые 50 Мбайт. Стало быть, какие-то отличия между "копией" и "оригиналом" все-таки есть, и этих отличий не просто много, а очень-очень много!

Листинг 6.59. Фрагмент оригинального диска

```
0049D2B0:  00 FF FF FF FF FF FF FF | FF FF FF 00 00 29 32 01          )2☺
0049D2C0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
…
0049DBE0:  00 FF FF FF FF FF FF FF | FF FF FF 00 03 01 33 01          ♥☺3☺
0049DBF0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC00:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC10:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC20:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC30:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC40:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049DC50:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

Листинг 6.60. Фрагмент скопированного диска

```
0049D2B0:  00 FF FF FF-FF FF FF FF-FF FF FF 00-00 29 32 01   yyyyyyyyyy  )2☺
0049D2C0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
…
0049DBE0:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 33 61   yyyyyyyyyy ☻?3a
0049DBF0:  00 28 00 1E-80 08 60 06-A8 02 FE 81-80 60 60 28   ( ▲И◘`♠?☻??И``(
0049DC00:  28 1E 9E 88-68 66 AE AA-FC 7F 01 E0-00 48 00 36  (▲z?hfo?u⌂☺a H 6
0049DC10:  80 16 E0 0E-C8 04 56 83-7E E1 E0 48-48 36 B6 96  И▬a♫E♦V?~aaHH6╢Ц
0049DC20:  F6 EE C6 CC-52 D5 FD 9F-01 A8 00 7E-80 20 60 18  oi?IROyY☺? ~И `↑
0049DC30:  28 0A 9E 87-28 62 9E A9-A8 7E FE A0-40 78 30 22  (◙z3(bzй?~?a@x0"
0049DC40:  94 19 AF 4A-FC 37 01 D6-80 5E E0 38-48 12 B6 8D  Ф↓?Ju7☺OИ^a8H↕╢?
0049DC50:  B6 E5 B6 CB-36 D7 56 DE-BE D8 70 5A-A4 3B 3B 53  ╢a╢E6?V??OpZд;;S
```

Прежде всего, бросается в глаза, что в области Pre-gap второго трека, ранее заполненной нулями, теперь появились какие-то данные, по внешнему виду очень смахивающие на "мусор" и не соответствующие никаким данным исходного трека с данными. Выберем любую, наугад взятую последовательность, например, `1E 9E 88 68 66 AE AA` (в тексте листинга 6.60 она выделена полужирным шрифтом), и попытаемся отыскать ее в исходном файле IMAGE.IMG. Ее там не окажется!

Абсолютный адрес сектора, находящийся в его заголовке (в тексте он взят в рамку), также выглядит весьма неадекватно, как студент после хорошей пьянки на следующее утро. Смотрите, поле A-SEC принимает неприлично высокое значение, дотягиваясь аж до `81h`, что представляет собой грубейшую ошибку. Максимум — здесь может находиться `59h`, но никак не больше! Поле Mode, равное в данном случае `61h`, тоже очевидно искажено.

Может быть, это просто маленькая локальная ошибка? Но нет! Просмотр заголовков секторов показывает, что они здесь все такие (листинг 6.61).

Листинг 6.61. Искаженные заголовки секторов

```
0049DBE0:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 33 61   yyyyyyyyyy ☻?3a
0049E510:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 34 61   yyyyyyyyyy ☻?4a
0049EE40:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 35 61   yyyyyyyyyy ☻?5a
0049F770:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 36 61   yyyyyyyyyy ☻?6a
004A00A0:  00 FF FF FF-FF FF FF FF-FF FF FF 00-02 81 37 61   yyyyyyyyyy ☻?7a
```

Вот так дела творятся в Багдаде! Ладно, черт с ними — с заголовками секторов, сейчас нас больше волнует вопрос: в какие-же "тартарары" провались наши исходные данные и что это за "мусор" читается из аудиотрека? Кто же

во всем этом виноват? Дисковые сбои? Аудиокоррекция или все-таки скремблирование?

Скремблирование! Это точно! Обнаружив в заголовке сектора сигнатуру синхропоследовательности `00 FF FF FF FF FF FF FF FF FF FF 00` и `MODE`, равный единице, привод, невзирая на тип трека, заданный в TOC, интерпретировал данный сектор как сектор с данными и отскремблировал все его байты — с 12 по 2351 включительно. Не только пользовательская область данных, но и поле MODE отказалось отскреблированным, и потому при последующем считывании данного сектора с диска его принадлежность к сектору данных оказалось не столь очевидной, и привод, заглянув в TOC, понял, что имеет дело с сектором "аудио", рескремблировать который не нужно. В результате, мы получили на выходе искаженные скремблером данные, которые оказалось некому восстанавливать!

Такая особенность поведения привода не санкционирована стандартом, который эти вопросы описывает слишком неоднозначно и туманно, поэтому часть приводов (и их большинство!) принудительно скремблируют записываемые "аудио" данные, а часть — пишут их так, как есть. Правда, возможность считывания неотскремблированных секторов никем не гарантирована, поскольку они могут содержать регулярные последовательности данных, дезориентирующих считывающий механизм и вводящий его в грубые ошибки. Поэтому для начала лучше поработать с приводами, насильственно скремблирующими сектора. К таковым, в частности, относятся NEC и TEAC.

Пропустив считанные данные через рескремблер, который можно позаимствовать, например, из библиотечки ElbyECC.dll, входящей в состав CloneCD, мы восстановим исковерканные скремблером сектора их исходный вид, с которым наша программа без труда сможет работать. А отдельные дисковые сбои могут быть устранены и вручную, ведь корректирующие коды находятся в нашем распоряжении! Если писать свой собственный декодер Рида-Соломона вам лень, то воспользуйтесь услугами все той же библиотеки ElbyECC.dll (только не забывайте при этом, что распространяя последнюю в составе своего продукта, вы нарушаете авторские права ее создателей).

Кажется, что мы напали на настоящую золотую жилу! Раз содержимое трека с данными, помеченного как аудиотрек, принудительно скремблируется при записи, попытка копирования такого диска приведет к его повторному скремблированию, в результате чего мы получим совершенно другие данные (строго говоря, это будет не совсем "другие" данные — повторное скремблирование равносильно рескремблированию, и исходный "аудиотрек" будет полностью восстановлен, однако поскольку защитный механизм также прогоняет данные через рескремблер, то он сможет работать только со "своим" диском, а не его копией (правда, копия с копии даст желанный результат, но всякий ли до этого догадается?).

Увы! Поскольку поле MODE также скремблируется, считанный с защищенного диска сектор уже не опознается приводом как сектор с данными и его принудительное скремблирование не выполняется, благодаря чему "защищенный" диск копируется вполне нормально. Однако мы проделали слишком большой путь, чтобы вот так запросто сдаваться!

"Аудиотреки", записанные на проводе, который не выполняет их автоматического скремблирования, при попытке копирования на всех остальных приводах приведут к полному провалу, — ведь эти приводы, выполняя скремблирование, необратимо "гробят" содержимое секторов, которое невозможно восстановить даже двукратным перекопированием. Единственный выход — найти нескремблирующий привод (что будет весьма непросто, лично я таких приводов так и не нашел), либо изменить прошивку своего пишушего привода так, чтобы он позволял включать/выключать скремблирование секторов по нашему желанию. Для этого подойдет любой привод, который только можно "прошивать" (например, TEAC). Скачав свежую прошивку с сайта его производителя, "натравите" на него дизассемблер, понимающий "язык" данного процессора, и проанализируйте алгоритм работы микропрограммы. Только поминитe, что некорректно измененная прошивка может полностью вывести привод из строя, поскольку процедура "прошивки" привода в самой "прошивке" и содержится, и если последняя вдруг перестанет работать, перестанут работать и все "артерии" привода. И хотя осуществить задумание вполне реально, квалификация взломщика должна быть очень и очень высока.

К сожалению, разработчики защищаемого приложения находятся ничуть не в лучших условиях, поскольку для записии оригинальных дисков им требуется аналогичный привод, который чрезвычайно трудно застать в продаже и ничуть не легче изготовить самому. С другой стороны — было бы желание, а уж пути для его осуществления завсегда найдутся! Зато данный защитный механизм как нельзя лучше подходит для штампованных CD, на логическую структуру которых вообще не наложено никаких ограничений. Диск, защищенный по данной технологии, скопировать практически нереально.

Теперь перейдем к "слабым" секторам, — то есть секторам, содержащим неблагоприятные для привода последовательности. И одна из таких последовательностей — `...04 B9 04 B9 04 B9...` Нескремблируемый сектор, содержащий такую запись в своем теле, запишется без проблем, но в силу определенных конструктивных ограничений даже лучшими из приводов будет читаться крайне нестабильно, а то и вовсе не будет читаться вообще. Так происходит потому, что физическое представление данной последовательности приводит к образованию длинных цепочек лендов (питов), а приводу для работы жизненно необходим постоянно изменяющийся HF-сигнал (HF — *High Frequency*, высокая частота), и читать однородные области спиральной дорожки он не в состоянии. Подробнее о слабых последовательностях можно прочитать в *разд. "Синхрогруппы, объединяющие биты и DSV" главы 1.* Для нас

же сейчас важен в первую очередь тот факт, что некоторые приводы все-таки ухитряются найти выход из положения, просто меняя стартовую позицию сектора во фрейме, что ведет к колоссальным изменениям на физическом уровне представления информации, и слабая последовательность внезапно перестает быть таковой, нормально читаясь всеми приводами. Но! Скопировать такой диск можно только на том приводе, который умело распознает и корректно обрабатывает слабые последовательности (к таким приводам, в частности, относятся приводы Plextor, за полным списком подходящих для этих целей моделей обращайтесь к справке CloneCD).

С другой стороны, приводы, пишушие слабые последовательности как есть, оказываются невероятно полезными для качественной имитации сбойных секторов (подробнее см. *разд. "Защиты, основанные на физических дефектах" главы 9*), поскольку сектора, содержащие слабые последовательности, не читаемы на физическом уровне. Это вам не тривиальное искажение полей EDC/ECC, легко обнаруживаемое защитным механизмом путем чтения сектора в сыром режиме. Слабые сектора к тому же заставляют привод сбросить скорость и немного поерзать головкой, вызывая тем самым определенную временную задержку — точно такую, какую вызывают настоящие сбойные сектора (и многие защитные механизмы закладываются на это). Сектор с искореженным EDC/ECC, напротив, читается практически мгновенно, чем и выдает себя с головой. Короче говоря, "папуас папуасу друг, товарищ и корм", — слабые сектора служает не только на благо защиты, но и неплохо чувствуют себя в руках "пролетариата" — то бишь хакеров и кракеров, не желающих платить "буржуинам" свои кровные. Надеюсь, вы будете не против немного поэкспериментировать с ними?

Итак, берем наш старый добрый образ оригинального файла (нет, не искореженный скремблированием образ защищенного файла, а образ, снятый с нормального диска), привычным дивжением руки меняем атрибуты трека с данными на аудио, как это мы уже делали ранее, но в дополнение к тому искажаем Sync и/или поле MODE нескольких секторов с заранее известными адресами. Записываем образ на диск и убеждаемся, что теперь их содержимое уже не скремблируется и с диска читается именно то, что мы на него писали (правда, если сектор содержит в себе регулярные последовательности, он может и не прочитаться — все же не от простой жизни секторам с данными скремблирование придумали).

А теперь забьем эти сектора последовательностью `...04 B9 04 B9 04 B9...` и запишем их снова. Если ваш привод не достаточно интеллектуален для того, чтобы выбирать стартовую позицию сектора во фрейме, наши сектора запишутся самым неблагоприятным образом и попытка их чтения даст ошибку! Кстати, если вы густо усеете диск сбойными секторами, — его копирование окажется чрезвычайно затруднено, особенно если расположить слабые сектора группами, размер которых варьируется от 9 до 99 секторов, а за концом каждой группы будет расположен один *ключевой сектор*

(т. е. обыкновенный сектор, содержащий ключевую информацию). Дело тут вот в чем. Умные копировщики (CloneCD или Alcohol 120%), обнаружив, что диск содержит большое число дефектных секторов, на чтение которых уходит колоссальное количество времени, предлагают пользователю задействовать режим быстрого пропуска сбойных секторов, — тогда, встретившись со сбойным сектором, копировщик пропускает 100 последующих секторов, экономя время на попытках их чтения. Защиты, привязывающиеся к настоящим физическим дефектам поверхности, на этом трюке "обламываются" по полной программе (т. к. дефекты имеют тенденцию со временем разрастаться и потому внедрять ключевую информацию в окрестности дефектной области крайне не рекомендуется). Однако слабые сектора не являются дефективными в физическом смысле этого слова, и потому чтению прилегающих к ним секторов ничуть не мешают. А раз так, то мы можем смело закладываться на их существование! Копирование защищенного диска в режиме "быстрого пропуска" пропустит не только "слабые" сектора, но и ключевые метки, а копирование в обычном режиме растянется на несколько часов (если не больше), да и то по причинам, о которых мы поговорим позже.

Некорректный Run-out как средство защиты или X-сектор

С хакерской точки зрения, программа Stomp Record NOW! интересна уже хотя бы тем, что это практически единственная сервисная программа, скрытно записывающая на каждый "прожигаемый" диск специальную метку, — своеобразный "водяной знак", о существовании которого подавляющее большинство пользователей даже и не догадывается и который со всей очевидностью нарушает их privacy. Но все по порядку.

Поддержка *произвольной записи* (Randomly Writable) в CD-RW-дисках реализована посредством механизма *блоков вбега/выбега* (run-in/run-out block), тесно связанных с режимом пакетной записи. Каждый *пакет* (packet) начинается с четырех вбегающих блоков (трех — на носителях DDCD) и завершается двумя блоками-выбегами (тремя — на носителях DDCD) (рис. 6.24). Блоки вбега/выбега представляют собой обыкновенные сектора, но с необычным значением поля MODE в их заголовке (листинг 6.62). В зависимости от режима адресации, блоки вбега/выбега либо адресуются точно так же, как и все остальные сектора, либо же исключаются из адресного пространства. Вне режима пакетной записи блоки Run-in/Run-out практически нигде не используются, однако интересным свойством программы Stomp Record NOW! является ее неявная поддержка блоков Run-out, скрыто внедряемых в конец каждого трека данных, точнее — в предпоследний сектор его области постзазора (Post-gap area), заголовок которого подвергается незначительным искажениям.

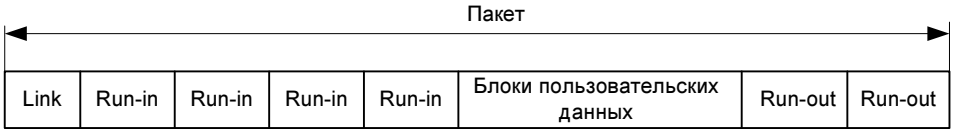


Рис. 6.24. Области вбега и выбега

Листинг 6.62. Расширенная интерпретация поля MODE

```
Bits 7, 6, 5 = 000 - User Data block
= 001 - Fourth Run-in block
= 010 - Third Run-in block
= 011 - Second Run-in block
= 100 - First Run-in block
= 101 - Link block. Physical linking of EFM data
= 110 - Second Run-out block
= 111 - First Run-out block
Bits 4, 3, 2 = 000 - Reserved
Bits 1, 0 = 00 Mode 0 Data
= 01 - Mode 1 Data
= 10 - Mode 2 Data
= 11 - Reserved
```

Поскольку выбегающий блок всего один (тогда так по "Оранжевой книге" их должно быть как минимум два), да и вбегающих блоков не наблюдается, речь идет именно об искажении. Случайным или непреднамеренном — неизвестно. Возможно, это своеобразный "фирменный знак" или "водяная метка" разработчиков, также иногда называемая "пасхальным яйцом", затрудняющая копирование оригинального диска (в рамках данной главы под "оригинальным диском" мы будем понимать диск, созданный с помощью Stomp Record NOW!, а предпоследний сектор области постзазора всякого трека — *"водяным" сектором* или *X-сектором*).

Поле MODEX-сектора, специфицирующее тип данного трека, вместо действительного трека замещается "водяной" константой `E1h` (реже — `E2h`), что соответствует квалификатору первого блока выбега (см. листинг 6.62). Три старших бита определяют конкретный квалификатор блока, а два младших — тип трека. Таким образом, "водяное" число в общем случае равно: `Water Mark Value (WMV) == E0h | Track MODE`, соответственно: `Track MODE == MODE & 3`, где `Track MODE` — тип трека, а `MODE` — значение соответствующего поля заголовка сектора.

Помимо этого, в область данных X-сектора заносится идентификационная строка и серийный номер привода, осуществляющего "прожиг" диска (если только данный рекордер "знаком" Stomp Record NOW!).

"Желтая книга", также известная под именем ECMA-130 (базовый стандарт лазерных дисков с данными), допускала существование лишь трех типов треков: MODE 0, MODE 1 и MODE 2, а все остальные трактовала как ошибку. Приложения, спроектированные в полном соответствии со стандартом ECMA-130, не в состоянии определить действительный тип X-сектора, поскольку не "знают", что шесть старших бит поля MODE должны быть обнулены. Такой сектор и в корзину не выкинешь, и через декодер Рида-Соломона не пропустишь, поскольку заранее не известно: присутствуют ли в нем коды EDC/ECC или же их там нет.

По стандарту области предзазора и постзазора предназначены исключительно для позиционирования оптической головки[2] и не содержат в себе пользовательских данных. При штатной работе привода к этим секторам вообще не происходит никаких обращений (стандарт позволяет закладываться лишь на данные субканалов), благодаря чему "водяные знаки" (равно как и блоки вбега/выбега) никак не сказываются на работе привода и последний их просто не замечает. В то же самое время, содержимое областей пред- и постзазора активно используется различными защитными механизмами для хранения ключевых меток, работающих приблизительно по такому же принципу, что и ключевые метки, записанные в инженерные сектора жестких дисков (71 и/или 72 трека дискеты). Не исключено, что между "водяными знаками" и ключевыми метками возникнет непредвиденный конфликт, приводящий к полной или частичной неработоспособности одной из сторон, так что злоупотреблять данной техникой, право же, не стоит.

Чтобы узнать, присутствуют ли на исследуемом диске "водяные знаки" или нет, мы должны считать предпоследние сектора всех Post-gap в сыром виде, для чего нам пригодится утилита CD_RAW_SECTOR_READ или любая другая, аналогичная ей (очень хорошо подходит для этой цели CloneCD). Остается определить абсолютные адреса конца областей постзазора всех треков. Это легко. Абсолютный адрес последнего сектора области Post-gap равен стартовому адресу следующего трека минус `sizeof(pre-gap)` (стартовому адресу выводной области, если этот трек — последний) минус единица. Соответственно, чтобы выйти на след предпоследнего сектора, полученное значение следует уменьшить на единицу. Другими словами: `x-sector address = (next track != lead-out)?next track address -3:lead out address - 2`.

---

[2] "Древние" приводы были слишком простыми устройствами, и на декодирование субканальной информации — между прочим, размазанной по большой поверхности диска! — им требовалось определенное время, в течение которого оптическая головка успевала отъехать по спиральной дорожке далеко в строну, и потому при попытке позиционирования на начало трека какое-то количество секторов оказывалось неизбежно пропущенным! Вот и пришлось добавить к началу всякого трека несколько витков "пустой" информации, чтобы компенсировать выпадения значащих секторов.

Стартовые адреса всех треков хранятся в point'ах с номерами от 01 до 99 включительно, а стартовый адрес выводной области диска — в point'е с номером `A2h`. Допустим, на исследуемом диске имеется всего один-единственный трек и стартовый адрес выводной области равен *00:29:33* (листинг 6.63), тогда X-сектор будет располагаться по адресу 00:29:31.

Листинг 6.63. Определение адреса выводной области

```
[Entry 2]
Session=1
Point=0xa2
ADR=0x01
Control=0x04
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
PMin=0
PSec=29
PFrame=33
PLBA=2058
```

Отметим, что абсолютному адресу 00:29:31 соответствует LBA-адрес, равный 2056. Запомним это значение, так как оно не раз и не два встретится в наших дальнейших экспериментах. Передав полученный адрес программе CD_RAW_SECTOR_READ, мы через секунду-другую получим на выходе его содержимое. Найти "водяной" сектор в образе диска, снятом с помощью CloneCD (или любой другой аналогичной ей программы), несколько сложнее, но все же возможно. Существует по меньшей мере два пути: зная размер одного сырого сектора (2352 байта) и LBA-адрес "водяного" сектора, мы можем вычислить смещение искомого сектора в файле простым перемножением обеих величин (`2352 * 2056 == 49C980h`). Другой путь: просто поискать контекстным поиском HEX-последовательность `00 FF FF FF FF FF FF FF FF FF FF FF 00 00 29 31`, т. е. `Sync + address`.

Но каким бы путем мы ни шли, результат будет таким, как показано в листинге 6.64 (некорректный номер трека в заголовке сектора — `E1h` вместо `01h` — выделен полужирным шрифтом и взят в рамку — и идентификатор рекордера, на котором выполнялся "прожиг" диска, также выделен полужирным шрифтом).

Листинг 6.64. "Водяные знаки", внедренные в предпоследний сектор области Post-gap программой Stomp Record Now!

```
0049C980:  00 FF FF FF FF FF FF FF | FF FF FF 00 00 29 31 E1                )1c
0049C990:  52 49 44 30 31 00 00 00 | 4E 45 43 00 00 00 00 00   RID01   NEC
0049C9A0:  4E 52 31 31 00 00 00 00 | 02 58 56 00 00 00 00 00   NR11    ☻XV
0049C9B0:  4E 45 43 20 20 20 20 20 | 20 20 20 20 20 20 20 20   NEC
0049C9C0:  20 20 20 20 20 20 20 20 | 20 20 20 20 20 20 20 20
0049C9D0:  4E 52 2D 39 31 30 30 41 | 20 20 20 20 20 20 20 20   NR-9100A
0049C9E0:  32 58 56 32 32 38 31 53 | 31 31 31 20 20 20 20 20   2XV2281S111
0049C9F0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049CA00:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

Штатные копировщики (и Ahead Nero в частности) не копируют содержимое областей пред- и постзазора, и потому "водяные знаки" на скопированных дисках просто отсутствуют! Чтобы отличить оригинальный диск от его копии, защитный механизм должен: 1) используя команду `READ TOC` (format `0x2` — full TOC), считать оглавление диска в "сыром" виде; 2) считать адрес выводной области любой из сессий (например, первой из них); 3) определить адрес "водяного" сектора и, используя команду `READ CD`, считать его в "сыром" виде; 4) проанализировать значение поля MODE (пятнадцатый байт заголовка сектора, считая от нуля), если это поле больше двух, то мы имеем дело с оригиналом или его качественной копией; 5) параноикам можно порекомендовать считать содержимое пользовательской области данных и сличить его с эталоном. Это не добавит защите стойкости (если копировщик скопирует "водяной" маркер в поле MODE, то он скопирует и пользовательскую часть сектора), но, возможно, придаст его разработчику чувство самоуспокоения.

Простейший пример реализации защитного механизма выглядит так, как показано в листинге 6.65 (конечно, условный переход, осуществляющий сравнение поля MODE с эталонной "водяной" константой, очень легко обнаружить и "отломать", поэтому для усиления стойкости защиты следует отказаться от явных проверок и использовать считанный заголовок сектора, например, для расшифровки критических участков кода).

Листинг 6.65. Поиск "водяных знаков", оставленных программой Stomp Record NOW!

```
// ВАЖНЕЙШИЕ КОНСТАНТЫ
#define _WATERMARK          0xE1     // код водяного знака
#define _A2                 3        // смещение point'a A2h в TOC'e
#define _MODE               15       // смещение поля MODE    в заголовке сектора
```

```
#define _M                  8        // смещение поля PMin    в TOC'e
#define _S                  9        // смещение поля PSec    в TOC'e
#define _F                  10       // смещение поля PFrame  в TOC'e

#define argCD     argv[1]
main(int argc, char** argv)
{
     int               a, b, x_sec, LBA_lead_out = 0;
     unsigned char     buf[RAW_SECTOR_SIZE*2];

     // TITLE
     fprintf(stderr,"crackme.68E8B0ABh Record NOW! watermark\n");

     // справка по ключам
     if (argc != 2)    {
          printf("USAGE: crackme.68E8B0ABh.exe CD\n"); return -1;}

     // читаем TOC в сыром виде
     a = cd_raw_toc_read(argCD, buf, RAW_SECTOR_SIZE, W_FULL_TOC);

     if (a != SCSI_OK) {     // операция выполнена успешно?
          fprintf(stderr, "-ERR: read TOC\x7\n"); return -1; }

     // поиск point'a A2h, хранящего стартовый адрес выводной сессии
     for (a = 4; a < buf[0]*0x100L+buf[1]; a+=11)
     {
          // это point A2?
          if (buf[a + _A2] == 0xA2)
          {
               // point A2 найден
               // получаем адрес выводной области первой сессии и сваливаем
               LBA_lead_out=((buf[a+_M]*60+buf[a+_S])*75+buf[a+_F])-150; break;
          }
     }

     // поиск адреса выводной области прошел успешно?
     if (LBA_lead_out == 0) {
          fprintf(stderr,"-ERR: find A2h point\x7\n"); return -1;}

     // вычисляем адрес x-сектора, хранящего водяной знак
     x_sec = LBA_lead_out - 2;
```

```
    // читаем сектор с водяным знаком в сыром виде
    a = cd_raw_sector_read(argCD, buf, RAW_SECTOR_SIZE*2, x_sec, 1, 0xF8);

    if (a != SCSI_OK) {     // чтение сектора прошло успешно?
        fprintf(stderr, "-ERR: read x-sector\x7\n"); return -1;     }

    // проверка на наличие водяного знака
    if (buf[_MODE] != _WATERMARK)
    {
        // это не оригинальный диск
        fprintf(stderr, "hello, hacker!\x7\n");     return 0;
    }

    // это оригинальный диск
    printf("hello, legal user!\n");
}
```

Тестирование копировщиков защищенных лазерных дисков показывает, что "водяные знаки" не копируются CloneCD, который наотрез отказывается обрабатывать такие "неправильные" (с его точки зрения!) сектора, молчаливо приводя их в более потребный вид — автоматически корректирует значение поля MODE и уничтожает всю идентификационную информацию, в результате чего скопированный сектор приобретает следующий вид (листинг 6.66).

Листинг 6.66. На копии диска, полученной CloneCD, "водяные знаки" бесследно исчезают и поле MODE нормализуется! Таким образом, защищаемая программа может легко отличить оригинальный диск от его пиратского дубликата

```
0049C980:  00 FF FF FF FF FF FF FF | FF FF FF 00 00 29 31 01          ♥$ ☺
0049C990:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049C9A0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049C9B0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049C9C0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049C9D0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0049C9E0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

В других случаях скопированный X-сектор вообще не читается и команда `READ CD` с завидным постоянством возвращает малоинформативное сообщение об ошибке типа "MEDIUM ERROR". Причины этого до конца не ясны. Возможно, CloneCD пытается таким образом эмулировать bad-сектор, ошибочно посчитав X-сектор плохим, возможно, это следствие неправильного об-

ращения с пишущим приводом (если `Track MODE > 2`, то скремблирование сектора, записываемого в "сыром" режиме, по идее не выполняется и на диске образуются неблагоприятные регулярные последовательности, которые в силу определенных конструктивных ограничений привода оказывается не так-то просто прочитать).

Но так или иначе, все "водяные знаки" копировщик CloneCD просто "съедает", и защищенный диск оказывается неработоспособным. Конечно, в следующих версиях CloneCD ситуация может круто измениться (поддержка X-секторов не требует кардинальных переработок кода копировщика и может появиться в любое время), и тогда защита, основанная на "водяных" метках, становится бесполезной, кроме того, X-сектора успешно копируются Alcohol 120% и вроде бы CDRWin, так что в чистом виде "водяные знаки" недостаточно эффективны и для усиления защиты их следует комбинировать с защитами других типов (искаженный стартовый адрес первого трека, нулевой трек на диске, фиктивный трек в Post-gap ключевого трека и т. д.). Подобные аддитивные защиты чрезвычайно стойки к копированию и на сегодняшний день не копируются ничем.

Понятие X-сектора

Открытие феномена X-секторов произошло совершенно случайно. В процессе разработки защитных механизмов, основанных на нестандартных форматах диска, автор стремился подобрать такие искажения, которые бы корректно обрабатывались всеми моделями приводов, но в то же время не копировались ни одним из копировщиков. Эксперименты шли с переменным успехом, и многие из защитных механизмов понуро сходили с дистанции, не выдержав чудовищной жесткости испытаний. Одни из них оказывались слишком придирчивыми к оборудованию, и на ряде проводов защищенные диски частично или полностью не читались; другие же копировались широко распространенными копировщиками Alcohol 120% и CloneCD.

Зачастую на защищаемых дисках проявлялись один или два сбойных сектора, поначалу списываемые на физические дефекты носителя. Однако вскоре от этой гипотезы пришлось отказаться, поскольку сбойные сектора появлялись в строго определенном месте, каким при ближайшем рассмотрении и оказался предпоследний сектор области постзазора всякого трека. Да, именно тот самый сектор, чей заголовок был искажен!

Копирование X-секторов. Теоретически, в копировании X-секторов нет ничего сложного. Достаточно подобрать современный читающий/пишущий софт, спроектированный с учетом требований "Оранжевой книги" и знающий о существовании блоков Run-in/Run-out (т. е. обнуляющий тип сектора по двум младшим битам, а не по всему полю MODE целиком). Также копировщик должен аккуратно переносить содержимое областей Pre-gap и Post-gap с оригинала на копию, игнорируя "незлобное ворчание" стандарта по поводу, что никаких пользовательских данных здесь все равно нет.

Как уже говорилось ранее, копировщик CloneCD не удовлетворяет этому требованию — дамп с защищенного диска он снимает вполне корректно, но с прожигом последнего уже не справляется. Программа Alcohol 120% снимает корректный дамп, прожигает корректный диск, победоносно выплевывая работоспособный дубликат. Разумеется, речь идет лишь о копировании "водяных знаков" в чистом виде, но даже незначительное усложнение защитного механизма повергает обоих копировщиков в глубокий ужас, граничащий с глубоким расстройством их нервной системы или, попросту говоря, — зависанием.

Поскольку CloneCD более компетентен в снятии дампов, чем Alcohol 120%, для создания образа защищенного диска лучше всего использовать CloneCD, а для "прожига" — CDRWin или Alcohol 120%. Если же это не поможет, — хакеру придется выбирать между разработкой собственного копировщика и взломом непосредственно самого защитного механизма.

Эксперименты с X-сектором. Любопытным побочным эффектом становится нечитаемость предпоследнего сектора, ранее принадлежавшему Post-gap настоящего трека (его абсолютный адрес равен `&Lead-Out` — 2), — далее по тексту именуемого *X-сектором*. Попытка чтения данного сектора командой `READ CD` приводит к возврату нестандартных сообщений об ошибке, специфичных для каждой конкретной модели привода. Далее (листинги 6.67—6.69) приведены SENSE INFO приводов ASUS, NEC и TEAC. SENSE-INFO, возвращенная приводом ASUS, наиболее интересна (листинг 6.67). Привод "рапортует" об ошибке чтения, но не может вразумительно объяснить, что именно послужило первопричиной ее возникновения; первый байт SENSE-INFO, равный нулю, "торжественно заявляет", что никаких ошибок здесь вообще нет!

Листинг 6.67. SENSE-INFO, возвращенная приводом ASUS

```
-ERR:00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

SENSE-INFO, возвращенная приводом NEC, более информативна (листинг 6.68). SENSE KEY, равный трем, указывает на ошибку типа "MEDIUM ERROR" (ошибка носителя); значения остальных полей — увы — нестандартны (на это указывает первый байт, равный `F0h`) и не могут быть расшифрованы по имеющимся у автора спецификациям.

Листинг 6.68. SENSE-INFO, возвращенная приводом NEC

```
-ERR:F0 00 03 00 00 00 12 0A 00 00 00 00 02 00
```

SENSE-INFO, возвращенная приводом TEAC (листинг 6.69), также указывает на "MEDIUM ERROR" (ошибка носителя), однако никаких других подробностей извлечь не удается.

Листинг 6.69. SENSE-INFO, возвращенная приводом TEAC

```
-ERR:F0 00 03 00 00 00 00 0A 00 00 00 00 11 00
```

Зато позиционирование головки на X-сектор командой SEEK (`2Bh`) с последующим чтением содержимого Q-канала подкода командой `READ SUBCHANNEL` (`42h`) на всех доступных мне приводах проходит успешно.

Используя тот факт, что все известные мне копировщики лазерных дисков (включая такие программы, как CloneCD и Alcohol 120%) всегда читают субканальную информацию в общем потоке данных (т. е. получают ее посредством команды `READ CD`), прочитать субканальную информацию нечитаемого X-сектора они оказываются не в состоянии! Спрятав в Q-канал X-сектора ключевую метку, мы сможем легко отличить копию диска от его оригинала.

Подготовка образа защищенного диска выглядит так. Запустив любой HEX-редактор (например, HIEW), мы открываем IMAGE.SUB, содержащий субканальную информацию, и находим в нем абсолютный адрес X-сектора (в данном случае равный 00:29:31) и, убедившись, что это действительно абсолютный адрес, а не поле контрольной суммы (все поля абсолютных адресов расположены в файле IMAGE.SUB по смещению `0xxxxxx0Ch`), каким-либо образом изменяем одно из полей Q-подканала данного сектора, не забыв соответствующим образом скорректировать и его контрольную сумму. Проще всего просто переставить содержимое Q-подканалов соседних секторов. Давайте, например, поменяем местами сектора 00:29:31 и 00:29:32 (листинг 6.70).

Листинг 6.70. Содержимое оригинального IMAGE.SUB (местоположение будущей метки выделено полужирным шрифтом)

```
0003030C:  41 01 01 00 27 31 00 00 | 29 31 8F AA 00 00 00 00   A☺☺ '1  )1Пк
0003036C:  41 01 01 00 27 32 00 00 | 29 32 51 1B 00 00 00 00   A☺☺ '2  )2Q←
```

Листинг 6.71. Содержимое "помеченного" IMAGE.SUB (метка выделена полужирным шрифтом)

```
0003030C:  41 01 01 00 27 32 00 00 | 29 32 51 1B 00 00 00 00   A☺☺ '2  )2Q←
0003036C:  41 01 01 00 27 31 00 00 | 29 31 8F AA 00 00 00 00   A☺☺ '1  )1_Є
```

Теперь запишем измененный образ на диск и убедимся, что субканальная метка действительно присутствует (листинг 6.72).

Листинг 6.72. Ключевая метка в Q-канале X-сектора (выше) и незащищенный диск (ниже)

```
>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 08 09 00 00 08 09

>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 08 09 00 00 08 08
```

Смотрите, субканальная информация сектора 2056 (`808h` в шестнадцатеричной нотации), утверждает, что LBA-адрес данного сектора равен `809h` (2057 в десятичной нотации), т. е. субканальная информация действительно искажена! Взяв для контраста любой незащищенный диск, мы убедимся, что его субканальная информация верна.

Как уже говорилось ранее, копировщик CloneCD не использует команду READ SUBCHANNEL, а получает субканальную информацию в общем потоке данных. Столкнувшись с нечитабельным X-сектором, CloneCD самостоятельно восстанавливает его субканальную информацию, такой, какой по его мнению она должна быть. Короче говоря, при копировании диска копировщиком CloneCD наша ключевая метка "умирает" и Q-канал X-сектора теперь содержит "верные" данные.

Не верите — смотрите сами (листинг 6.73).

Листинг 6.73. Субканальная информация X-сектора оригинального диска (выше) и его копии, полученной с помощью CloneCD (ниже)

```
>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 08 09 00 00 08 09

>seek_and_Q.exe 1.1 2056
seek CD-ROM & read Q-subcode by KK
00 15 00 0C 01 14 01 01 00 00 08 09 00 00 08 08
```

Для получения более детальной информации давайте запустим CloneCD и, убедившись, что защищенный диск все еще находится в приводе, нажмем на "чтение CD в файл-образ". Субканальная информация, добытая CloneCD, существенно варьируется от одного привода к другому, но во всех случаях она оказывается неверна (сравните это с субканальной информацией, приведенной в листингах 6.67—6.69). Привод ASUS, не поддерживающий режим возвращения субканальных данных в общем потоке, заставил CloneCD их восстаналивать

самостоятельно, используя для этой цели информацию из секторных заголовков. Как следствие — ключевую метку как "ветром сдуло" (листинг 6.74).

Листинг 6.74. Содержимое возвращенного X-сектора, привод ASUS

```
000302AC:  41 01 01 00 27 31 00 00 | 29 31 8F AA 00 00 00 00   A☺☺ '1  )1Пк
0003030C:  41 01 01 00 27 32 00 00 | 29 32 51 1B 00 00 00 00   A☺☺ '2  )2Q←
```

Привод NEC на таком диске вообще "поехал крышей", возвращая "мусор" вместо субканальных данных (листинг 6.75).

Листинг 6.75. Содержимое возвращенного X-сектора, привод NEC

```
000302AC:  01 01 01 00 00 00 00 00 | 02 00 5A 28 00 00 00 00   ☺☺☺     ☻ Z(
0003030C:  01 01 01 00 00 00 00 00 | 02 00 5A 28 00 00 00 00   ☺☺☺     ☻ Z(
```

Привод TEAC корректно возвратил субканальную информацию сектора (X-сектор+1), но конкретно "обломался" на самом X-секторе (листинг 6.76).

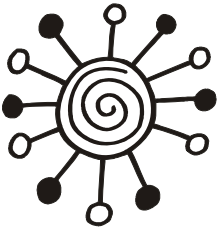
Листинг 6.76. Содержимое возвращенного X-сектора, привод TEAC

```
0003030C:  41 01 01 00 27 31 00 00 | 29 31 8F AA 00 00 00 00   A☺☺ '1  )1Пк
0003030C:  41 01 01 00 27 31 00 00 | 29 31 8F AA 00 00 00 00   A☺☺ '1  )1Пк
```

Таким образом, защита типа *"фиктивный трек в Post-gap настоящего трека с меткой в Q-канале подкода предпоследнего сектора в Post-gap"* (кодовое имя защиты "Лиса") не копируется ни одним известным мне копировщиком лазерных дисков и в то же время не конфликтует ни с каким доступным мне оборудованием (да она и не должна ни с чем конфликтовать!), поэтому "Лису" можно считать достаточно качественной и надежной защитой.

Тем не менее, не стоит переоценивать ее стойкость ко взлому. Взломать "Лису" не просто, а очень просто! Давайте, воспользовавшись парой команд `SEEK` и `READ SUBCHANNAL`, прочитаем субканальную информацию всего диска целиком и сверим ее с содержимым файла IMAGE.SUB, созданного CloneCD или аналогичным ему копировщиком. Вообще-то, считывать всю субканальную информацию совершенно ни к чему, — достаточно проверить на предмет "вшивости" одни лишь сбойные сектора — те, что CloneCD не смог прочесть. Обнаружив ключевую метку, просто скорректируйте соответствующие поля в файле IMAGE.SUB и "запишите" модифицированный образ на диск CD-R/CD-RW. Все! Теперь, с точки зрения защиты, копия и оригинал будут совершенно идентичны друг другу и "Лиса", "стыдливо махнув своим огненным хвостом, скроется в лесу". Единственная проблема состоит в том, что команда READ SUBCHANNAL возвращает субканальную информацию в несколько урезанном виде — без следов наличия контрольной суммы.

Глава 7



Защиты, препятствующие проигрыванию диска в PC CD-ROM

Бытовые *проигрыватели аудиодисков* — намного менее интеллектуальные устройства, нежели компьютерные приводы CD-ROM. Подавляющее большинство из них не поддерживает многосессионные диски и довольно терпимо относится к искажению TOC, поскольку игнорирует добрую половину его содержимого. Таким образом, внося строго дозированные искажения в TOC, можно добиться того, чтобы с точки зрения CD-плееров диск выглядел нормально, но был не читаем на компьютерных приводах CD-ROM.

Конкретные воплощения этой идеи выглядят довольно разнообразно. Вместо качественной музыки вам могут подсунуть специально созданную для компьютерного CD-ROM сессию данных, содержащую сильно сжатый MP3; могут скорректировать указатель на область Lead-Out так, чтобы компьютерный CD-ROM прерывал воспроизведение через несколько секунд после начала проигрывания диска; могут исказить абсолютный стартовый адрес первого трека так, чтобы его LBA-адрес стал отрицательным. Во всех этих случаях нормальное использование диска на компьютерном приводе CD-ROM оказывается невозможным. Тем не менее, искаженный TOC защищенного диска может быть элементарно восстановлен (и чуть позже станет понятно как).

Защиты, базирующиеся на внесении неустранимых ошибок уровня C1/C2, в этом смысле гораздо хуже, и для их взлома понадобится привод, согласный не только обнаруживать ошибки, но и указывать на место их возникновения.

Аудио, перекрываемое данными

Рассмотрим многосессионный диск, состоящий из двух сессий — аудиосессии и сессии с данными. При просмотре такого диска средствами операционный системы Windows мы увидим лишь том с данными, в то время как обыкновенные аудиопроигрыватели будут исправно воспроизводить сессию с аудиотреками, ничего не подозревая о существовании сессии данных (для

совместимости с аудиопроигрывателями сессия с аудиотреками всегда должна располагаться первой). Теоретически аудиотреки могут быть прослушаны путем "ручного" запуска CD-плеера, но практически же это обычно не удается, т. к. защита использует "дополнительные рубежи обороны" (например, урезанная область Lead-In, внесение неисправимых CIRC-ошибок и т. д.), препятствующие такому простому способу "взлома".

Визуально такие компакт-диски распознаются по характерной области Lead-Out, расположенной вблизи внешнего края диска. Это и есть тот барьер, что отделяет аудиосессию от сессии данных. Само по себе наличие "посторонней" области Lead-Out еще не свидетельствует о наличии защиты, она присутствует и на так называемых, дисках *CD-Enhanced*, т. е. дисках смешанного типа (аудио плюс данные), вполне успешно поддерживаемых современными приводами или операционными системами, однако в любом случае — это весьма тревожный симптом и без особой нужны такой диск лучше не покупать (а впрочем, чего уж там, покупайте — все равно взломаем!). Блестящее колечко области Lead-Out, расположенной недалеко от внешнего края лазерного диска и хорошо различимое в отраженном свете, еще не свидетельствует о наличии защиты от воспроизведения, но вероятность ее наличия чрезвычайно велика. В частности, изображенный на фотографии диск (рис. 7.1) защищен Cactus Shield 2.00, — наиболее популярной защитой от цифрового копирования аудиодисков на сегодняшний день.

Рис. 7.1. Диск защищен Cactus Shield 2.00, — наиболее популярной защитой от цифрового копирования аудиодисков на сегодняшний день

Сессия данных может включать в себя все что угодно, вплоть до того, чтобы быть совсем пустой. Но обычно здесь находится сильно сжатое аудио в формате MP3. Причем это MP3 чаще всего записывается не в виде отдельных файлов, которые можно скопировать с диска и выложить в Интернет, а "закатывается" в исполняемый файл-оболочку, запускающийся только с оригинального CD! Естественно, файл-оболочка работает только в среде Windows и пользователям UNIX/Mac остается лишь облизываться или ломать защитный механизм на корню! Ну, если процедура восстановления некорректно записанного компакта, осуществляемая за собственный счет, уже называется *взломом*, то...

Если никаких дополнительных защитных уровней не предусмотрено, то содержимое аудиосесссии может быть элементарно "сграблено" в MP3/WMA. Большинство штатных программ для "прожига" CD-R/RW позволяют просматривать содержимое всех сессий диска и первой сессии в том числе. Вы можете использовать Roxio Easy CD Creator, Stomp Record Now! или любую другую аналогичную программу по своему выбору. В Roxio Easy CD Creator просто выберите в меню **CD** пункт **CD Information** и выделите один или несколько аудиотреков первой сессии, а затем нажмите **Convert Audio** и наслаждайтесь высококачественной музыкой, сграбленной в вашем любимом формате (та версия, Roxio Easy CD Creator, что поставляется вместе с рекордером PHILPS, не поддерживает формат MP3 и предлагает на выбор всего два варианта: WMA и WAV).

Как вариант можете создать исправленную копию защищенного диска, удалив из него лишнюю сессию с данными. Программа CloneCD способна справиться с этим и автоматически. Просто найдите в окне **Profile parameters** (Параметры профиля) вкладку **Audio read parameters** (Параметры чтения аудио), а в ней — флажок **Read first session only** (Чтение только первой сессии). Установите его и нажмите кнопку **OK** для подтверждения. Копирование диска будет протекать в обычном режиме, но в результате него, вы получите обыкновенный аудиодиск безо всякого постороннего "мусора".

Копировщик Alcohol 120% не поддерживает такой возможности, но позволяет добиться аналогичного результата вручную. Осуществляется это следующим образом: на первом этапе мы должны получить образ диска, записанный в формате CloneCD (Alcohol 120% это позволяет). Затем отредактируем CCD-файл так, чтобы вычеркнуть из него все упоминания о сессии (сессиях) с данными. Прежде всего, мы должны уменьшить значение поля Session с двух до одного. Затем удалить из файла все Entry, в которых значение поля Session больше единицы. Теперь скорректируем значение поля TocEntries, сократив его на величину удаленного количества Entry. Остается удалить лишь один или несколько последних треков с данными. Сквозная нумерация треков несколько осложняет эту, казалось бы, с виду простую задачу, поскольку мы не можем быстро установить, какой трек к какой сессии принадлежит. Поэтому приходится либо подсчитывать количество

треков вручную (поля Point со значением больше нуля, но меньше `0x64`), либо удалять все треки, чей Mode не равен нулю. Конкретный пример работы с CCD-файлом приведен в листинге 7.1.

Листинг 7.1. Коррекция CCD-файла для взлома диска

```
[CloneCD]
Version=3
[Disc]
TocEntries=24
Sessions=2
DataTracksScrambled=0
CDTextLength=0
[Session 1]
PreGapMode=0
PreGapSubC=0
[Session 2]
PreGapMode=2
PreGapSubC=0
[Entry 0]
Session=1
Point=0xa0
ADR=0x01
Control=0x00
TrackNo=0
AMin=97
ASec=26
AFrame=66
ALBA=-11634
Zero=0
PMin=1
PSec=32
PFrame=0
PLBA=6750
…
[Entry 20]
Session=2
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=72
```

```
[CloneCD]
Version=3
[Disc]
TocEntries=20
Sessions=1
DataTracksScrambled=0
CDTextLength=0
CDTextLength=0
PreGapMode=0
PreGapSubC=0
[Session 2]
PreGapMode=2
PreGapSubC=0
[Entry 0]
Session=1
Point=0xa0
ADR=0x01
Control=0x00
TrackNo=0
AMin=97
ASec=26
AFrame=66
ALBA=-11634
Zero=0
PMin=1
PSec=32
PFrame=0
PLBA=6750

[Entry 20]
Session=2
Point=0xa0
ADR=0x01
Control=0x04
TrackNo=0
AMin=72
```

ASec=22	~~ASec=22~~
AFrame=38	~~AFrame=38~~
ALBA=325538	~~ALBA=325538~~
Zero=0	~~Zero=0~~
PMin=16	~~PMin=16~~
PSec=32	~~PSec=32~~
PFrame=0	~~PFrame=0~~
PLBA=74250	~~PLBA=74250~~
…	
[Entry 23]	~~[Entry 23]~~
Session=2	~~Session=2~~
Point=0x10	~~Point=0x10~~
ADR=0x01	~~ADR=0x01~~
Control=0x04	~~Control=0x04~~
TrackNo=0	~~TrackNo=0~~
AMin=72	~~AMin=72~~
ASec=23	~~ASec=23~~
AFrame=17	~~AFrame=17~~
ALBA=325592	~~ALBA=325592~~
Zero=0	~~Zero=0~~
PMin=73	~~PMin=73~~
PSec=54	~~PSec=54~~
PFrame=38	~~PFrame=38~~
PLBA=332438	~~PLBA=332438~~
[TRACK 1]	[TRACK 1]
MODE=0	MODE=0
FLAGS= DCP	FLAGS= DCP
INDEX 1=0	INDEX 1=0
[TRACK 2]	[TRACK 2]
MODE=0	MODE=0
FLAGS= DCP	FLAGS= DCP
INDEX 1=19173	INDEX 1=19173
…	
[TRACK 16]	[TRACK 16]
MODE=2	MODE=2
INDEX 1=0	INDEX 1=0

Некоторые диски содержат зловредную программу, автоматически запускающуюся при загрузке диска и скрытно внедряющуюся в оперативную память компьютера. Ее основная задача — охрана аудиотреков от цифрового грабежа. Конкретные реализации такого "сторожа" довольно различны. Можно, например, периодически сканировать список окон верхнего уровня

(см. описание функции `FindWindows` или `EnumWindows` в Platform SDK) на предмет поиска заголовков наиболее популярных "грабилок". Хороших программ для работы с цифровым аудио существует не так уж и много, поэтому вполне реально научиться опознавать их всех. Коль скоро окно программы найдено, защита может вытворять с ним все что угодно. В частности, для принудительного завершения приложения ему достаточно послать сообщение `WM_DESTROY`.

Урезанный Lead-Out

Другой популярный способ воспрепятствовать прослушиваю аудиодисков в компьютерных CD-приводах состоит в искажении содержимого TOC таким образом, чтобы указатель на область Lead-Out указывал не на реальную область Lead-Out, а на гораздо более близкое к началу диска место. Подавляющее большинство бытовых аудиопроигрывателей игнорирует значение этого поля (оно им действительно ни к чему), а вот компьютерные приводы CD-ROM при достижении фиктивной области Lead-Out ведут себя менее предсказуемо. Часть из них послушно прерывает воспроизведение в указанное в TOC время. Часть — просто "зависает", тщетно пытаясь найти фиктивную область Lead-Out там, где ее и в помине нет (причем это может случиться и до начала проигрывания диска, — сразу же после загрузки диска в привод!). Правда, некоторые приводы все-таки "догадываются" о том, что TOC предумышленно искажен (а может, просто игнорируют его) и успешно обходят такую защиту, но закладываться на это я бы все-таки не стал.

Для создания защищенного диска нам потребуется любой обычный аудиодиск (который мы, собственно, и будет защищать) и какой-нибудь копировщик защищенных программ (например, Alcohol 120% или CloneCD). Поддержка режима RAW DAO пишущим приводом необязательна.

Сняв образ защищаемого диска, откроем сформированный копировщиком CCD-файл и найдем в нем следующую текстовую строку `point=0xa2`. Это и есть тот entry, который собственно и указывает на область Lead-Out, чей абсолютный адрес хранится в полях PMin, PSec и PFrame, представляющих собой минуты, секунды и фреймы соответственно. Уменьшим абсолютный адрес до любого разумного времени (например, до 28 сек) и, не забыв сохранить изменения в файле, запишем отредактированный образ на диск CD-R/CD-RW. Дав записанному диску немного остыть (шутка!), извлечем его из привода и установим его в бытовой аудиопроигрыватель. С вероятностью, близкой к единице, защищенный диск должен воспроизводиться вполне нормально (однако шансы нарваться на конфликт у вас все-таки есть). Теперь вернем "подопытного" обратно в компьютерный CD-ROM. Привод, едва начав воспроизведение, послушно прерывает проигрывание диска на 28-й секунде.

На самом же деле такую защиту очень легко обойти. Сняв образ диска с помощью копировщика CloneCD, просто отредактируйте TOC, установив

указатель Lead-Out на его законное положение. Для определения последнего достаточно вручную просмотреть содержимое Q-канала подкода в поисках трека, поле TNO которого содержит значение `0xAA`. Впрочем, копировщик CloneCD способен снимать такие защиты и самостоятельно.

Отрицательный стартовый адрес первого аудиотрека

Еще один извращенный трюк: установить TOC-адрес первого аудиотрека в область, предшествующую Lead-In, то есть, попросту говоря, присвоить этому треку отрицательное смещение (рис. 7.2).

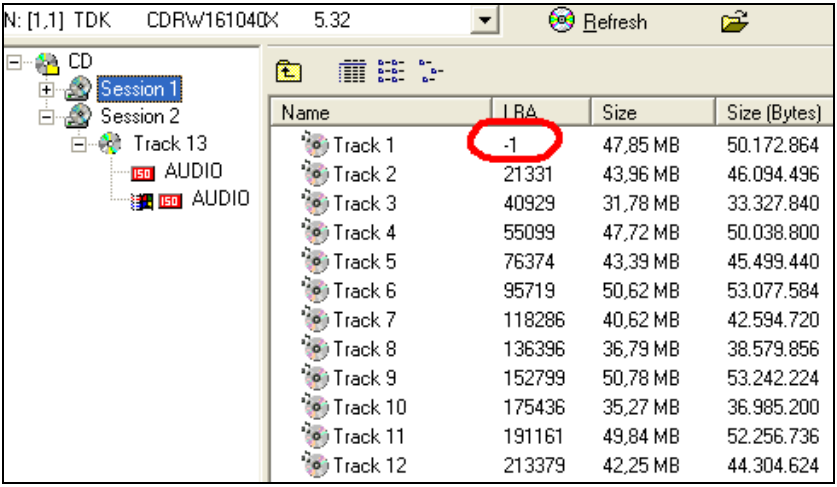
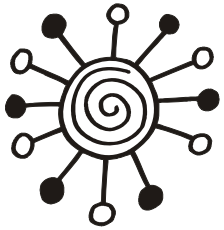


Рис. 7.2. Отрицательный стартовый адрес первого трека препятствует проигрыванию диска в PC CD-ROM

Ломается такая защита точно так же, как и предыдущие, поэтому не будем на ней подробно останавливаться.

Глава 8



Защиты от пофайлового копирования диска (защиты уровня файловой системы)

Качественные и грамотно спроектированные защитные механизмы обычно содержат две независимых "линии обороны", работающих на секторном и на файловом уровнях соответственно. Защиты секторного уровня ориентированы на борьбу с посекторными копировщиками, копирующими весь диск целиком, то есть создающими его точную копию, а защиты файлового уровня, в свою очередь, предотвращают пофайловое копирование диска, равно как и выдирание с диска отдельных файлов.

Может показаться, что при наличии качественной секторной защиты, файловая защита абсолютно бесполезна (тем более что все файловые защиты элементарно обходятся даже штатными копировщиками, такими, например, как Roxio CD Copier). Однако это не так! Невозможность создания точной копии диска еще не означает невозможности грабежа хотя бы *части* его содержимого. Никто не спорит, что часть меньше целого, но все же это лучше, чем совсем ничего. Хотите наглядный пример? Пожалуйста! Защитные механизмы мультимедийных энциклопедий "Наутилус Помпилиус — Погружение" и "Агата Кристи — Виртуальный Концерт" работают лишь на секторном уровне, но никак не препятствуют пофайловому копированию содержимого диска на винчестер или другой CD. Конечно, полученная "копия" оказывается неработоспособной, и при запуске мультимедийной оболочки защита, смачно ругнувшись на пиратскую копию, аварийно завершает свою работу. Тем не менее, с обоими дисками можно работать и без оболочки, поскольку все альбомы/фотографии/видеоклипы хранятся в открытом, незашифрованном виде в формате WAV/BMP/AVI соответственно. Даже неквалифицированный пользователь без труда получит понравившиеся ему файлы с диска и сможет делать с ними все, что захочет!

Отсюда мораль: *не стоит путать защиту носителя (лазерного диска) с защитой его содержимого*. Для защиты содержимого существует множество путей.

Искажение размеров файлов

Еще (или, скорее, уже) во времена монохромных терминалов и 8" и 5.25" дискет существовал некрасивый, но элементарно реализуемый защитный примем, препятствующий пофайловому копированию носителя. Внося определенные искажения в структуры файлов системы, разработчики "грохали" дискету ровно настолько, чтобы работа с ней становилась возможной лишь при условии учета характера внесенных искажений. Защищенная программа, знающая об искажениях файловой структуры, работала с ней без проблем, то штатные утилиты операционной системы на таких дисках конкретно "обламывались", а общедоступных "хакерских" копировщиков в те времена еще не существовало.

Несколько файлов зачастую ссылались на общие для всех них кластера, тогда запись данных в один файл приводила к немедленному их появлению в другом файле, что так или иначе могла использовать защита. Естественно, после копирования файлов на новый диск, пересекающиеся кластеры "разыменовывались" и хитрый способ неявной пересылки данных переставал работать, а вместе с ним переставала работать и сама защищенная программа. Если, конечно, содержимое диска вообще удавалось скопировать. Ведь копирование файлов с пересекающимися кластерами приводило к тому, что эти кластера многократно дублировались в каждом копируемом файле, в результате чего их суммарный объем подчас увеличивался настолько, что емкости тогдашних носителей попросту не хватало для его вмещения! Если же последний кластер файла "приклеивался" к его началу (т. е. файл попросту зацикливался), то объем и время его копирования обращались в бесконечность. Конечно, дисковые доктора в то время уже существовали, но их использование не давало желаемого результата, т. к. лечение файловой системы приводило к полной неработоспособности защиты (в том же случае с зацикливанием — если защита закладывалась на то, что за концом файла следует его начало, то после обработки диска "доктором" осуществление этого приема становилось невозможным со всеми вытекающими последствиями).

Файловые системы лазерных дисков, конечно, совсем не те, что на гибких дисках, но общие принципы их искажений достаточно схожи. Увеличивая фиктивные длины защищаемых файлов на порядок-другой, разработчик защиты может довести их суммарный объем до нескольких сотен гигабайт, так что для копирования защищенного диска понадобится по меньшей мере пачка DVD-дисков или винчестер солидного объема. Защитный механизм, "помнящий" оригинальные длины всех файлов, сможет работать с ними без проблем, но все файловые копировщики не поймут юмора и "поедут крышей".

В принципе, выход за границы файла ничем не чреват. Файловые системы лазерных дисков очень просты. Лазерные диски не поддерживают фрагментацию файлов, а потому не нуждаются в *FAT* (File Allocation Table). Все файлы

занимают непрерывный ряд секторов, и с каждым файлом связаны только две важнейшие характеристики: *номер первого сектора* файла, заданный в *LBA* (Logical Block Address), и его *длина*, заданная в байтах. Остальные атрибуты, вроде имени файла и времени его создания — не в счет, мы сейчас говорим исключительно о секторах.

Увеличение длины файла приводит к "захвату" того или иного количества примыкающих к его "хвосту" секторов, и при условии, что номер последнего сектора, принадлежащего файлу, не превышает номера последнего сектора диска, копирование файла в принципе протекает нормально ("в принципе" потому, что в копируемый файл оказывается включены все файлы, встретившиеся на его пути). Если же в процессе своего копирования файл "выскакивает" за конец диска, привод сигнализирует об ошибке и прекращает чтение. Штатный копировщик операционной системы (равно как и большинство оболочек сторонних производителей) автоматически удаляет "огрызок" недокопированного файла с диска, в результате чего пользователь остается вообще ни с чем. Впрочем, написать свой собственный копировщик — минутное дело, но как узнать, сколько именно байт следует скопировать? Как определить: где идут полезные данные, а где начинается послехвостовой "мусор" (over-end garbage). Вот этим мы сейчас и займемся!

Стоп! Но ведь далеко не у всех есть лазерные диски, защищенные подобным образом. Что ж, сейчас они будут! Возьмем любой незащищенный диск и самостоятельно защитим его (все-таки эта книга посвящена технике защиты лазерных дисков от копирования, а не технике их взлома). Задачей номер один будет получение образа диска, который вы собрались защищать. Лучше всего это делать с помощью программы *Roxio Easy CD Creator* или аналогичной ей. Копировщик *Clone CD* для этой цели непригоден, т. к. он наотрез отказывается осуществлять короткое чтение секторов (т. е. user data only) и всегда обрабатывает сектора целиком, принудительно записывая в конец каждого сектора контрольную сумму и корректирующие коды. В результате, все наши манипуляции над пользовательской областью сектора не возымеют никакого действия и будут налету исправлены микропроцессорной начинкой привода. Конечно, контрольную сумму и корректирующие коды можно рассчитать заново, но зачем понапрасну усложнять себе жизнь? Если у вас нет Easy CD Creator'a, — возьмите *Alcohol 120%*, выбрав из всех предлагаемых им форматов "Стандартные образы ISO".

Хорошо, будем считать, что образ диска успешно сохранен в файл trask.iso, с которым мы сейчас и будем работать. Откроем его в HIEW или любом другом HEX-редакторе и найдем сектор, содержащий оглавление диска. Хорошенькое дело! А как нам его найти? Требуется как минимум полистать спецификацию файловых систем Joliet или ISO-9660 или немного подумать головой. Поскольку размер файла задается в байтах, а не секторах (в секторах он задаться никак не может, время файловых систем, измеряющих файлы блоками, давно прошло), то соответствующее поле можно найти тривиальным

контекстным поиском. Выберем файл, длину которого мы хотим изменять, и запишем ее в шестнадцатеричном виде. Пусть для определенности это будет файл 01 — Personal Jesus.mp3 с длиной в 3591523 байт. В шестнадцатеричной нотации с учетом обратного порядка байт она будет выглядеть так: `63 CD 36 00`. Нажимаем <F7> и вводим искомую последовательность.

Листинг 8.1. Первое вхождение искомой последовательности в образе диска

```
0000CBD0:  07 06 14 38 16 0C 02 00 | 00 01 00 00 01 01 01 30   •♠¶8▬♀☻ ☺ ☺☺☺0
0000CBE0:  00 91 01 00 00 00 00 01 | 91 63 CD 36 00 00 36 CD    C☺    ☺Cc═6  6═
0000CBF0:  63 67 06 1D 17 0D 0A 28 | 0C 00 00 00 01 00 00 01   cg♠↔↨♪◙(♀   ☺ ☺
0000CC00:  0E 30 31 30 5F 30 30 30 | 31 2E 4D 50 33 3B 31 00   ♫010_0001.MP3;1
```

Листинг 8.2. Второе вхождение искомой последовательности в образе диска

```
00010370:  01 00 00 00 00 01 91 63 | CD 36 00 00 36 CD 63 67   ☺    ☺Cc═6  6═cg
00010380:  06 1D 17 0D 0A 28 0C 00 | 00 00 01 00 00 01 32 00   ♠↔↨♪◙(♀   ☺ ☺2
00010390:  30 00 31 00 20 00 2D 00 | 20 00 50 00 65 00 72 00   0 1  -   P e r
000103A0:  73 00 6F 00 6E 00 61 00 | 6C 00 20 00 4A 00 65 00   s o n a l   J e
000103B0:  73 00 75 00 73 00 2E 00 | 6D 00 70 00 33 00 3B 00   s u s . m p 3 ;
```

Искомое значение действительно присутствует в образе, причем не в одном, а в двух нет, даже в *четырех* экземплярах! Нет, это не чертовщина — все так и должно быть. Современные лазерные диски содержат две файловых системы: одна из которых — ISO-9660 — записывается на диск исключительно для его совместимости с устаревшим программным обеспечением, ограничивающим максимальную длину файла одиннадцатью символами (восемь из которых приходятся собственно на само имя, а оставшиеся три — на расширение). Современное программное обеспечение работает с более "продвинутыми" файловыми системами, к которым принадлежит и файловая система Joliet (Джульетта), разработанная компанией Microsoft. Говорите, "только Ромео для полноты компании не хватает"? А ведь файловая система Romeo (Ромео) действительно есть и разработана она компанией Adaptec. К сожалению, этот "Ромео" не получил большого распространения и скоропостижно скончался, так что Джульетта осталась в одиночестве.

Но довольно романтики, возвращаемся к делу. Заботиться о синхронизации обоих файловых систем в общем-то и необязательно, т. к. операционная система Windows "видит" только Joliet и игнорирует ISO-9660, а MS-DOS поступает с точностью до наоборот. Поэтому, если мы увеличим длину файла в Joliet, но "забудем" внести соответствующее изменения в ISO-9660 (а некоторые разработчики защит именно так и поступают!), Windows

не заподозрит в этом и тени обмана. Вот хакеры — другое дело! Оригинальные длины файлов, оставленные в ISO-9660, значительно упрощают задачу взломщика, и потому оставлять их там ни в коем случае не стоит! К тому же, существуют драйвера, позволяющие вручную выбирать какую из имеющихся файловых систем следует монтировать. Так что не будем лениться и скорректируем оба значения сразу, изменив два старших байта с `36 00` на `FF 66` (естественно, вы можете предпочесть и другое значение). Когда будете это делать, обратите внимание на двойное слово `00 36 CD 63` — это тоже длина файла, но записанная в "противоестественном" для IBM PC порядке. Здесь младший байт располагается по *большему* адресу. Адрес стартового сектора файла также записан в двух вариантах. Очевидно, такая схема представления информации выбрана по соображениям переносимости, и каждая платформа вправе выбирать наиболее естественный для нее порядок байтов, однако не факт, что операционная система Windows выберет вариант "младший байт по меньшему адресу". Все решает файловый драйвер, а он в зависимости от особенностей реализации может работать с любым из этих полей. Поэтому оба этих поля всегда должны быть согласованы.

Теперь исправленный (в смысле искаженный) ISO-образ можно смело записывать на CD-R/CD-RW-диск или смонтировать образ на виртуальный CD-привод (для этого вам понадобится Alcohol 120% или его аналоги). Даем команду `DIR` и смотрим (листинг 8.3).

Листинг 8.3. Размер файла Personal Jesus.mp3 умышленно искажен

```
> dir N:\Depeche Mode
Том в устройстве N имеет метку 030706_2038
 Серийный номер тома: 61A1-A7EE

 Содержимое папки N:\Depeche Mode

06.07.2003  21:56       <DIR>          .
06.07.2003  21:56       <DIR>          ..
01.01.1601  04:00        1 728 040 291 01 - Personal Jesus.mp3
30.06.2003  00:11            3 574 805 02 - See You.mp3
30.06.2003  00:12            3 472 405 03 - Strangerlove.mp3
30.06.2003  00:12            3 718 165 04 - Enjoy The Silence.mp3
30.06.2003  00:13            2 956 643 05 - The Meaning Of Love.mp3
30.06.2003  00:14            3 820 565 06 - Master and Servant.mp3
30.06.2003  00:15            3 066 149 07 - Never Let Me Down Again.mp3
30.06.2003  00:16            3 806 772 08 - Its Called a Heart.mp3
30.06.2003  00:16            3 813 460 09 - Little 15.mp3
30.06.2003  00:17            3 574 805 10 - Everything Counts.mp3
```

```
30.06.2003  00:18            3 687 236 11 - People Are People.mp3
30.06.2003  00:19            4 916 036 12 - The Thing You Said.mp3
30.06.2003  00:20            4 182 100 13 - Agent Orange.mp3
30.06.2003  00:21            4 585 012 14 - World in my Eyes.mp3
30.06.2003  00:22            3 646 276 15 - Behind The Wheel.mp3
30.06.2003  00:22            3 049 012 16 - Black Celebration (live).mp3
30.06.2003  00:23            3 800 085 17 - Nothing.mp3
30.06.2003  00:25            7 151 700 18 - Bonus (unnamed).mp3
              18 файлов  1 794 861 517 байт
               2 папки               0 байт свободно
```

Вот это да! Размер файла увеличился до 1 728 040 291 байт (см. выделенную полужирным шрифтом строку листинга), что более чем в два с половиной раза превышает объем всего лазерного диска. А еще говорят, что часть не может быть больше целого! Естественно, попытка скопировать файл на винчестер заканчивается провалом и приходится искать обходные пути. Будем исходить из того, что файлы на диске располагаются последовательно, т. е. за последним сектором одного файла непосредственно следует стартовый сектор следующего. А поскольку стартовые сектора всех файлов нам известны, определение номеров последних секторов для всех файлов, за исключением самого последнего, не составит никакого труда.

Скопируем ISO-образ защищенного диска в файл и рассмотрим его каталог еще раз (листинг 8.4).

Листинг 8.4. Подследственный фрагмент "препарируемого" образа файла

```
0000E040:  00 01 01 01 54 00 94 01 | 00 00 00 00 01 91 63 CD   ☺☺☺T Ф♦    ☺Cc═
0000E050:  FF 66 66 FF CD 63 00 00 | 00 00 00 00 00 00 00 00    ff ═c
0000E060:  01 00 00 01 32 00 30 00 | 31 00 20 00 2D 00 20 00   ☺  ☺2 0 1   -
0000E070:  50 00 65 00 72 00 73 00 | 6F 00 6E 00 61 00 6C 00   P e r s o n a l
0000E080:  20 00 4A 00 65 00 73 00 | 75 00 73 00 2E 00 6D 00     J e s u s . m
0000E090:  70 00 33 00 3B 00 31 00 | 46 00 6B 08 00 00 00 00   p 3 ; 1 F k◘
0000E0A0:  08 6B 15 8C 99 00 00 99 | 8C 15 67 06 1D 17 0B 1C   ◘k§MЩ  ЩM§g♠↔↨♂└
0000E0B0:  0C 00 00 00 01 00 00 01 | 24 00 30 00 32 00 20 00   ♀   ☺   ☺$ 0 2
0000E0C0:  2D 00 20 00 53 00 65 00 | 65 00 20 00 59 00 6F 00   -   S e e   Y o
0000E0D0:  75 00 2E 00 6D 00 70 00 | 33 00 3B 00 31 00 50 00   u . m p 3 ; 1 P
```

Наименьший номер стартового сектора файла, следующий за сектором `0191h`, равен `086Bh`. Таким образом, файл 01 — Personal Jesus.mp3 не может содержать более `086Bh - 0191h == 6DAh` секторов или `1754 * 2048 == 3 592 192` байт. Конечно, это несколько завышенная оценка, и действительный размер файла на 1,5 Кбайта короче, но такое расхождение уже не критично. Большинство

мультимедийных файлов будут вполне нормально обрабатываться даже при наличии некоторого количества постороннего "мусора" на хвосте. Исправив образ файла, запишем его на диск или просто усечем файл до необходимых размеров с помощью любой подручной утилиты типа "Добермана Пинчера" ("Pinch of File").

А что делать, если вас не устраивает столь низкая стойкость подобной защиты? Ну, кое-что вы можете сделать. Например, уменьшить стартовые сектора некоторых файлов, убивая тем самым сразу двух зайцев. Во-первых, файл с некорректно заданным сектором уж точно не будет нормально обрабатываться ассоциированным с ним приложением (что и не удивительно, ведь действительное начало файла теперь окажется глубоко в его середине), во-вторых, алгоритм определения оригинальных длин по разнице соседних стартовых секторов даст глубоко неверный результат и восстановленный файл окажется обрезанным.

Защитный механизм, "знающий", на сколько секторов сдвинуто действительное смещение файла относительно его начала, должен либо переместить файловый указатель вызовом функции `SetFilePointer`, либо "проглотить" "мусорные данные" с помощью функции `ReadFile`. Оба способа практически равнозначны, и каждый из них имеет свои сильные и слабые стороны. Функция `SetFilePointer` работает значительно быстрее, но слишком заметна (для хакеров), напротив, с вызовом функции `ReadFile` еще предстоит разобраться, какие данные он читает — значимые или нет.

Рассмотрим, как выглядит процесс взлома на практике. Поскольку писать полноценный MP3-плеер мне было откровенного лениво (да и места он бы занял немеряно), то вся обработка данных сводится к выводу оригинального содержимого файла на экран (листинг 8.5). Перед первым запуском программы, стартовый сектор защищенного файла должен быть уменьшен на величину `_NSEC_`, а размер увеличен по меньшей мере на `2048*_NSEC_` байт, верхнего же ограничения на максимальную длину нет (все 32 бита поля длины — ваши).

Листинг 8.5. [crackme.27AF7A2Dh] Демонстрация обработки файлов с искаженными атрибутами стартового сектора и длины

```
/*---------------------------------------------------------------------------
 *
 *                        crack me 27AF7A2D
 *                        =================
 *
 *  демонстрационный пример обработки файлов с умышленно уменьшенным номером
 * стартового сектора и увеличенной длиной; позиционирование файлового
 * указателя осуществляется вызовом функции fseek, потому этот crackme
```

```
 * очень легко взломать (см. также ..... - как более стойкий пример
 * реализации той же защиты)
 *
 * Build 0x001 @ 02.07.2003
-----------------------------------------------------------------------------*/
#include <stdio.h>

// настройки программы
// ===================
// имя открываемого файла (если защита находится на CD, то полный путь к файлу,
// очевидно, указывать не обязательно)
#define _FN_                    "M:\\Depeche Mode\\01 - Personal Jesus.mp3"

// количество секторов, на которые смещено начало файла
#define _NSEC_                  4

// оригинальный размер файла
#define _FSIZ_                  3591523

// размер пользовательской части сектора
#define SECTOR_SIZE             2048

// ширина экрана в символах (нужна для вывода дампа)
#define _SCREEN_LEN_       80

// размер обрабатываемого блока
#define BLOCK_SIZE         0x666

// поиск минимума двух чисел
#define _MIN(a,b)  ((a<b)?a:b)

// ВЫВОД HEX-ДАМПА НА ЭКРАН
//-------------------------
// src - указатель на выводимые данные
// n   - кол-во выводимых на экран байт
print_hex(unsigned char *src, int n)
{
     int a; static p = 1;
     for (a=1; a <= n; a++)
           printf("%02x%s",src[a-1],(p++%(_SCREEN_LEN_/3-1))?" ":"\n");
}
```

```
main()
{
     int  a;
     FILE *f;
     long p = _FSIZ_;
     char buf[BLOCK_SIZE];

     // TITLE
     fprintf(stderr, "crackme 27af7a2d by Kris Kaspersky\n");

     // пытаемся открыть файл
     if ((f = fopen(_FN_, "rb")) == 0)
     {
           fprintf(stderr, "-ERR: can not open %s\n",_FN_); return -1;
     }

     // пропускам _NSEC_ лишних секторов, находящихся в начале файле
     fseek(f, _NSEC_*SECTOR_SIZE, SEEK_SET);

     // читаем файл блоками, тщательно следя за тем, чтобы не вылететь
     // за пределы его оригинального размера
     while(p)
     {
           // внимание! для обработки файлов с искаженным размером категорически
           // не рекомендуется использовать функцию fgetc, поскольку в большин-
           // стве своих реализаций она обрабатывает файл не байтами, но блоками
           // заранее неизвестного размера. То есть она осуществляет прозрачную
           // буферизацию ввода, опираясь на установленный размер файла как на
           // эталонный. Если же установленный размер файла искажен, то нет
           // никаких гарантий, что функция fgetc не залезет за пределы диска
           // со всеми отсюда вытекающими последствиями (особенно это вероятно,
           // если обрабатываемый файл - последний на лазерном диске), так что
           // используйте fread, а еще лучше ReadFile, который уж точно не
           // полезет в пекло поперед батьки
           fread(buf, 1, a = _MIN(p,BLOCK_SIZE), f);     // читаем очередной блок

           print_hex(buf,a); p-= a                       // выводим его на экран
     }
}
```

Обнаружив, что основные атрибуты файла искажены (то есть файл попросту не читается), хакер, очевидно, захочет определить смещение первого действи-

тельного байта и оригинальный размер файла. "Винни поплевал на лапки для храбрости и набрал заветную команду soft-ice" (почти по Щербакову) (листинг 8.6).

Листинг 8.6. Протокол работы с Soft-Ice

```
:bpx CreateFileA              ; ставим точку останова на CreateFileA
:x                            ; выходим из айса
…
Break due to BPX KERNEL32!CreateFileA (ET=3.37 seconds)
; отладчик всплывает, значит, CreateFileA кем-то только что была вызвана
; но вот кем? пытаемся определить это по имени открываемого файла

:d esp->4                     ; смотрим первый слева аргумент, передаваемый функции
0010:0040706C 4D 3A 5C 44 65 70 65 63-68 65 20 4D 6F 64 65 5C  M:\Depeche Mode\
0010:0040707C 30 31 20 2D 20 50 65 72-73 6F 6E 61 6C 20 4A 65  01 - Personal Je
0010:0040708C 73 75 73 2E 6D 70 33 00-4D 3A 5C 44 65 70 65 63  sus.mp3.M:\Depec
; ага! это как раз то что нам нужно!

:p ret                        ; выходим из функции
:? Eax                        ; подсматриваем значение дескриптора открытого файла
00000030  0000000048  "0"     ; дескриптор равен 0x30 (или 48 в десятичной нотации)

:bpx SetFilePointer   if (esp->4 == 0x30);
:bpx ReadFile         if (esp->4 == 0x30)
; устанавливаем точки останова на основные файловые функции SetFilePointer и
; ReadFile, заставляя отладчик всплывать тогда и только тогда, когда им передается
; "наш" дескриптор! (специальное замечание для разработчиков защиты: господа,
; не позволяйте же, право, себя так легко обмануть! открывайте файл несколько
; раз подряд и попеременно работайте с ним через различные дескрипторы, -
; это сильно затрудняет анализ)

:x                            ; выходим из отладчика
…
Break due to BPX KERNEL32!SetFilePointer  IF ((ESP->4)==0x30) (ET=76.19 microse-
conds)
; это сработала точка останова на SetFilePointer, теперь нам необходимо подсмотреть
; значение offset, на которое смещается указатель, и origin - чтобы определить,
; относительно какой части файла осуществляется отсчет

:? esp->8                     ; смотрим второй слева аргумент функции
00002000  0000008192  "  "    ;     указатель смещается на 0x2000 байт относительно…
```

```
:? esp->0C                    ; смотрим третий слева аргумент функции
00000000  0000000000  "  "    ;     …относительно начала файла (SEEK_SET)

:p ret                        ; выходим из отладчика
```

…

; больше функция SetFilePointer не вызывается, но зато наблюдается многократные
; вызовы функции ReadFile. Для анализа защитного кода мы не будем давать P RET
; (как это рекомендуют делать некоторые хакерские руководства). Ведь ReadFile
; скорее всего вызывается не напрямую, а из библиотечной функции-обертки,
; анализ которой нам ничего не даст. Лучше посмотрим стек вызовов…

…

```
:stack                        ; смотрим стек вызовов
12F8C8  401E1C        KERNEL32!ReadFile
12F8F8  4010E5        crackme!.text+0E1C
12FFC0  77E87903      crackme!.text+00E5
12FFF0  0             KERNEL32!SetUnhandledExceptionFilter+005C
```

; адрес 55E87903h, очевидно, принадлежит недрам операционной системы, и потому нам
; не интересен, адрес 401E1Ch (адрес возврата из ReadFile) нам также неинтересен,
; поскольку, как мы уже и говорили, скорее всего принадлежит библиотечной функции-
; обертке, а вот на адрес 4010E5h имеет смысл взглянуть:

```
:u 4010E5
001B:00401072   MOV    EDI, 36CD63        ; EDI := 36CD63
001B:004010C8   CMP    EDI,00000666       ; \                          ← (1)
001B:004010CE   MOV    ESI,EDI            ;  +- ESI := _min(0x666, EDI)
001B:004010D0   JL     004010D7           ;  +
001B:004010D2   MOV    ESI,00000666       ; /
001B:004010D7   PUSH   EBX                ; …
001B:004010D8   PUSH   ESI                ; кол-во читаемых элементов
001B:004010D9   LEA    EAX,[ESP+14]       ; получаем указатель на буфер
001B:004010DD   PUSH   01                 ; размер одного элемента
001B:004010DF   PUSH   EAX                ; передаем указать на буфер
001B:004010E0   CALL   00401141           ; эта функция вызывает ReadFile
001B:004010E5   LEA    ECX,[ESP+1C]       ; получаем указатель на буфер
001B:004010E9   PUSH   ESI                ; …
001B:004010EA   PUSH   ECX                ; …
001B:004010EB   CALL   00401000           ; обрабатываем прочитанные данные
001B:004010F0   ADD    ESP,18             ; вычищаем ненужные аргументы из стека
001B:004010F3   SUB    EDI,ESI            ; EDI := EDI - _min(0x666, EDI)
```

```
001B:004010F5   JNZ    004010C8     ; мотаем цикл, пока есть что обрабатывать (1) →
001B:004010F7   POP    ESI               ; …
; изучение окрестностей адреса 4010E5 позволяет за считанные минуты восстановить
; алгоритм обработки файла. файл читается кусками по 0x666 байт до тех пор, пока
; этих байт не наберется ровно 0x36CD63 (или 3.591.523 в десятичной нотации)
```

Таким образом, после открытия файла его указатель смещается вперед на `0x2000` байт (4 сектора), а затем из файла считывается 3 591 523 байт данных, после чего его обработка прекращается. Следовательно, защищенный файл может быть восстановлен так.

Попробуйте его проиграть каким-нибудь MP3-плеером. Если все было сделано правильно, то вы окунетесь в ритмичные звуки Depeche Mode, не омраченные более никакими защитами! Под такую музыку очень хорошо заниматься усилением защитных механизмов, а усиливать здесь есть чего!

"Хитрая" обработка защищенных файлов подразумевает использование как минимум трех дескрипторов для каждого файла: два из них обрабатывают действительно полезные данные, а третий "замысловато пляшет" по файлу, читая бессмысленный "мусор". Этот "мусор" передается громоздкой и жутко запутанной процедуре, выполняющей сложные, но реально никак не используемые вычисления. "Скормив" такой процедуре первые `_NSEC_` секторов защищенного файла, мы создадим обманчивую видимость, что обработка файла начинается с его начала (ну или почти начала, разработчику защиты ничего не стоит переместить указатель на любую понравившуюся ему позицию).

Реально используемые дескрипторы должны открываться после возращения "подсадного", т. к. большинство хакеров отслеживают лишь первый вызов функции `CreateFileA`, открывающей заданный файл, и игнорируют все остальные (многим просто не приходит с голову, что один и тот же файл может открываться дважды).

Позиционирование на первый значимый байт лучше всего осуществлять не функцией `SetFilePointer`, а путем чтения "мусорных" данных с последующей имитацией их обработки. В грамотно сконструированной защите определить, где кончается мусор и начинаются действительно значимые данные, очень и очень сложно. Однако и запрограммировать такую защиту нелегко (а ведь ее еще и отлаживать придется!), поэтому для простоты можно ограничиться тем, что начало значимых данных совпадает с первым байтом очередного читаемого блока.

Оригинальную длину файла хранить в виде константы крайне нежелательно, т. к. при анализе программы все константы сразу же бросаются в глаза и нужное значение быстро находится даже тупым перебором (большинство программ содержат не так уж много констант, соизмеримых по величине с длинами обрабатываемых файлов). Храните не длину файла, а длину его "хвоста", т. е. остаток, полученный от деления оригинальной длины на размер

обрабатываемых блоков. Естественно, размер и количество блоков также придется где-то хранить, но проанализировать взаимосвязь трех констант значительно труднее одной!

Учитывая все ранее сказанное, мы сможем значительно усилить нашу защиту. Один из вариантов ее реализации может выглядеть, например, так, как показано в листинге 8.7.

Листинг 8.7. [crackme.CEE99D84h.c] Программная реализация защитного механизма, основанного на искаженном оглавлении диска

```
// размер "хвоста" последнего блока
#define TAIL_SIZE            (_FSIZ_ % BLOCK_SIZE)

// кол-во целых блоков
#define N_BLOCKS             (_FSIZ_ / BLOCK_SIZE /2)

// ХОЛОСТАЯ ОБРАБОТКА ДАННЫХ
// -------------------------
// желательно сделать эту функцию как можно более сложной и запутанной,
// чтобы факт ее холостой работы был не так очевиден
threshing(unsigned char *src, int n)
{
      int a, sum=0;for (a = 0; a< n; a++) sum += src[a]; return sum;
}

main()
{
      int  a = 0;
      long p = _FSIZ_;
      FILE *f_even, *f_uneven, *f_threshing;
      char buf[BLOCK_SIZE + (_NSEC_*SECTOR_SIZE)];

      // TITLE
      fprintf(stderr, "crackme 27af7a2d by Kris Kaspersky\n");

      // пытаемся открыть файл
      // f_threshing лучше всего открывать первым, т.к. первый же встретившийся
      // хакеру вызов CreateFileA должен давать "подсадной" дескриптор
      if (      ((f_threshing   = fopen(_FN_, "rb")) == 0) ||
                ((f_even        = fopen(_FN_, "rb")) == 0) ||
                ((f_uneven      = fopen(_FN_, "rb")) == 0))
                { fprintf(stderr, "-ERR: can not open %s\n",_FN_); return -1;}
```

```
    // устанавливаем f_even
    fread(buf, 1, _NSEC_*SECTOR_SIZE, f_even);

    // имитируем пропуск NSEC*SECOR_SIZE/2 байт (вдруг хакер на это купится?)
    // на самом же деле, все NSEC*SECTOR_SIZE первый байт файла идут
    // в мусорное ведро
    threshing(buf + _NSEC_*SECTOR_SIZE/2,_NSEC_*SECTOR_SIZE/2);

    // устанавливаем f_uneven
    fread(buf, 1, _NSEC_*SECTOR_SIZE+BLOCK_SIZE, f_uneven);

    // имитируем пропуск NSEC*SECOR_SIZE/3 байт
    threshing(buf + _NSEC_*SECTOR_SIZE/3,  2*_NSEC_*SECTOR_SIZE/3+BLOCK_SIZE);

    // устанавливаем threshing, пуская хакера по ложному следу
    fseek(f_threshing,_NSEC_*SECTOR_SIZE/4,SEEK_SET);

    // читаем файл блоками, тщательно следя за тем, чтобы не вылететь
    // за пределы его оригинального размера

    for (a=0; a < N_BLOCKS; a++)
    {
        // читаем данные вхолостую
        fread(buf, 1, BLOCK_SIZE, f_threshing);     threshing(buf,BLOCK_SIZE);

        // читаем четный действительный блок
        fread(buf, 1, BLOCK_SIZE, f_even);          print_hex(buf,BLOCK_SIZE);

        // пропуск нечетного блока для дескрпитора f_even
        fread(buf, 1, BLOCK_SIZE, f_even);          threshing(buf,BLOCK_SIZE);

        // читаем нечетный действительный блок
        fread(buf, 1, BLOCK_SIZE, f_uneven);        print_hex(buf,BLOCK_SIZE);

        // пропуск уже четного блока для дескрпитора f_uneneven
        fread(buf, 1, BLOCK_SIZE, f_uneven);        threshing(buf,BLOCK_SIZE);
    }
    // дочитываем хвост
    fread(buf, 1, TAIL_SIZE, f_even); print_hex(buf, TAIL_SIZE);
}
```

Попробуйте взломать эту защиту. Что, не получается? Попытка отследить вызовы файловых функций `SetFilePoiner` и `ReadFile` ничего не дает, т. к. данные считываются глубоко нелинейным образом и способов быстрого отделения "зерен" от "плевел" здесь не существует. Такие защиты вообще не ломаются в отладчике, — тут требуется помощь дизассемблера, но и с дизассемблером на скорый успех рассчитывать не приходится. Сложность и запутанность алгоритма обработки данных значительно усложняет анализ программы и на определение действительных границ файла даже у профессионала может уйти несколько часов (а в некоторых случаях и *дней!*). По соображениям экономии места дизассемблерные листинги и описание процесса взлома здесь не приводятся, поскольку в них нет ничего интересного — просто тупая рутина и все. Единственная зацепка — функция `Threshing`, имитирующая обработку данных. Как только хакер поймет, что результаты ее работы никак не используются в программе, он тут же продвинется далеко вперед. Контрольные точки останова на чтение/записи памяти позволяют быстро и элегантно определить: происходит ли обращение к заданным ячейкам или нет. Короче говоря, нет ничего тайного, что при помощи отладчика Soft-Ice и дизассемблера IDA Pro не стало бы явным...

Шифровка файлов

Для предотвращения пофайлового копирования в идеале следовало бы использовать *собственные нестандартные форматы данных*, просмотр/прослушивание которых в обход программы-оболочки был бы невозможным. Однако разработка "своего" формата требует значительных вложений, которые все равно не будут оправданы, т. к. прежде чем программа окупит себя, хакеры успеют "отвязать" ее от диска, взломав секторную защиту и тем самым получив возможность массового тиражирования носителя.

Поэтому разработчики защиты предпочитают отталкиваться от уже существующих форматов (например, того же MP3), просто *зашифровывая файлы перед записью на мастер-диск*, а при проигрывании — расшифровывая их содержимое "на лету". Минус такого подхода в том, что защиты подобного типа очень легко взломать, — достаточно просто установить точку останова на функцию `CreateFile` и, дождавшись открытия нужного нам файла, подсмотреть значение регистра EAX на выходе из функции — оно-то и будет представлять собой дескриптор открытого файла. Теперь остается лишь поставить точки останова на функции `SetFilePointer`/`ReadFile`, приказав отладчику "всплывать" только в случае передачи им "нашего" дескриптора. Точка останова, установленная на область памяти, содержащую прочитанные с диска данные, выведет хакера непосредственно на процедуру расшифровки, проанализировав алгоритм которой, хакер сможет написать свой собственный расшифровщик!

А если алгоритм шифровки представляет собой тривиальную операцию `XOR` (как в подавляющем большинстве случаев и наблюдается), взломать содержимое диска можно еще быстрее! Ведь практически все стандартные форматы файлов содержат в себе некоторое количество более или менее предсказуемой информации и потому могут быть расшифрованы атакой по открытому тексту. И AVI, и MP2/MP3, и WMA, и ASF-файлы содержат в себе длинные цепочки подряд идущих нулей (и/или символов с кодом `FF`), а потому ключ шифрования обнаруживается тривиальным просмотром содержимого защищенного файла в любом HEX-редакторе.

Рассмотрим следующий пример. Пусть у нас имеется мультимедийный диск "Depeche Mode The best", содержимое одного из файлов которого выглядит так, как это показано в листинге 8.8.

Листинг 8.8. Hex-дамп заголовка исследуемого файла

```
K:\sex\1\03 - Strangerlove.dat                    DOS     3472405
00000000:  9D 9A F0 39 62 61 60 3A | 65 A4 8B F0 52 C1 01 98   ЭЪЁ9ba`:едЛЁR┴☺Ш
00000010:  BA DB 03 5A 54 27 A6 4C | 43 ED 46 1D 8B 21 ED 9A   ║╙♥ZT'жLCэF↔Л!эЪ
00000020:  D3 C7 7B 58 4B A6 78 5D | F6 FA F0 A9 55 63 66 A8   ╙╟{XKжx]Ў·ЁйUcfи
00000030:  7E 6A 5A 79 61 68 E8 7B | 69 47 F9 7B 60 22 E3 88   ~jZyahш{iG·{`"уИ
00000040:  61 E2 67 98 E0 E2 2D ED | 13 AD E3 38 C5 A5 71 FB   атgШрт-э‼ну8┼eq√
00000050:  1A 01 C0 B6 85 77 5A 49 | 46 4F 93 7B BF 30 A5 9D   →☺╢╢EwZIFOУ{┐0eЭ
```

По внешнему виду это не похоже ни на MP3 (MP3-файлы начинаются с сигнатуры `FF FB`, впрочем, не всегда расположенной в самом начале), ни на WAV (WAV-файлы начинаются с сигнатуры `RIFF`), ни на RealAudio (RealAudio-файлы начинаются с `.RMF`), но ведь каким-то же образом они все-таки воспроизводятся! И вряд ли это собственный формат разработчиков мультимедийного диска. Скорее всего, файл просто зашифрован. А раз так, то его можно попробовать расшифровать!

Прокручиваем экран HEX-редактора вниз, пока не встречаем следующую регулярную последовательность (листинг 8.9).

Листинг 8.9. Регулярная последовательность, обнаруженная внутри исследуемого файла

```
K:\sex\1\03 - Strangerlove.dat                                DOS     3472405
000001E0:  C3 5A AF F8 70 4A D8 83 | 5D 9E 9D 86 9D 9E 9D 86   ├Zп°pJ╪Г]ЮЭЖЭЮЭЖ
000001F0:  9D 9E 9D 86 9D 9E 9D 86 | 9D 9E 9D 86 9D 9E 9D 86   ЭЮЭЖЭЮЭЖЭЮЭЖЭЮЭЖ
00000200:  9D 9E 9D 86 9D 9E 9D 86 | 9D 9E 9D 86 9D 9E 9D 86   ЭЮЭЖЭЮЭЖЭЮЭЖЭЮЭЖ
00000210:  9D 9E 9D 86 AA C2 62 79 | 62 B4 C0 6A 9D 9E 9D 86   ЭЮЭЖк┬byb┤└jЭЮЭЖ
00000220:  9D 9E 9D 86 9D 9E 9D 86 | 9D 9E 9D 86 9D 9E 9D 86   ЭЮЭЖЭЮЭЖЭЮЭЖЭЮЭЖ
```

```
00000230:  9D 9E 9D 86 9D 9E 9D 86 | 9D 9E 9D 86 9D 9E 9D 86   ЭЮЭЖЭЮЭЖЭЮЭЖЭЮЭЖ
00000240:  9D 9E 9D 86 9D 9E 9D 86 | 70 B8 46 0B 3B 61 63 30   ЭЮЭЖЭЮЭЖр¬F♂;ac0
```

Весьма вероятно, что в оригинальном файле здесь располагалась цепочка байт с идентичными значениями, например, последовательность нулей или `FF`, после выполнения операции `XOR` с неким четырехбайтовым ключом.

Поскольку `XOR` — симметричная операция, то `((A XOR B) XOR A) == B`, то есть повторная шифровка файла его исходным содержимым дает нам ключ. Предположим, что на этом месте были нули, то тогда искомый ключ шифрования будет равен "…`9E 9D 86 9D`…". Точки по обе стороны от ключа означают, что мы еще не готовы выделить начало и конец регулярной последовательности. В самом деле, это может быть как `9E 9D 86 9D`, так и `9D 86 9E 9D` и даже `86 9D 9E 9D` или вообще `9D 9E 9D 86`. Однако вместо того, чтобы тупо перебирать все четыре варианта, давайте обратим внимание вот на что. Длина регулярной последовательности равна четырем, так? Следовательно, первый байт каждого "периода" должен лежать по смещению, кратному четырем. Отсюда искомая последовательность имеет вид `9D 9E 9D 86`, а все остальные варианты неверны, т. к. лежат по неподходящим адресам. Поскольку начальные адреса HEX-строк, выводимых редактором, выровнены по границе `0x10` байт (а `0x10` кратно 4), то первый байт ключа должен совпадать с начальным адресом любой из строк.

Теперь допустим, что в данном месте оригинального файла находилась цепочка нулей. Тогда ключ шифровки должен выглядеть так: `9D 9E 9D 86` (т. к. `(A XOR 0) == A`). Запускаем HEX-редактор HIEW, нажатием <Enter> переводим его в шестнадцатеричный режим, давим <F3> для активизации режима редактирования, далее нажимаем <F8> и в появившееся диалоговое окно **Enter XOR mask** вводим следующую HEX-последовательность и, "уронив кирпич" на <F8>, идем пить чай, так как HIEW расшифровывает файл ну очень долго и потому ничего не остается, как садиться за свой любимый компилятор и писать собственную программу расшифровки. Приведенному листингу (листинг 8.10) до образца программистского искусства, конечно, далеко, но как рабочий вариант, сделанный на скорую руку, он вполне сойдет.

Листинг 8.10. Демонстрационный вариант простейшего шифровщика

```
/*---------------------------------------------------------------------------
 *
 *              XORит СОДЕРЖИМОЕ ФАЙЛА ПО ПРОИЗВОЛЬНОЙ МАСКЕ
 *              ============================================
 *
 * Build 0x001 @ 09.07.2003
---------------------------------------------------------------------------*/
```

```
#include <stdio.h>

#define MAX_LEN          666                    // макс. длина маски
#define MAX_BUF_SIZE     (100*1024)             // размер буфера чтения
#define FDECODE          "decrypt.dat"          // имя расшифрованного файла

main(int argc, char **argv)
{
      long       a, b;
      long       key_len;
      FILE       *fin, *fout;
      long       buf_size, real_size;
      unsigned char key[MAX_LEN];
      unsigned char buf[MAX_BUF_SIZE];

      if (argc<3)            // HELP по ключам командной строки
      {
            fprintf(stderr,"USAGE: DeXOR.exe file_name AA BB CC DD EE...\n");
            return 0;
      }

      // определение длины ключа и установка размера буфера кратной длине ключа
      key_len = argc - 2;      buf_size = MAX_BUF_SIZE - (MAX_BUF_SIZE % key_len);

      // извлекаем ключи из командной строки в массив key
      for(a = 0; a <  key_len; a++)
      {
            // преобразование из HEX-ASCII в long
            b = strtol(argv[a+2],&" ",16);

            if (b > 0xFF)            // проверка на предельно допустимое значение
                  {fprintf(stderr, "-ERR: val %x  not a byte\x7\n",b); return -1;}

            key[a] = b;            // заносим значение очередного байта ключа
      }

      printf("build a key:"); // выводим ключ на экран (для контроля)
      for(a=0;a<key_len;a++) printf("%02X",key[a]); printf("\n");

      // открываем файлы для чтения/записи
      fin = fopen(argv[1],"rb"); fout=fopen(FDECODE,"wb");
      if ((fin==0) || (fout==0))
```

```
            { fprintf(stderr, "-ERR: file open error\x7\n"); return -1; }

      // основной цикл обработки
      while(real_size=fread(buf,1, buf_size, fin))
      {
            // цикл по буферу
            for (a=0; a<real_size; a+=key_len)
            {
                  // цикл по ключу
                  for(b=0; b < key_len; b++)
                        buf[a+b] ^= key[b];
            }

            // скидываем зашифрованный (расшифрованный) буфер на диск
            if (0 == fwrite(buf, 1, real_size, fout))
            {
                  fprintf(stderr,"-ERR: file write error\x7\n");
                  return -1;
            }
      }
      // сваливаем
}
```

Компилируем и запускаем: `DeXOR.c "03 - Strangerlove.dat" 9D 9E 9D 86` и ничего не получается! Расшифрованный файл все равно не похож ни на MP3, ни на другие форматы. Что ж, значит, это была не цепочка нулей, а что-то другое, например, последовательность символов `FF`. В плане проверки нашей гипотезы выполним операцию `XOR` над регулярной последовательностью `9D9E9D86h` и числом `FFFFFFh`, и, если нам повезет, в результате этой операции мы получим оригинальный ключ. Для осуществления задуманного нам вновь понадобится HEX-редактор HIEW или штатный калькулятор Windows. Запустим его и, кликнув мышью по меню **Вид**, найдем пункт **Инженерный**. Теперь выберем положение **Hex** (шестнадцатеричная система) для переключателя, ответственного за выбор системы исчисления, или же попросту нажмем <F5>. Используя мышь и/или клавиатуру, введем 9D9E9D86, затем нажмем кнопку **Xor** и введем FFFFFFFF. Подтвердим серьезность своих намерений нажатием <Enter>. Калькулятор выдаст следующий результат: 62616279. Это и есть искомый ключ, вводим его в программу DeXOR, разделяя байты пробелами, и после переименования Strangerlove.dat в Strangerlove.mp3 он соглашается воспроизводиться любым MP3-плеером. Аналогичным путем расшифровываются и все остальные файлы, находящиеся на защищенном диске (ключ шифрования у них разный, но методика его поиска общая).

Какие из этого следуют выводы? Если уж взялись шифровать файлы с предсказуемым содержимым, то выбирайте длину ключа шифрования так, чтобы она была сравнима с длиной этих предсказуемых последовательностей, а желательно — в несколько раз превышала ее. Или же меняйте операции `XOR` на более продвинутые алгоритмы шифрования (если, конечно, вам не лень их реализовывать).

На самом деле, задача получения длинного непериодического ключа элегантно решается с помощью *генератора случайных чисел* (или, точнее, *псевдослучайных* чисел). Как, наверное, помните, псевдослучайная последовательность, генерируемая библиотечной функцией `rand()`, постоянная при каждом запуске программы, поэтому представляет собой отличный и в то же время категорически неочевидный ключ! Собственно говоря, приведенная далее программа (листинг 8.11) именно так и делает.

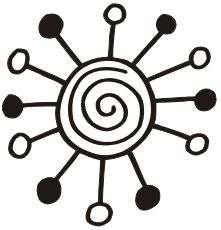
Листинг 8.11. Использование rand() для хранения ключа шифровки

```
/*-----------------------------------------------------------------------------
 *
 *            НЕЯВНАЯ ГЕНЕРАЦИЯ КЛЮЧА РАСШИФРОВКИ С ПОМОЩЬЮ RAND()
 *            ====================================================
 *
 *            шифрует/расшифровывает файлы, используя функцию rand() для генерации
 * ключа; поскольку функция rand() при каждом запуске дает одну и ту же
 * последовательность, устанавливаемую srand(), мы получаем оччччень длинный
 * и не периодичный ключ шифрования, "ослепляющий" атаку по открытому тексту.
 * К тому же, если чуть-чуть изменить код rand(), то IDA не сможет ее
 * распознать (правда, это не сильно усложнит взлом, т.к. реализация функции
 * rand() в подавляющем большинстве случаев до смешного проста).
 *            дополнительные уровни защиты обеспечиваются запутываем алгоритма
 * обработки данных (можно, например, создать целую серию расшифровщиков,
 * из которых полезные данные будет выдавать только  один, а  остальные
 * станут возвращать бессмысленный мусор)
 *
 *            NOTE: чтобы зашифровать оригинальный файл, запустите программу
 * с ключом "-crypt", это достаточно сделать один раз (файл, поставляемый
 * на CD-диске, прилагаемом к книге, уже зашифрован, и попытка его повторной
 * зашифровки приведет к прямо противоположному результату - файл будет
 * записан на диск в расшифрованном виде)
 *
 * Build 0x001 09.07.2003
-----------------------------------------------------------------------------*/
```

```
#include <stdio.h>
#include <math.h>

#define FNAME        "file.dat"          // имя файла для (за|рас)шифровки
#define MAX_SIZE     (100*1024)          // максимально возможный размер файла

#define SEED          0x666              // установка последовательности rand()
                                         // это число может быть любым, главное,
                                         // чтобы оно было!

//--[crypt]-------------------------------------------------------------------
// fname       - имя файла
// buf         - указатель на буфер, куда помещать расшифрованные данные
// buf_size    - размер буфера
// need_store  - требуется ли записывать (за|рас)шифрованный файл на диск
//             :0 - не записывать, != 0 - записывать
//-----------------------------------------------------------------------------
crypt(char *fname, char *buf, int buf_size, int need_store)
{
      FILE       *f;
      long      a, b;

      // не забудьте явно проинициализировать генератор случайных чисел, иначе
      // если другие ветки программы так же используют rand(), результат
      // расшифровки получается "плавающим"
      srand(SEED);

      // открытие расшифровываемого файла
      f=fopen(fname, "rb"); if (f==0) return -1;

      // загрузить данные в буфер
      a = fread(buf, 1, buf_size, f); if (!a || (a == buf_size)) return -1;

      // (за|рас)шифровать содержимое буфера ключом, налету генерируемым
      // функцией rand()
      for (b = 0; b < a; b++) buf[b] ^= (rand() % 255); fclose(f);

      // отладочный шлюз для автоматической шифровки файла
      if (need_store)
      {
            f=fopen(fname, "wb"); if (f==0)  return -1;
            fwrite(buf, 1, a, f); fclose(f); return -1;
```

```
        }
        return a;
}

main(int argc, char** argv)
{
        long  a,x;
        long  need_store  = 0;
        unsigned char buf[MAX_SIZE];

        // TITLE
        fprintf(srderr,"crackme 765b98ec by Kris Kaspersky\n");

        // если есть отладочный ключ -crypt зашифровать
        if ((argc > 1) && !strcmp(argv[1],"-crypt")) need_store++;

        // загрузить файл FNAME, расшифровать и вывести его содержимое на экран
        if ((x=Crypt(FNAME, buf, MAX_SIZE, need_store))!=-1)
              for (a=0;a<x;a++) printf("%c",buf[a]);
}
```

Глава 9



Защиты, основанные на привязке к носителю

Считается, что правильно снятая цифровая копия целиком идентична своему оригиналу, благодаря чему, собственно, разница между пиратскими и фирменными дисками заключается лишь в их стоимости и потребителю не приходится ломать голову над вопросом, чью продукцию покупать. На самом же деле, двух абсолютно идентичных лазерных дисков не существует, — каждый из них обладает рядом уникальных характеристик, которые отличают его от других. Эти уникальные характеристики (далее по тексту — *метки*) могут использоваться защитными механизмами для идентификации оригинального носителя и разоблачения несанкционированных копий.

Профессиональная этика обязывает использовать для идентификации лишь те метки, которые отвечают следующим, достаточно жестким, требованиям:

- метка должна безошибочно распознаваться любыми приводами;
- метка не должна воспроизводиться какими бы то ни было копировщиками;
- устойчивость метки к внешним воздействиям (царапинам, старению диска) должна быть не ниже, чем у остальных данных, записанных штатным способом.

К сожалению, защиты, удовлетворяющей всем вышеперечисленным критериям, до сих пор не существует, и достаточно часто приходится сталкиваться с тем, что легально купленная программа ошибочно считает себя пиратской копией и потому отказывается запускаться. Какие же характеристики носителя разработчики защит выбирают в качестве ключевых? Анализ существующих защит показывает, что это: во-первых, физические дефекты носителя (как естественные, так и умышленно нанесенные); во-вторых, временные характеристики чтения групп секторов; в-третьих, показатель стабильности чтения и, в-четвертых, та информация, которую сообщает сам носитель (ATIP в частности).

Нанесение меток vs[1]. Динамическая привязка

Умышленное нанесение уникальных меток на носитель само по себе не сложно и широко распространено. Кто победнее — царапает диск циркулем, ну а кто побогаче — уродует его лазером. Более изощренные защитники программ прибегают к питам нестандартной формы или сложным образом манипулируют с плотностью спиральной дорожки и/или ее узором. Однако все эти способы не лишены недостатков. Во-первых, они требуют применения специального оборудования. Во-вторых, *уникальные характеристики носителя* потому и называется уникальными, что не могут быть "состряпаны" по заказу и формируются в процессе рождения самого носителя. То есть до того момента, пока диск не будет вынут из "печки", защитный механизм еще не знает тех характеристик носителя, к которым он привязывается. А после завершения процесса "выпечки" сообщать эти самые характеристики защите уже поздно, т. к. записывать их уже некуда, — дозапись на носители CD-ROM, увы, не возможна. Теоретически, можно поместить закодированные метки на дискету, прилагающуюся к защищенному диску CD-ROM, но ведь это маразм. В-четвертых, те "уникальные" характеристики, которые наносятся на мастер-копию CD-ROM, оказываются бессильными против тех "нечистых на руку" заводов по штамповке, что часть отпечатанного тиража "сливают" в свой карман.

Поэтому лучше вообще не наносить на диск никаких меток, а использовать те, что уже есть, определяя их уникальность "на лету". Как это можно сделать? Да очень просто! Защитный механизм измеряет ту характеристику диска, что подвержена наибольшему разбросу от одного экземпляра носителя к другому (как правило, это временная характеристика чтения). Затем по специально заданному алгоритму защитный механизм преобразует ее в некоторый код, который и сообщается владельцу этого диска. Владелец передает этот код разработку программы и получает (не бесплатно, конечно) регистрационный номер, который представляет собой некоторую производную от этого кода (для простоты будет считать, что регистрационный номер равен коду характеристики диска, умноженному на `0x666`). После ввода регистрационного номера, защитный механизм проделывает с ним обратную операцию, а затем сравнивает полученный результат с кодом характеристики диска (как вариант: защита может самостоятельно вычислить регистрационный номер по коду характеристики и сравнить его с регистрационным номером, введенным пользователем). Если они совпадают, то все OK, в противном же случае пользователь посылается туда, куда Макар телят не гонял.

[1] vs. — versus (лат.) — против. — *Ред.*

Достоинство этого механизма в том, что для создания защищенного диска не обязательно иметь никакого специфического оборудования, — вполне подойдет обычный пишущий привод. Это раз. Копирование защищенного диска протекает без каких-либо осложнений, однако все дубликаты автоматически теряют свой регистрационный статус (ведь их код характеристики будет уже другой!), но могут быть зарегистрированы путем обращения к разработчику программы — это два!

Естественно, алгоритм генерации регистрационного номера должен быть выбран так, чтобы между ним и кодом характеристик не обнаружилось никакой видимой зависимости, а сама процедура его проверки всячески сопротивлялась ее исследованию дизассемблером или отладчиком. В противном случае, вашу защиту смогут легко и эффективно вскрыть.

Защиты, основанные на физических дефектах

Идея защит данного типа заключается в умышленном повреждении поверхности диска в одном или нескольких местах. При попытке чтения секторов, попавших в дефектную область, привод, поерзав некоторое время головкой, возвратит сообщение об ошибке. Большинство штатных копировщиков вообще не смогут скопировать такой диск и при встрече с первым же нечитаемым сектором аварийно завершат свою работу. Более "продвинутые" копировщики, умеющие пропускать сбойные сектора, все-таки перепишут всю читаемую информацию на физически здоровую болванку. Однако теперь на месте дефектных областей будут располагаться вполне нормальные сектора, свободно читающиеся без каких-либо сбоев (хотя и содержащие "мусор", т. к. скопировать эти сектора копировщику не удалось).

Защите остается лишь проверить подсунутый ей носитель на предмет присутствия физических дефектов в заданном месте, и, если сектора с соответствующими номерами читаются без проблем, защита делает вывод, что это не оригинал, а его несанкционированная копия.

Физические дефекты, к которым привязывается защита, как правило, представляют собой крошечные пятнышки (от одного до двух миллиметров в диаметре), выжженные на его поверхности лазером. При рассматривании диска в отраженном свете они легко обнаруживаются невооруженным глазом (рис. 9.1). На первый взгляд кажется, что, точно измерив их геометрические координаты (например, с помощью палетки[2]), нам удастся воспроизвести

[2] Палетка (от франц. palette — пластинка, планка), начерченная на прозрачной бумаге, стекле или целлулоидной пластинке сетка линий, образующих квадраты известных размеров, при помощи которых определяется площадь участков на плане или карте. — *Ред.*

идентичные повреждения на скопированном носителе. Однако на проверку оказывается, что это не так и причина нашей неудачи заключается в том, что номера секторов никак не привязаны к их физическому расположению. В зависимости от ширины спиральной дорожки, размеров области Lead-In, длины питов и лендов, в одной и той же геометрической точке диска могут находиться различные сектора! Вероятность же того, что в двух несвязанных между собой партиях болванок эти сектора совпадут, ничтожно мала! А потому, скопировать такой диск "в лоб" не получится!

Рис. 9.1. Диск с умышленно поврежденным сектором (эдакий "вулканчик", расположенный ниже середины экрана). Использование специального оборудования позволяет прожечь поверхность точно по центру спиральной дорожки (впрочем, вряд ли вы увидите все эти подробности при такой полиграфии)

Приходится хитрить и имитировать сбой программными средствами! Первое, что приходит в голову, — создать простейший эмулятор диска CD-ROM, который при попытке обращения к проверяемым защитой секторам будет возвращать ошибку чтения, вводя тем самым "глупую" защиту в заблуждение. Причем и сам эмулятор, и образ оригинального диска можно без труда "закатать" на болванку CD-R, используя "пережиг" (over-burn). Другой способ взлома заключается в реализации резидентного перехватчика, контролирующего все обращения к диску CD-ROM и возвращающего ошибку чтения при проверке определенных секторов на "дефективность".

Оба способа широко использовались еще во времена MS-DOS и дискет на 5,25", а потому методика создания эмуляторов и перехватчиков хорошо отработана. Конечно, лазерный диск — совсем не то же, что дискета, однако методика работы с ним в целом общая. Единственная проблема — для установки дискового эмулятора операционная система Windows NT требует наличия прав администратора, которые не у всех есть. В этом смысле использование перехватчика

представляется более предпочтительным, поскольку для перехвата API-функций порожденного нами процесса никаких особенных прав не нужно — просто правим таблицу импорта и все! (Как вариант — ставим на API-функции точки останова или закрепляем на них "хуки" (hooks).) О том, какие именно функции мы должны перехватывать — читайте далее в этой главе.

Кстати, лазерный диск выгодно отличается от дискеты тем, что позволяет имитировать физические дефекты поверхности на логическом уровне. Вспомните о тех корректирующих кодах, которые находятся в "хвосте" каждого сектора. Вспомнили? А теперь представьте, что произойдет, если умышленно исказить их так, чтобы даже при успешном чтении сектора возникала неустранимая ошибка контрольной суммы (т. е. такая ошибка, которая не может быть исправлена за счет избыточности). Микропрограмма, "зашитая" в привод, всесторонне проанализировав ситуацию и даже попытавшись считать "подопытный" сектор несколько раз, в конце концов перестанет ерзать головкой и возвратит сообщение об ошибке, однако не будет уточнять, физическая это ошибка или логическая. А значит, с точки зрения программного обеспечения, и физические, и логические дефекты диска будут выглядеть совершенно одинаково! Вообще-то, защита может попросить вернуть содержимое дефектного сектора в "сыром" (RAW) виде, и, если поверхность диска в действительности не содержит никаких физических повреждений, этот сектор будет успешно прочитан, а раз так, то мы имеем дело не с оригиналом, а копией. Однако не все приводы поддерживают "сырое" чтение (в просторечии называемое "сыром"), и потому попытка промышленного внедрения такой проверки вызовет нарекания большого количества легальных пользователей, вызванных несовместимостью с оборудованием, так что прибегать к ней не стоит. Разработчики защиты могут сделать "ход конем": если привод не поддерживает "сырого" чтения, то защита доверяет тем показаниям, что у нее есть, в противном случае осуществляется дополнительная проверка. Выходит, что и "волки сыты, и овцы целы!" Скопированный таким образом диск будет работать лишь в тех приводах, что не поддерживают "сырого" чтения.

Наконец, защиту можно просто загрузить в дизассемблер или отладчик, найти процедуру проверки дефектных секторов и "отломать"! Естественно, для этого потребуется знать, как вообще может осуществляться такая проверка, на что в первую очередь обращать внимание и что, собственно, нам следует искать (полный анализ защитного кода — не предлагать).

Операционные системы семейства Windows оказываются на удивление "богаты" в отношении средств работы с дисками CD-ROM на сектором уровне. Помимо очевидных функций `CreateFile` или `DeviceIoControl`, существуют такие "приблуды", как ASPI (Windows 9x/NT) и SPTI (только Windows NT), да и другие интерфейсы имеются во множестве. Под Windows 9x можно

напрямую обращаться к CDFS-драйверу через функцию `ABSOLUTE_READ`[3] прерывания `INT 2Fh` (точнее, не совсем напрямую, а через специальный переходник 16-разрядной динамической библиотеки DLL, вызывающей DPMI-функцию `Simulate Real Mode Interrupt`[4]; подробнее об этом можно прочитать в технической заметке Q137813 *"How Win32 Applications Can Read CD-ROM Sectors in Windows 95"* из Microsoft Knowledge Base, входящей в состав MSDN, прилагаемой к диску с Microsoft Visual Studio. Там же содержится исходный текст функции для работы с дисками CD-ROM на секторном уровне).

Под Windows NT чтение/запись секторов осуществляется еще проще. Достаточно открыть диск в cooked-mode[5], после чего с ним будет можно работать на логическом уровне. Образно говоря, все содержимое диска будет трактоваться как один большой файл. Детальное описание этого процесса вы найдете в технической заметке Q138434 *"How Win32-Based Applications Read CD-ROM Sectors in Windows NT"* из Microsoft Knowledge Base.

Кстати, для проверки наличия дефекта на существование, вовсе не обязательно спускаться на уровень "голых" секторов. Ничуть не с меньшей эффективностью можно воспользоваться и файловым обменом. Очевидно, что файл, содержащий по крайней мере один сбойный сектор, не сможет читаться и даст ошибку! Основное достоинство этого способа в том, что его можно реализовать на любом языке, оперируя лишь штатными средствами. Никаких API-функций и "монструозных" `IOCTL`, — вызова функций `fopen`/`fread` будет вполне достаточно!

Таким образом, возможных способов привязки к диску очень много и локализация защитного механизма в коде ломаемого приложения представляет довольно "муторную" задачу. Если точки останова, установленные на функцию `DeviceIoContorl`, не дадут никакого результата (что, впрочем, случается не так уж и часто), то вашему положению трудно позавидовать. Покупайте бочонок пива и занимайтесь с защитой до утра. Да помогут вам API-шпион и контекстный поиск. Ну, со "шпионом", положим, все ясно, согласитесь вы. Но поиск-то тут причем? А вот причем: зная номера сбойных секторов, вы можете попробовать отыскать соответствующие им константы в теле программы. Если разработчик защиты не использовал дополнительных

---

[3] `EAX := 1508h` см. Interrupt List By Ralf Braun.

[4] Кстати, это очень хорошая функция и с ее помощью Win32-приложения могут сделать все то, что разрешается делать приложениям MS-DOS (бесконтрольный доступ к оборудованию, прерывание и все такое).

[5] Открытие диска в cooked-mode осуществляется посредством функции `CreateFile`, вызываемой следующим образом: `hCD = CreateFile ("\\\\.\\X:", GENERIC_READ,FILE_SHARE_READ| FILE_SHARE_WRITE,NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL)`, где `X` — буква, обозначающая логический диск.

ухищрений, то номера контролируемых секторов записаны в программе, так, как они есть, и все, что вам остается, — поставить на соответствующую ячейку памяти точку останова и немного подождать, пока к ней кто-нибудь не обратится. Естественно, если вы исследуете программу не в отладчике, а дизассемблере, то вместо точек останова вам следует использовать перекрестные ссылки.

Если же, несмотря на все усилия, взломать защиту никак не получается, то попробуйте прибегнуть к помощи CloneCD и/или Alcohol 120%. Обе эти утилиты распознают сбойные сектора и имитируют их на логическом уровне. CloneCD — с помощью корректирующих кодов, а Alcohol 120% — с помощью корректирующих кодов и виртуального диска! Впрочем, как показывает практика, в подавляющем большинстве случав к виртуальному диску прибегать нет никакой необходимости. Защитные механизмы в своей массе слишком "доверчивы", и искажение контрольной суммы сектора легко вводит их в заблуждение.

Таким образом, стойкость защит данного типа следует считать крайне неудовлетворительной, и вам следует хорошо подумать, прежде чем оснащать такой защитой ваши собственные программы. Любой грамотный пользователь, умеющий "держать в руках" CloneCD, скопирует защищенный диск в два счета, и вы останетесь не у дел. К тому же, для создания качественных физических дефектов ("качественный физический дефект" — это звучит!) требуется труднодоступное и весьма дорогостоящее оборудование, которого у программистов, "пасущихся на вольных хлебах", попросту нет! Царапать же диск гвоздем категорически недопустимо! Это не только ненадежно, небезопасно для здоровья привода, но и не эстетично.

Если же, несмотря на все мои предупреждения, вы все-таки решили остановиться на защите данного типа, то позвольте дать вам несколько советов. Во-первых, наносить повреждения следует не со стороны поликарбонатной подложки, а со стороны защитного слоя. То есть, попросту говоря, сверху диска. Нанося диску повреждения, помните о том, что попытка нанести глубокую радиальную царапину заканчивается, как правило, очень печально. Диск теряет свою механическую прочность, и центробежные силы рано или поздно разрывают его на куски, что практически всегда приводит к гибели и сам привод CD-ROM. Лучше проколите в отражающем слое небольшую дырочку, — того будет вполне достаточно, чтобы один или даже несколько секторов перестали читаться. Осуществить такую операцию можно обычной швейной иголкой.

Теперь поговорим о том, как при нанесении физических дефектов не повредить никаких полезных данных. Царапать незаписанную область лазерного диска — бесполезно, все равно он ее не прочтет. Вот и приходится записывать на диск что-нибудь ненужное. Такое, что не жалко и повредить. Один из возможных способов действия выглядит так.

1. Взяв чистую болванку CD-R, вы "заливаете" на нее все файлы программы, за исключением того, что содержит защитный механизм привязки

к диску. Как правило, это главный исполняемый файл программы, хотя защита вполне может быть помещена и в одну из динамических библиотек или даже внедрена в файл данных, но это уже извращение.

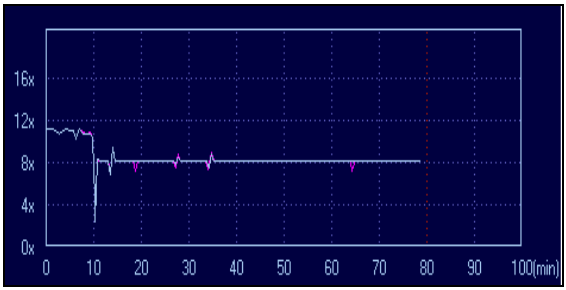
2. Нажмите кнопку выброса диска и каким-либо образом пометьте положение последней записанной дорожки (например, измерьте диаметр "прожженной" области обычной ученической линейкой).
3. Верните диск на место и запишите на него примерно 150 Мбайт всякой всячины, которая и послужит плацдармом для царапанья. Закрывать сессию все еще не надо!
4. Теперь нанесите диску одно или несколько физических повреждений, тыкая иголкой в область последнего записанного кольца.
5. Запустите любой дисковый доктор и определите позиции всех обнаруженных сбойных секторов.
6. "Прошейте" номера сбойных секторов в защищаемую программу и "залейте" защитный модуль на болванку, закрыв за собой сессии.
7. Все! Защита готова!

Защиты, основанные на временных характеристиках чтения

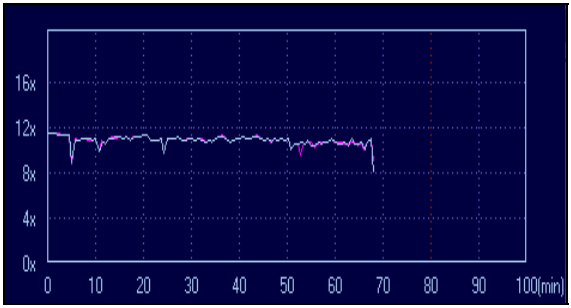
Скоростная диаграмма чтения — вероятно, наиболее легко измеряемая, но в то же время и наиболее уникальная характеристика диска, значительно варьирующаяся от одной болванки к другой. Проведем простой эксперимент: возьмем какой-нибудь компакт-диск, сделаем его копию, а затем сравним скоростную диаграмму чтения копии с оригиналом. Результат, полученный автором, иллюстрируют два рисунка, приведенные далее (в роли эталонного диска выступил сборник альбомов "Агаты Кристи", в роли пишущего привода — PHILIPS CDRW 24000; за построение графиков и копирование дисков отвечала программа Alcohol 120%, а само копирование осуществлялось на болванку Imation 48x) (рис. 9.2).

Почувствуйте, как говорится в рекламе, разницу! Не все диски одинаковы! Причем они *очень сильно* неодинаковы, — и отличия между ними видны даже невооруженному глазу! Как это можно использовать для защиты программ? Выделяем на временной кривой ряд *узловых точек*, которые соответствуют "пикам", "провалам" или отсутствию таковых на заданном отрезке. Затем преобразуем их в код характеристик, памятуя о том, что в процессе эксплуатации диска (а так же чтении диска в других приводах) профиль кривой будет значительно меняться: одни изломы могут исчезать, а другие — появляться. Поэтому для идентификации диска придется применить алгоритм нечеткого сравнения, то есть даже при совпадении нескольких узловых точек диск считается оригинальным. Конечно, чем мягче критерии

отбора, тем больше вероятность того, что "левая" копия будет принята за правильную. Чрезмерно жесткие критерии, напротив, вызывают "угарный мат" тех пользователей, "раздолбанные" приводы которых исказили временную кривую настолько, что оригинальный диск неожиданно перестал восприниматься. По опыту автора, хороший баланс между надежностью и стабильностью представляет отношение 3:10 — т. е. если распознаны хотя бы три узловые точки из десяти, то диск считается оригинальным, ну и соответственно наоборот. В конечном счете, пусть лучше защита не заметит факта своего копирования, чем обругает законного пользователя.



а



б

Рис. 9.2. Скоростные диаграммы чтения оригинального диска (*а*) и его копии (*б*)

Кстати, для построения временной диаграммы вовсе не обязательно работать с диском на сектором уровне, — ничуть не худший результат дает измерение скорости чтения отдельных файлов штатными средствами (например, функцией `fread` языка Си). Разумеется, это должны быть большие файлы. Настолько большие, чтобы гарантированно не уместиться в кэше и заставить операционную систему обращаться к диску, а не брать эти файлы из оперативной памяти. Минус этого решения в том, что для построения временной кривой вам потребуется прочесть, по крайней мере, половину диска (а это — время), поскольку "разрешающая" способность у файлового измерителя очень низка. С другой стороны, вам ведь все равно придется читать записанные на диск данные, так почему бы тогда не совместить приятное с полезным?!

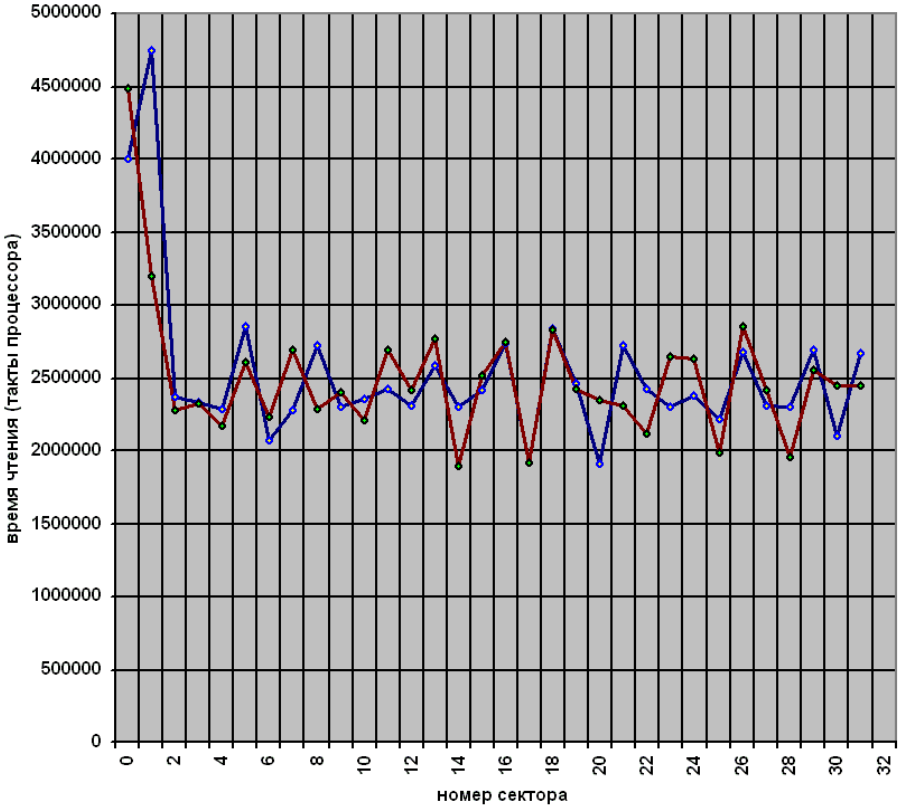


Рис. 9.3. Узловые профили, полученные при двух прогонах одного и того же диска

Если же вы "закатываете" на диск пару небольших утилит, то, конечно, для привязки к диску лучше воспользоваться замером времени чтения отдельных секторов. Поскольку в силу несовершенства технологических процессов длина секторов плавает в довольно широких пределах, то вместе с нею плавает и время, затраченное приводом, на их чтение — ведь линейная скорость вращения диска постоянна! Ну, практически постоянна. Лазерные диски, будучи по своей природе самосинхронизующимися устройствами, крайне неприхотливы к стабильности вращения привода. Главное, чтобы градиент изменения скорости был значительно ниже частоты следования питов и лендов (ведь при пересечении границы каждого из них происходит автоподстройка генератора). Легко показать, что при постоянной угловой скорости вращения диска его линейная скорость в процессе "раскрутки" спиральной дорожки неизбежно нарастает, и, если не предпринять никаких мер, то на внешних секторах диска питы будут проноситься мимо лазерной головки с такой скоростью, что она попросту не успеет их считать. Чтобы

этого не произошло, в приводах CD-ROM используются специальные механизмы динамического регулирования, удерживающие линейную скорость вращения в *заданных пределах*. К тому же, конкретное значение линейной, равно как и угловой скорости вращения, нам не известно, и его невозможно измерять с требуемой точностью. Но если скорость вращения диска неизвестна, то как прикажете определять длину сектора?

Давайте отталкиваться от того, что на коротком участке спиральной дорожки скорость вращения диска останется более или менее постоянной. Тогда, сравнивая точное время чтения соседних секторов, мы сможем приблизительно определить соотношение их длин. Действительно, если между двумя короткими секторами находится один длинный, то на графике возникнет резкий излом, изображающий "пик". Напротив, если длины трех или более секторов более или менее идентичны, то на графике образуется обширное "плато".

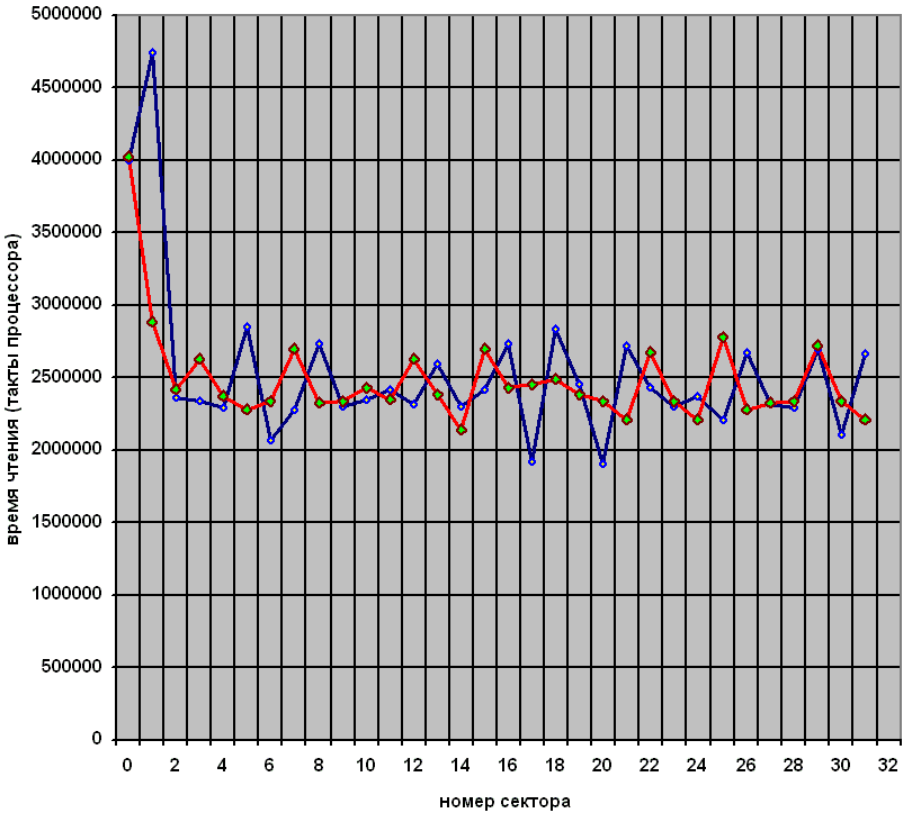


Рис. 9.4. Узловые профили двух различных дисков

Результаты обследования одного из произвольно взятых дисков выглядят так (рис. 9.3). Эта унылая изрытая горами местность, почему-то ассоциирующаяся с двуручной пилой, довольно капризна по своей природе. Повторный прогон программы с тем же самым диском никогда не даст полностью идентичный результат. Тем не менее, большинство пиков и впадин совпадают, и идентифицировать оригинальный диск все же возможно!

А теперь давайте скопируем диск и попробуем сравнить его с оригиналом (рис. 9.4). Совсем другая картина! "Рельеф местности" до неузнаваемости изменился. Во-первых, вершины пиков категорически не совпадают. Во-вторых, у дубликата отмечается на удивление невысокое количество "впадин" (видать, болванка хорошая попалась). И, наконец, в-третьих, между 14 и 22 секторами, вместо огромного "трезубца", украшающего диск-оригинал, теперь раскинулась "протяжная равнина", "нахально" загибающаяся книзу.

Таким образом, измерение относительного времени чтения секторов позволяет однозначно отличить оригинальный носитель от его копии. Причем такое измерение легко осуществимо штатными средствами операционной системы и совместимо практически со всеми моделями приводов.

Измерение угла между секторами

Услышав, что некоторые защитные механизмы измеряют угол между первым и последним логическим блоком на лазерном диске, я задумался: а как они, собственно, это делают? Поскольку самих защищенных программ в моем распоряжении не было, а технические детали разработчиками защит умалчивались, пришлось заняться логическими рассуждениями и практическими экспериментами. Ценой пары пачек безнадежно загубленных болванок CD-R и суток свободного времени, фирменный секрет был раскрыт и создана вполне работоспособная защита от копирования. Но все по порядку.

Известно, что лазерные диски представляют собой устройство последовательного доступа с ускоренной переметкой, осуществляемой радиальным перемещением головки вдоль спиральной дорожки. Переместившись на некоторое расстояние, головка сама наводится на новую дорожку и дожидается прихода ближайшей синхропоследовательности, отмечающей начало каждого сектора. Прочитав содержащийся в заголовке сектора адрес, головка сравнивает текущий адрес с искомым и, при необходимости, совершает еще одно перемещение вперед или назад. Этот процесс повторяется до тех пор, пока головка не приблизится к искомому сектору на достаточно близкое расстояние (в пределах одного оборота диска). Теперь головка прекращает суетиться и начинает спокойно ждать, пока сектор сам не приплывает в ее поле зрения.

Предположим, что поиск нужной дорожки всегда занимает одно и то же время (хотя это и не совсем так, но в качестве отправной точки рассуждений такое допущение вполне сойдет, поскольку позиционирование на сектора,

находящиеся на соседних витках спирали, осуществляется путем отклонения головки в магнитом поле, т. е. происходит практически мгновенно, и только при позиционировании на удаленные сектора головка движется на "салазках" специальным приводным механизмом, работающим со скоростью черепахи). Тогда полное время доступа к сектору будет напрямую зависеть от угла между данным и последним прочитанным сектором (рис. 9.5). Соответственно, измерив время доступа, мы сможем вычислить угол. Единственная проблема заключается в определении времени позиционирования головки. Поскольку оно сильно варьируется от привода к приводу, то закладываться на абсолютное время доступа нельзя. Однако относительные изменения видны вполне отчетливо. Последовательно гоняя головку между сектором 0 и секторами X, X+1, X+2, X+3, мы будем наблюдать волнообразные колебания полного времени доступа. Гребень волны соответствует максимальному углу между этими секторами, а впадина — минимальному (в этом случае нужный сектор попадает в головку сразу же после завершения процесса позиционирования). Запомнив, какие сочетания секторов соответствуют минимуму, а какие — максимуму, попытаемся "натравить" эту комбинацию на дубликат диска.

Что, не получилось? Действительно, различные партии болванок CD-R отформатированы далеко не идентичным образом и плотность спиральной дорожки у них различна. А раз так, то различным оказывается и угол между нашими секторами, причем эти различия стремительно нарастают с удалением секторов друг от друга. Предположим, что средняя длина секторов оригинала и дубликата отличается на 0,01%. Тогда при условии, что полная емкость диска составляет ~350 000 секторов, изменение угла между первым и последним сектором диска составят 3,5% — а это вполне измеряемая величина! Причем на практике указанная точность разметки практически никогда не наблюдается и при копировании эталонного диска на болванки других производителей "поворот" угла порой достигал 180 градусов, т. е. половины оборота диска!

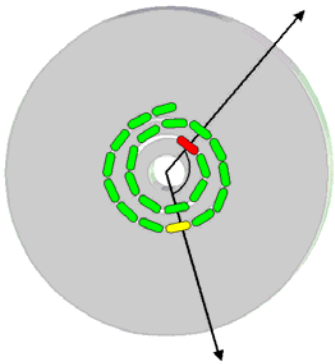


Рис. 9.5. Измерение угла между секторами

Запустив программу, измеряющую межсекторные углы на выполнение, и дав приводу CD-ROM вволю прошуршать головкой (собственно, по этому шуршанию алгоритм работы защиты и распознается), мы обнаружим, что время доступа к секторам с различными номерами изменяется весьма любопытным образом. Четыре или пять соседних секторов читаются приблизительно с одинаковой скоростью, а затем кривая резко изгибается, изменяя время доступа чуть ли не вдвое! Однако, спустя один или несколько секторов, время доступа вновь скачкообразно изменяется. Чередование пиков и провалов обнаруживает достаточно строгую периодичность, отклоняющуюся от всего среднего положения всего на несколько секторов, что, судя по всему, вызвано непостоянством времени перемещения оптической головки. Естественно, чем сильнее изношен дисковод, тем с меньшей точностью удается определить период, однако при большом количестве замеров величина погрешности окажется относительно невелика.

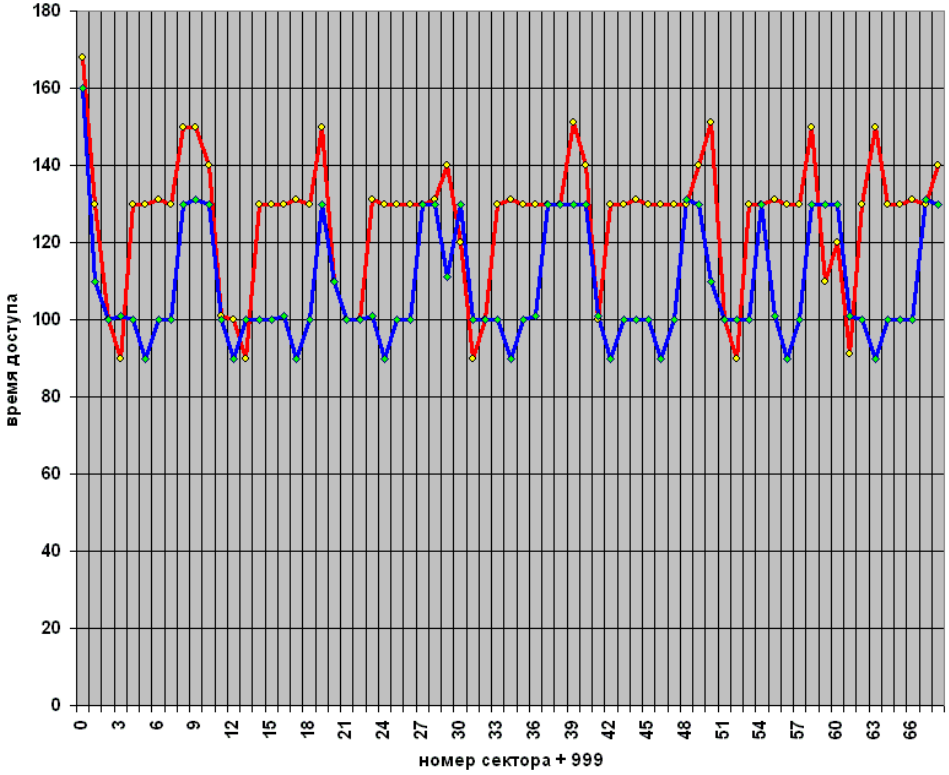


Рис. 9.6. Профиль спиральной дорожки диска Imation (темно-серая кривая) и TDK (черная кривая)

На рис. 9.6 приведены профили спиральных дорожек двух различных болванок. Темно-серая (на цветном рисунке красная) кривая соответствует болванке Imation, а черная (на цветном рисунке голубая) — болванке TDK. Обратите внимание, насколько отличается один график от другого!

Защиты, основанные на "слабых" секторах

Столкновение с этими защитами поразило даже меня, видавшего виды хакера. Да как тут было не поразиться! Судите сами — копирование защищенного диска происходит без ошибок, но при проверке копии обнаруживается множество сбойных секторов, которые появляются даже в том случае, если содержимое оригинального диска пофайлово перегнать на винчестер и уже оттуда записать его на болванку CD-R. Что это: неисправность оборудования или результат работы хитрого драйвера, скрыто установленного защитой при первом запуске защищенной программы? Но нет, — все оборудование полностью исправно и никакие драйвера тут не причем, сбойные сектора возникают даже при копировании диска на заведомо "стерильной" машине. Руки, естественно, также "не кривые" и голова "не дырявая".

Исследование самих копируемых файлов под отладчиком (HEX-редактором, дизассемблером) также не выявляет ничего необычного, и если "отломать" защиту от привязки к CD (при условии, что она там есть), то защищенное приложение будет успешно запускаться с жесткого диска (Zip-накопителя), но вот "закатать" его на болванку, увы, не получится. Правда, если защищенные файлы каким-либо образом "подрихтовать" (например, сжать любым архиватором), то перенос на болванку CD-R пройдет без ошибок, но ведь это уже совсем не то, верно?

Таким образом, причина столь странного поведения защиты лежит отнюдь не на программном, а физическом уровне. Это самый "крутой" антиотладочный прием, которой мне только доводилось когда-либо встречать! Действительно, с точки зрения хакера, не обремененного сложным измерительным оборудованием, привод CD-ROM представляет собой черный ящик, работающий приблизительно по такому же принципу, как и любой другой накопитель. Даже если снять с него крышку, мы все равно ничего не увидим кроме переплетений проводов и нагромождений микросхем. Единственное, что остается — вдумчиво перечитывать стандарты. Ведь если защита работает на всех (или хотя бы подавляющем большинстве) моделей приводов CD-ROM, то она должна обязательно опираться на те или иные стандартные свойства/особенности/характеристики!

Вот, допустим: *"A regular bit pattern fed into the EFM encoder can cause large values of the Digital Sum Value in case the merging bits cannot reduce this value (see annex E). The scrambler* ***reduces this risk*** *by converting the bits in byte 12 to 2 351 of a Sector in a prescribed way" ("Регулярные битовые шаблоны, скормленные EFM-декодеру,*

могут вызвать большие значения DSV, неустранимые merging-битами. Скремблер ***сокращает риск*** *этого путем преобразования бит с 12 по 2.351 байт сектора определенным образом".)*[6]. Если вы все еще продолжаете думать, что диски CD-ROM — идеальные носители для хранения исполняемых файлов и баз данных, то вы глубоко заблуждаетесь. Лазерные диски изначально разрабатывались для записи музыки и лишь ценой больших инженерных извращений согласились хранить бинарные данные. Обратите внимание на выделенные полужирным шрифтом слова. Скремблер не гарантирует, что записанные данные смогут читаться, он лишь снижает риск образования неблагоприятных (с точки зрения привода) последовательностей до разумного минимума. Тем не менее, если постараться, то вполне можно создать пару файлов, напичканных этими самыми наиболее неблагоприятными последовательностями по самую завязку. Теоретически их можно прочесть, но с этим справятся лишь наиболее качественные модели приводов CD-ROM, а остальные дадут ошибку.

Рассмотрим следующую комбинацию: `04 B9 04 B9 04 B9`. Обратившись к таблице EFM-перекодировки, найдем, что `04` преобразуется в `01000100000000`, а `B9` — в `10000000001001`. Попробуем теперь их записать вместе: `01000100000000 xxx 10000000001001 yyy 01000100000000`, где `xxx` и `yyy` — merging-биты. Поскольку `04` содержит в своем хвосте восемь нулей, а `B9` начинается с единицы, то единственной возможной комбинацией для первой партии merging-бит будет `100`. Соответственно, поскольку `B9` кончается на единицу, а `04` имеет в своей голове всего один нуль, то единственно подходящей комбинацией для второй партии объединяющих бит окажется `000`. А теперь изобразим это графически и вставим разработчикам CD-ROM хороший "пистон".

Смотрите (рис. 9.7)! Значение DSV оказывается резко отрицательным! То есть, попросту говоря, питы преобладают над лендами и поверхность диска становится темной, как негр. Как следствие — следящее устройство сбивается с дорожки из-за недостаточного количества света, попадающего в фотоприемник. Самое интересное, что по стандарту такие комбинации битов читаться и не обязаны (хотя некоторые модели приводов с ними все-таки справляются). Вот вам и надежное устройство для хранения данных! В силу природы своих конструктивных особенностей, лазерные диски способны хранить лишь некоторые.

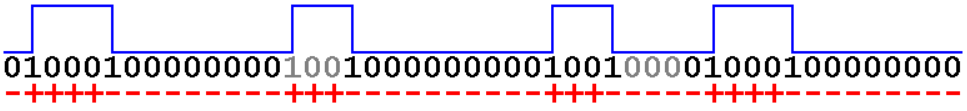


Рис. 9.7. Физическое представление последовательности `04 B9 04` при записи

[6] Выделение полужирным шрифтом сделано автором книги. — *Ред.*

Конечно, если просто создать файл, битком набитый `\x04\xB9\x04\xB9`, его запись на болванку CD-R и последующее чтение пройдут без проблем, ведь поток записываемых данных предварительно обрабатывается скремблером! Грамотно выбранный алгоритм скремблирования не должен допускать эффективного обращения, в противном случае злоумышленник может прогнать наиболее неблагоприятные регулярные последовательности через "антискремблер" и тогда при повторном скремблировании они запишутся на диск в своем исходном виде. Теперь собственно "пистон". Алгоритм скремблирования, используемый приводами CD-ROM, такое обращение вполне допускает! Все, что понадобится для написания антискремблера, — это пара вечеров свободного времени и сам стандарт ECMA-130. Поскольку алгоритм скремблирования базируется на функции `XOR`, то повторное скремблирование уже обработанных скремблером данных возвращает их в исходный вид. Благодаря этому обстоятельству, мы сможем обойтись и одной функцией — функцией скремблера.

Прогнав через скремблер защищенные файлы, мы не без удовлетворения обнаружим, что они содержат по крайней мере одну очень неблагоприятную последовательность, значение DSV которой резко отрицательно (реже — резко положительно). В общем, категорически не равно нулю. Дополнив скремблер функцией подсчета DSV (подробности о ее реализации см. в ECMA-130), мы получим автоматический сканер защит, основанный на слабых секторах. Wow! Разве не "круто"?! Если внутри защищенных файлов обнаружатся те или иные неблагоприятные последовательности, то можно и не пытаться их копировать на CD-R — все равно из этой затеи ничего не получится.

Хорошо, но как нам объяснить тот факт, что эти же самые неблагоприятные последовательности успешно читаются с оригинального диска?! Чтобы ответить на этот вопрос, нам придется забраться в самые дебри спиральных дорожек лазерного диска. Труден и опасен будет этот путь! Вас станут отговаривать и предостерегать. Вот, например, цитата из одного реферата неизвестного автора: *"...на самом деле все еще интереснее, т. к. в дополнение к секторам определяются секции того же полезного размера, но с несовпадающими границами, причем часть адресов является адресами секторов, а другая — адресами секций. Но об этом лучше сразу забыть :)"*. Смайлик на конце этих строк очень сильно возбуждает и заставляет перечитывать и перечитывать сухие строчки стандарта (поскольку ничего более информативного под рукой, увы, нет). Так или иначе (на худой конец, посетите форум на сайте **http://club.cdfreaks.com**) мы выясним, что границы секторов и фреймов могут и не совпадать, — сектор может начинаться с 0, 4, 8, 12, 16 или 20-го байта по счету! Изменение стартовой точки неизбежно изменяет и DSV первого фрейма, — вот тут-то и начинается самое интересное. Если количество бинарных единиц фрейма нечетно, то второй фрейм инвертируется (то есть питы и ленды меняются местами), в противном случае последующий фрейм идет так, как он есть. Благодаря этому становится возможным подобрать

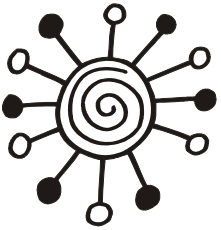
такую регулярную последовательность, которая окажется вполне благоприятной для одной из точек входа и крайне неблагоприятной для всех других.

К сожалению, пишущие приводы все еще не позволяют выбирать произвольную точку входа и назначают ее самостоятельно по своему усмотрению. Хорошие пишущие приводы (типа Plextor) выбирают точку входа так, чтобы значение DSV сектора было минимальным по модулю (и потому позволяют копировать защищенные диски без проблем). К сожалению, подавляющее большинство остальных моделей слишком тупы и задача минимизации DSV им оказывается "не по зубам". Они либо вовсе не пытаются вычислить правильную точку входа, либо же вычисляют ее неправильно. Как следствие — при чтении скопированных защищенных дисков возникает ошибка!

Тем не менее, "продвинутые" копировщики (к которым, в частности, относится CloneCD) такие защиты уже давно обошли. Как они ухитряются это делать? Подготовив образ сектора для "нарезки" в "сыром" виде, они слегка уродуют его содержимое, рассыпая тем самым неблагоприятные последовательности в пух и прах (искажения одного бита исходных данных вызывает драматические изменения данных после скремблирования!). Корректирующие коды (ранее подготовленные для неизуродованного содержимого сектора) при этом *не изменяются*. В результате, чтение записанного таким образом сектора дает ошибку и приводу приходится ее исправлять по избыточной информации, содержащейся в корректирующих кодах. После исправления, сектор вновь возвращается в неискаженный вид.

Достоинство такого подхода заключается в том, что копия защищенного диска содержит ослабленный защитный "организм", свободно дублируемый в "сыром" режиме. Штатное копирование, правда, будет приводить к ошибке, поскольку честные копировщики помещают на диск уже исправленные сектора. С другой стороны, копировщики, записывающие сектора как они есть, оказываются не в состоянии отличить умышленные искажения от простых физических ошибок чтения. Запись неисправленного сектора приведет к увеличению числа ошибок, и при попытке снять копию с копии мы рискуем получить вообще не читаемый дубликат, что есть большой недостаток! Какой из этого выход? Читаем сектор, исправляем его, прогоняем через сканнер и, если в нем обнаружится та или иная неблагоприятная последовательность, умышленно изменяем в нем несколько бит. Таким образом, ошибки копирования больше не будут накапливаться!

Глава 10



Техника восстановления данных с лазерных дисков или практическое знакомство с сессиями

> ...такой объем информации можно уничтожить в один миг разве что динамитом, потому что существуют дублирующие системы и у скорости обработки есть предел.
>
> *Джон Варли. "Нажмите ENTER"*

Записываемые и перезаписываемые лазерные диски представляют собой идеальное средство для резервирования информации умеренных объемов (а всякий уважающий себя программист обязательно должен заботиться о периодическом резервировании вверенной ему информации!). К сожалению, никакая работа без ошибок не обходится (что поделаешь — ERRARE HUMANUM EST — человеку свойственно ошибаться— как говорили древние) и ошибочное удаление файлов с дисков CD-R и CD-RW, равно как и непредумышленная очистка последних — хотя бы однажды да случается (на самом же деле, как показывает практика, с этим явлением приходится сталкиваться далеко не однажды).

Насколько известно автору, утилит, предназначенных для восстановления информации с лазерных дисков, до сих пор не разработано (во всяком случае, они не были широко представлены на рынке), поэтому восстановлением "запоротых" дисков в подавляющем большинстве случаев приходится заниматься самостоятельно. О том, как именно это сделать, и рассказывает настоящая глава.

Восстановление удаленных файлов с CD-R/CD-RW

Заявляя о своей поддержке многосессионных дисков, операционные системы Windows 9x и Windows NT (вплоть до Windows 2000 включительно) тактично умалчивают о том, что поддерживают их лишь частично. Каждая

сессия — это вполне самостоятельный том (в терминологии Windows — *"логический диск"*), имеющий свою собственную файловую систему и свои собственные файлы. Благодаря сквозной нумерации секторов лазерного диска, файловая система одной сессии может ссылаться на файлы, физически расположенные в любой другой сессии. Для того чтобы с многосессионным диском можно было работать как с единым томом, файловая система последней сессии должна включать в себя содержимое файловых систем всех предыдущих сессий. Если этого не сделать, то при просмотре диска штатными средствами Windows, оглавления остальных сессий окажутся потерянными, поскольку Windows монтирует лишь последнюю сессию диска, а все прочие — игнорирует. Программы "прожига" CD-R/CD-RW по умолчанию добавляют содержимое файловой системы предыдущей сессии к последующей, однако это еще не означает, что последняя сессия диска *всегда* содержит в себе все то, что имеют предыдущие.

Рассмотрим например, как осуществляется удаление файлов с CD-R/CD-RW. Нет, это не опечатка! Содержимое дисков CD-R, несмотря на физическую невозможность их перезаписи, в принципе все же уничтожаемо. Для имитации удаления файла, программы записи на CD просто не включают ссылку на уничтожаемый файл в файловую систему последней сессии (правда, это умение даровано не всем, вот программе Roxio Easy CD Creator оно даровано, а, например, Stomp Record Now! — нет). И хотя "удаленный" файл все еще присутствует на диске, "отъедая" часть дискового пространства, при просмотре содержимого диска из-под Windows он уже не отображается в каталоге. Какой же тогда смысл несет в себе удаление файлов с CD-R, если свободная емкость диска при этом не увеличивается, а даже *уменьшается*?! — удивленно спросит иной читатель. Уменьшение свободного пространства объясняется тем, что каждая открываемая сессия требует для своего размещения определенного места, однако, если удалению одних файлов сопутствует запись других, то открывать новую сессию по любому приходится и в этом случае, накладные расходы на удаление отсутствуют. На самом же деле, смысл этой операции (если его вообще можно назвать "смыслом") заключен исключительно в сокрытии "удаляемых" файлов от простых пользователей. Раз удаленные файлы не видны при просмотре содержимого диска штатными средствами, то неквалифицированному пользователю они формально недоступны. Подчеркиваю: для штатных средств операционной системы Windows — недоступны, но те же "Маки"[1] позволяют монтировать любую сессию диска на отдельный том, благодаря чему при просмотре многосессионных дисков под "Маками" все удаленные файлы сразу же "всплывают".

Аналогичным образом обстоят дела и при удалении информации с дисков CD-RW. Несмотря на теоретическую возможность физического уничтожения

[1] Автор имеет в виду Mac OS (Macintosh Operating System) — операционную систему фирмы Apple Computer для ПК Macintosh. — *Ред.*

их содержимого, подавляющее большинство записывающих программ поддерживают лишь функцию очистки всего диска целиком, но не в состоянии выборочно удалять отдельные файлы. Так что все, сказанное ранее о дисках CD-R, в равной мере применимо и к CD-RW.

Записывая на диск информацию, предназначенную для передачи постороннему лицу, ни в коем случае не используйте для этой цели болванки, содержащие конфиденциальные данные. "Удаление" ранее записанных на болванку данных на самом деле не уничтожает их!

Просматривая содержимое лазерного диска, полученного от приятеля (купленного на радио-рынке, вытащенного из мусорной корзины), имеет смысл попытаться заглянуть внутрь предыдущих сессий на предмет поиска скрытой информации. Как показывает практика, там действительно очень часто обнаруживается много интересного. С другой стороны, вам может потребоваться восстановить ошибочно удаленный файл со своего собственного диска, а то и "воскресить" всю "пришибленную" сессию целиком (некоторые программы записи на CD позволяют пользователю выбирать: следует ли при создании новой сессии добавлять в нее файловую систему предыдущей или же в новую сессию следует включать только новые файлы. Неверный выбор настроек приводит к утрате содержимого всех предыдущих сессий, но, к счастью, эта утрата обратима).

Отсутствие штатных средств выборочного монтирования сессий значительно усложняет жизнь всем пользователям Windows, заставляя искать обходные пути. В идеале, конечно, следовало бы реализовать своей собственный драйвер CDFS, обладающий максимумом функциональных возможностей. Однако это довольно сложная и трудоемкая задача, оправдывающая себя лишь в тех случаях, когда с необходимостью восстановления удаленных файлов вы сталкиваетесь по несколько раз на дню. Гораздо проще написать набор утилит для непосредственной работы с диском на физическом уровне. Собственно, все, что нам требуется, — это отобразить содержимое файловой системы в удобно-читаемом виде. Конкретно нас интересует: имя файла, его стартовый адрес и длина. Знания трех этих важнейших атрибутов без труда позволяют "сграбить" файл на жесткий диск, где вы сможете работать с ним в свое удовольствие. Такая методика идеальна для восстановления небольших количеств удаленных (перезаписанных) файлов из произвольных сессий, но она непрактична в тех ситуациях, когда приходится восстанавливать всю сессию целиком, и тогда обычно прибегают к копированию восстанавливаемых сессий на отдельный CD-R/CD-RW-диск.

Получение доступа к удаленным файлам

Прежде чем приступить к восстановлению удаленных файлов, давайте-ка вспомним основные принципы организации файловой системы ISO 9660 (Joliet). Итак, шестнадцатый сектор первого трека каждой сессии жестко

закреплен за дескриптором тома. Его легко узнать по сигнатуре "CD001", хранящейся в секторе по смещению 1. Если это действительно так (ну мало ли, вдруг нам подсунули диск без файловой системы, например, аудиодиск), тогда по смещению 156 в секторе расположена структура `Directory Record` корневого каталога. Наибольший интерес представляют ее следующие поля: длина самой `Directory Record` (байт по смещению 0), стартовый LBA-адрес файла/вложенной директории (двойное слово в low-endian формате по смещению 2), длина файла/вложенной директории (двойное слово в low-endian-формате по смещению 10), атрибуты файла (байт по смещению 25), длина имени файла/вложенной директории (байт по смещению 32) и, наконец, непосредственно само имя файла (цепочка байт, начинающаяся со смещения 33). Файловая система Joliet устроена аналогичным образом, но соответствующий ей дескриптор тома расположен не в шестнадцатом, а в семнадцатом секторе.

Если первый, считая от нуля, бит атрибутов равен единице, то мы имеем дело с вложенным каталогом, в противном случае — с файлом. Вложенные каталоги представляют собой совокупность записей типа `Directory Record`, каждая из которых указывает либо на целевой файл, либо на очередную вложенную директорию.

Таким образом, для просмотра содержимого произвольной сессии нам потребуется всего лишь узнать стартовый адрес ее первого трека. Эту информацию легко получить путем чтения TOC на "сыром" уровне (команда: `43h`, формат: `02h`). Также можно воспользоваться любой утилитой "прожига" дисков, способной выдавать информацию о геометрии диска (для этой цели подходят, в частности, Roxio Easy CD Creator, Stomp Record Now, Nero Burning Rom и многие другие). Увеличив стартовый адрес первого трека на 16 секторов (для работы с ISO-9660) или на 17 секторов (для работы с Joliet), мы попадем на описатель тома, то есть — на корневой каталог. А дальше уже рекурсивный спуск по древу каталогов пойдет "как по маслу" (под Windows можно даже не заботиться об исчерпании стека, если, конечно, файловая система не содержит грубых ошибок, приводящих к ее зацикливанию. Между прочим, такие трюки с зацикливанием достаточно часто используются в защитах).

Остается лишь пролистать каталоги всех дисковых сессий на предмет поиска файлов, отсутствующих в каталоге самой последней сессии. При этом следует обращать внимание не только на имена файлов, но и на их стартовые адреса. Файлы с идентичными именами, но различными стартовыми адресами — это различные файлы! Например, если вы периодически сохраняете свой текущий проект на лазерный диск, все время записывая его под одним и тем же именем, то все предыдущие версии этого файла для штатных средств операционной системы оказываются утерянными. Однако "ручной" просмотр содержимого сессий позволяет моментально восстановить любую из ранее записанных версий файла! Между прочим, необходимость

подобного восстановления на практике возникает очень и очень часто, поэтому такое умение лишним не будет!

Для последующих экспериментов нам потребуется утилита ISO9660.dir, которая расположена на прилагающемся к книге компакт-диске. Используя любую программу "прожига", запишем на диск CD-RW/CD-R три сессии с таким расчетом, чтобы в первой или второй сессии образовалось несколько удаленных файлов, которые нам, собственно, и предстоит найти.

Допустим, стартовые адреса дисковых сессий располагались так, как это показано в листинге 10.1. Трек номер `AA` представляет собой выводную область и не представляет для нас никакого интереса.

Листинг 10.1. Стартовые адреса первых треков каждой из трех сессий диска

```
> ISO9660.dir.exe 1.1
track |  Start LBA
------+-----------
  1   |          0
  2   |      13335
  3   |      22162
 AA   |      24039
```

Теперь, последовательно вызывая утилиту ISO9660.dir.exe со стартовыми адресами 0, 13 335 и 22 162, мы увидим содержимое каталогов первой, второй и третьей сессий соответственно. Для удобства сравнения результат работы программы представлен в виде горизонтальной таблицы (листинг 10.2). При сравнении видно, что вторая сессия содержит удаленный файл See You.mp3 (в тексте он выделен полужирным шрифтом), отсутствующий в оглавлении третьей сессии.

Листинг 10.2. Сравнение содержимого каталогов трех сессий

>ISO9660.dir 1.1 0 -Joliet	>ISO9660.dir 1.1 13335 -Joliet	>ISO9660.dir 1.1 22162 -Joliet
start|size |name	start|size |name	start|size |name
-----+-------+----------	-----+-------+----------	-----+-------+----------
22 |2048 |.	13357|2048 |.	22184|2048 |.
22 |2048 |..	13357|2048 |..	22184|2048 |..
25 |3591523|PersonalJesus.mp3	25 |3591523|PersonalJesus.mp3	25 |3591523|PersonalJesus.mp3
	13360|3574805|See You.mp3	22187|3472405|Strangerlove.mp3

Видите, вторая сессия содержит файл See You.mp3, который отсутствует в оглавлении третьей сессии, а это значит, что штатным средствам операционной системы он категорически недоступен! И действительно, команда `dir`

показывает всего лишь два файла (листинг 10.3). Но нас-то не проведешь! Мы-то уже знаем, что здесь содержится один удаленный файл.

Листинг 10.3. Таким представляется содержимое диска операционной системе

```
Том в устройстве G имеет метку NEW
Серийный номер тома: 4171-70DC

 Содержимое папки G:\

30.06.2003  00:10            3 591 523 01 - Personal Jesus.mp3
30.06.2003  00:12            3 472 405 03 - Strangerlove.mp3
               2 файлов      7 063 928 байт
               0 папок               0 байт свободно
```

Для восстановления файла достаточно запустить нашу утилиту со следующими ключами: `ISO9660.dir.exe 1.1 "See You.mp3" 13360 3574805` или воспользоваться любой программой для "грабежа" секторов с диска (но тогда нам придется вручную усечь "хвост" последнего сектора на необходимую величину). Короче говоря, мы должны считать 3 574 805 байт, начиная с сектора 13 360 и до последнего сектора файла включительно (в силу отсутствия фрагментации сектора, принадлежащие файлу, всегда располагаются последовательно).

Если все сделано правильно, то на жестком диске образуется файл See You.mp3, который теперь можно проиграть любым MP3-плеером, наслаждаясь своей любимой музыкой. А можно, используя любую программу "прожига", "закинуть" восстановленный файл на тот же самый лазерный диск, дописав к нему еще одну сессию. Конечно, это далеко не самое лучшее и элегантное решение (ведь теперь восстановленный файл будет записан дважды), но, к сожалению, ни одна из известных мне утилит "прожига" не позволяет вмешиваться в процесс генерации файловой системы и не позволяет вручную формировать ссылки на файлы предшествующих сессий.

Теперь подготовим следующий диск для восстановления. Запишем на болванку какой-нибудь файл. Затем, слегка изменив его содержимое, запишем под тем же самым именем на диск еще один или несколько раз. Убедимся, что при просмотре оглавления штатными средствами Windows мы видим всего лишь один файл — тот, что был записан последним. А теперь вообразим себе, что нам жизненно потребовалось получить доступ к одной из его предшествующих версий. "No problem", — как говорят англичане! Запускаем утилиту ISO9660.dir и видим, что на диске имеются две следующих сессии (листинг 10.4).

Листинг 10.4. На восстанавливаемом диске имеются две сессии со стартовыми адресами 0 и 12000 соответственно

```
>ISO9660.dir.exe 1.1
track |  Start LBA
------+-----------
  1   |          0
  2   |      12000
 AA   |      12600
```

Последовательно запуская утилиту ISO9660.dir с ключами 0 и 12000, мы обнаружим, что А, впрочем, давайте не будем забегать вперед. Сейчас вы сможете увидеть все это сами (листинг 10.5).

Листинг 10.5. Обе сессии содержат один и тот же файл asm.drf.zip, однако стартовые адреса файла в обоих случаях не совпадают, да и длина различна тоже

```
> ISO9660.dir.exe 1.1 0 -Joliet
start     | size     |  name
----------+----------+------------
22        |2048      |.
22        |2048      |..
25        |38189     |asm.drf.zip
```

```
> ISO9660.dir.exe 1.1 12000 -Joliet
start     | size     |  name
----------+----------+---------------
12022     |2048      |.
12022     |2048      |..
12025     |354533    |asm.drf.zip
```

На первый взгляд здесь все нормально: файл asm.drf.zip присутствует в обоих сессиях, и невнимательному наблюдателю может показаться, что в обоих случаях речь идет про один и тот же файл. Однако более подробный анализ показывает, что стартовые адреса файла не совпадают, следовательно, мы имеем дело с двумя различными файлами! Длины файлов также различны, и, судя по всему, последний файл является более свежей версией своего собрата. Добыть предшествующую версию файла можно так: `ISO9660.dir.exe 1.1 asm.drf.zip 25 38189`. И этот прием действительно работает!

Разумеется, для практического использования утилита ISO9660.dir.exe катастрофически неудобна. Однако не судите ее строго, ведь это всего лишь демонстрационный пример. Надеясь пробудить в своих читателях стремление к творчеству, я предлагаю всю недостающую функциональность реализовать самостоятельно. Задача минимум — "обернуть" утилиту в комфортный графический (а еще лучше консольный) интерфейс, например, написать "плагин" к файловому менеджеру FAR, отображающий содержимое выбранной сессии на панели. Это гораздо проще, чем запрограммировать полноценный дисковый драйвер, а эффект, с точки зрения пользователя, все равно один и тот же. Задачу сравнения имен и стартовых адресов файлов также не помешало бы возложить на компьютер, поскольку ручной поиск удаленных (заме-

щенных) файлов крайне непродуктивен — типичный CD-ROM содержит тысячи, если не десятки тысяч файлов, рассредоточенных по мощной древовидной структуре. Вот и попробуй перебери их все!

Восстановление целых сессий

Альтернативный способ восстановления удаленных файлов заключается в удалении одной или нескольких последних сессий с диска (условимся называть этот метод "очисткой луковицы"). При этом обнажается предыдущий "слой", который теперь становится последним, а значит, непосредственно доступным для штатных средств операционной системы. Разумеется, физическое удаление сессий с CD-R дисков невозможно в принципе (удаление сессий с CD-RW дисков теоретически вполне возможно, но практически проблема упирается в отсутствие необходимого софта), однако ничто не мешает "скинуть" образ диска в файл, обработать его должным образом и "залить" обратно (на CD-RW — обратно, а в случае с CD-R нам потребуется свежая болванка). Никто не спорит — все это нудно и утомительно, но и писать свой "прожигающий" софт ничуть не быстрее.

Хорошо, начинаем готовить экспериментальный диск. Запишем на подопытную болванку одну или несколько сессий, указав программе "прожига" объединять файловую систему новой сессии со всеми предыдущими (обычно так и происходит по умолчанию), а затем добавим еще одну сессию, на этот раз стоящую ото всех остальных особняком. Для этого в программе Ahead Nero Burning Rom необходимо выбрать пункт **Start Multisession disk** вместо **Continue Multisession**, а в Stomp Record Now — пункт **New Volume** вместо **Load Last Track**. Включаем рекордер на "прожиг" и убеждаемся, что содержимое всех предыдущих сессий оказалось утрачено. Впрочем, справедливости ради следует отметить, что та же программа Stomp Record Now позволяет легко "починить" загубленный диск — просто выберите в ниспадающем меню **Mutlisession** пункт **Load Track 1** и запишите на диск произвольный файл (это необходимо для инициирования процесса записи). Теперь на диске появится содержимое сессии номер один, а сессия номер два "уйдет в туман". Но это ли беда? Меняем пункт **Load Track 1** на **Load Track 2** и вновь инициируем запись. Теперь на диске появляется содержимое первых двух сессий плюс все те файлы, которые мы были вынуждены записывать, чтобы программа Stomp Record Now согласилась на "прожиг".

Программа Ahead Nero Burning Rom при выборе пункта **Continue Multisession** автоматически запрашивает у пользователя, оглавление какой из сессий следует использовать. К сожалению, возможности слияния оглавлений двух и более сессий в текущих версиях не предусмотрено, но в случае возникновения такой необходимости можно прибегнуть к последовательной "каскадной" методике, описанной ранее.

В отсутствие же Stomp Record Now (или аналогичных ей программ) приходится осуществлять восстановление затертых сессий вручную, то бишь при

помощи программ наподобие CloneCD или Alcohol 120%. Создаем еще один "подопытный" диск, последняя сессия которого перекрывает собой все остальные, "натравливаем" на него CloneCD с целью создания образа диска, и теперь остается выкинуть все упоминания о последней сессии из файла IMAGE.CCD. Во-первых, значение поля `Sessions` необходимо уменьшить на единицу, во-вторых, выкинуть все секции `[Entry]`, чей Session был последним, в-третьих, вычесть из поля `TocEntries` количество удаленных `[Entry]`, наконец, в-четвертых, удалить последнюю секцию `[TRACK]`. Теперь отредактированный образ можно смело писать на свежую болванку, и последней сессии как не бывало!

Ошибки начинающих или то, чего делать не следует

Иногда можно услышать утверждение, что, "сграбив" одну из сессий восстанавливаемого диска в ISO-образ и смонтировав его на виртуальный CD-ROM-диск (записав на болванку), можно получить доступ к его содержимому. Действуя таким макаром, мы сможем быстро просмотреть содержимое всех необходимых нам сессий.

Так или иначе, доступ к удаленным файлам будет получен и вы сможете делать с ними все, что хотите.

Внимание

При просмотре содержимого "сграбленной" сессии всегда учитывайте, что: во-первых, файлы, физически принадлежащие другим сессиям, из данной сессии окажутся недоступными, в то время как ссылки на них здесь могут изобиловать. При обращении к реально несуществующему файлу будет выдаваться либо "мусор", либо сообщение об ошибке. Как альтернативный вариант — операционная система может просто зависнуть. Если это произошло, то просто нажмите кнопку выброса диска "Eject". Windows тут же выйдет из ступора и радостно "завопит" — "устройство не готово". Во-вторых, в силу сквозной адресации секторов, каждая "сграбленная" сессия должна записываться на то же самое место диска, на котором она была ранее, в противном случае все ссылки на стартовые адреса файлов внутри этой сессии окажутся недействительными. Требуемый результат обычно достигается изменением стартового адреса первого трека. О том, как это сделать, рассказывается в следующем разделе главы, посвященном восстановлению информации с очищенных CD-RW-дисков.

Восстановление очищенных CD-RW

Существует две принципиально различных методики очистки CD-RW: *быстрая* (*quick*) и *полная* (*full*). При быстрой очистке диска с него удаляется лишь область TOC, в результате чего диск выглядит "пустым", хотя его основное содержимое остается совершенно нетронутым. Напротив, при полной очистке луч лазера "выжигает" всю поверхность диска целиком — от первого пита

до последнего. Естественно, на это требуется время и полная очистка диска может растянуться на добрый десяток минут, в то время как быстрая спокойно укладывается в одну-две минуты.

Восстановление полностью очищенных дисков возможно только на специальном оборудовании, способном улавливать даже незначительные изменения отражательной способности рефлекторного слоя. Такое оборудование подавляющему большинству пользователей, разумеется, недоступно. Однако диски, подвергнувшиеся быстрой очистке, могут быть восстановлены и на штатном рекордере (правда, не на всех моделях).

Мы не будем касаться этической стороны проблемы и для простоты предположим, что вы хотите реанимировать свой собственный непредумышленно очищенный CD-RW-диск, или условимся считать всех читателей сотрудниками КГБ, которым поручили восстановить информацию с диска, добытого бесстрашными советскими разведчиками у американских шпионов. Отметим лишь то, что восстановление конфиденциальной информации с чужих CD-RW может быть классифицировано как получение несанкционированного доступа к последней со всеми вытекающими отсюда последствиями (на долгие годы — друзья в клетку и небо в полоску).

Для опытов по восстановлению информации с очищенных CD-RW-дисков нам потребуется следующее:

- *пишущий привод*, не слишком дотошно следящий за корректностью содержимого TOC, поддерживающий режим RAW DAO и умеющий читать содержимое области Pre-gap первого трека. Не все модели пишущих устройств подходят для этой цели, поэтому будьте готовы к тому, что вам придется перепробовать большое количество различного оборудования (из двух моих рекордеров для восстановления очищенных дисков подходит лишь NEC, а PHILIPS на это, увы, не способен);
- *"продвинутый" записывающий софт*, позволяющий манипулировать служебными областями диска по своему усмотрению. Вы можете использовать CloneCD, CDRWin, Alcohol 120% или любую другую аналогичную утилиту по своему выбору. Однако весь последующий материал рассчитан исключительно на CloneCD, и при переходе на остальные программы вы можете столкнуться с теми или иными проблемами. Если вы не уверены, что сможете справиться с ними самостоятельно, — используйте CloneCD, ну а затем, по мере приобретения профессиональных навыков и должного опыта, вы без труда восстановите диск любой такой программой;
- *средство для работы с диском на секторном уровне* — утилита, позволяющая прочесть любой заданный сектор (конечно, при условии, что он вообще читается приводом) и не пытающаяся пропустить те сектора, в которых по ее самоуверенному мнению ничего интересного все равно нет. Копировщики защищенных дисков, перечисленные выше, для этой цели

не подходят, т. к. отказываются читать "бесполезные" с их точки зрения сектора. Может быть, другие копировщики ведут себя и иначе — не знаю, не проверял. Вместо этого необходимую для работы утилиту я написал самостоятельно.

Прежде чем начинать экспериментировать, давайте разберемся, почему после очистки диск перестает читаться. Вопрос не так глуп, каким он кажется, — ведь информация, необходимая для позиционирования головки и поиска конкретных секторов, при быстрой очистке диска остается нетронутой! Управляющие данные "размазаны" вдоль всей спиральной дорожки и для чтения диска на секторном уровне TOC, в общем-то, и не нужен. Да, отсутствие TOC значительно усложняет анализ геометрии диска и для определения количества треков/сессий диска, в общем случае, привод должен прочитать весь этот диск целиком. Но при восстановлении информации временной фактор играет второстепенную роль, и им можно полностью пренебречь.

Тем не менее, при попытке чтения любого из секторов очищенного диска, привод с неизменным упорством возвращает ошибку. Почему? Очень просто, — это "защита" от чтения заведомо некорректной информации. Еще ни один из всех знакомых мне приводов не мог читать сектора за пределами области Lead-Out (собственно, на программном уровне содержимое областей Lead-In/Lead-Out недоступно тоже). Тем не менее, эта невозможность отнюдь не концептуального уровня и удаление из микропрограммы привода "лишних" проверок позволят прочитать такой диск на ура. Нет, даже не подумайте! Призывать вас к дизассемблированию прошивок я не собираюсь. Дело это сложное, трудоемкое, да к тому же небезопасное. Неверно, "хакнутая" прошивка может ко всем чертям угробить привод без малейшей надежды на его восстановление. Нет, уж лучше мы пойдем другим путем!

Идея восстановления информации, предлагаемая автором, в общих чертах сводится к записи на диск фиктивного TOC, адреса областей Lead-In и Lead-Out которого указывают на первый и последней сектор диска соответственно, а стартовый адрес первого трека аккурат совпадает с концом области Pre-gap, которая по стандарту должна занимать не менее 150 секторов (или 2 секунд в пересчете на абсолютные адреса). После этой нехитрой операции привод будет читать оригинальное содержимое очищенного диска как миленький, конечно, при том условии, что мы ухитримся настроить пишущий софт так, чтобы он, записав фиктивный TOC, никоим образом не пытался интерпретировать подсунутые ему указатели на области Lead-In/Lead-Out как указание "выжечь" всю поверхность диска целиком.

Проверка показывает, что CloneCD вообще не записывает такой TOC на диск, ругаясь на несоответствие размеров диска и образа файла. Alcohol 120% выполняет нашу просьбу без лишних препирательств, но совсем не так, как мы бы хотели! Забив весь восстанавливаемый диск непонятно откуда взятым "мусором", он авторитетно сообщает, что в процессе записи

произошли ошибки и, возможно, вам следует убедиться в исправности оборудования.

Хорошо, зайдем с другой стороны. Запишем на диск один реальный трек, занимающий минимально возможное количество секторов (по стандарту — 300, но некоторые проводы вполне удовлетворяются и меньшими значениями), но расширим его область Pre-gap с двух секунд на весь диск! В результате, мы потеряем лишь 300 последних секторов, но получим доступ ко всему остальному содержимому. Учитывая, что на диске этих секторов насчитывается немногим более 300 тысяч, нетрудно подсчитать, что процент успешно восстановленной информации составляет по меньшей мере 99,999% емкости всего диска, да и то лишь при том условии, что исходный диск был забит целиком, что в живой природе практически никогда не наблюдается. Если же это вас не удовлетворяет — разрабатывайте своей собственный софт, корректно записывающий фиктивный TOC, но ничего не делающий сверх этого (область Lead-In по любому записывает сам привод, ну а без области Lead-Out при аккуратном обращении с диском, в принципе, можно и обойтись, главное — пытаться прочитать сектора, находящиеся за пределами диска, иначе поведение привода станет трудно предсказуемым). Мне же так или иначе это делать лень, — с восстановлением полностью забитых дисков я еще не сталкивался. Во всяком случае, пока.

Процедура восстановления состоит из трех частей:

- подготовки исходного образа трека с нормальным значением области Pre-gap;
- увеличения области Pre-gap до размеров целого диска;
- записи исправленного образа на восстанавливаемый диск.

Первые два этапа достаточно выполнить всего один раз, т. к. полученный образ (далее мы будем называть его "лечебным") может использоваться для всех дисков (читай: *для всех дисков той же самой емкости*, по понятным соображениям вы не сможете корректно восстановить 23-минутный диск с помощью образа, предназначенного для 80-минутного диска и, соответственно, наоборот).

Для начала возьмем чистый CD-RW-диск ("чистый" не в смысле "ни разу не записанный", а очищенный быстрой или полной очисткой, так же для этих целей подойдет и CD-R). Используя любую утилиту для штатного "прожига", запишем на него один крошечный файл, "весящий" не более 500 Кбайт (более "тяжелый" файл просто не уместится в запланированные 300 секторов). Выполнять финализацию диска не нужно.

Запустим CloneCD (Alcohol 120%) и снимем образ диска. Спустя минуту-другую на винчестере образуются два файла: file name.img и file name.ccd (если вы попросили CloneCD сохранять так же и субканальную информацию, образуется третий файл — file name.sub, однако субканальная инфор-

мация в данном случае будет только мешать, потому опцию "чтение субканалов из треков с данными" лучше всего отключить или же просто удалить file name.sub с диска; также нам не нужен "Cue-Sheet", который CloneCD предлагает создавать для совместимости с другими программами, конкретно — с CDRWin).

Открыв файл file name.ccd любым текстовым редактором (например, "Блокнотом"), найдем в нем следующие строки (листинг 10.6) (ключевые слова для поиска `Point=0xa2` и `Point=0x01`).

Листинг 10.6. Оригинальный стартовый адрес Lead-Out (слева) и стартовый адрес первого трека диска (слева)

```
[Entry 2]                [Entry 3]     ; TOC entry
Session=1                Session=1     ; номер сессии
Point=0xa2               Point=0x01    ; point (A2h:leadout/01h:№ трека)
ADR=0x01                 ADR=0x01      ; в q-подканале данные позиционир.
Control=0x04             Control=0x04  ; трек с данными
TrackNo=0                TrackNo=0     ; Lead-In трек
AMin=0                   AMin=0        ; \
ASec=0                   ASec=0        ;  +- абсолютный адрес в M:S:F
AFrame=0                 AFrame=0      ; /
ALBA=-150                ALBA=-150     ; - абсолютный адрес в LBA [не исп.]
Zero=0                   Zero=0        ; зарезервировано
PMin=0                   PMin=0        ; \
PSec=29                  PSec=1        ;  + - относительный адрес в M:S:F
PFrame=33                PFrame=0      ; /
PLBA=2058                PLBA=0        ; - относительный адрес в LBA
```

Изменим поля PMin:PSec:PFrame, принадлежащие указателю (point) `A2h` так, чтобы они указывали на самый конец диска (`A2h` — это как раз и есть область Lead-Out). Измененное значение указателя на Lead-Out может выглядеть, например, так: `74:30:00`. Адрес Lead-Out следует выбирать с тем расчетом, чтобы между ним и внешней кромкой диска оставался по меньшей мере 30-секундный зазор. Еще лучше, если ширина области Lead-Out составит полторы минуты или около того. Однако в этом случае будут неизбежно теряться последние треки восстанавливаемого диска (если, конечно, вам действительно требуется их восстановить).

К содержимому полей PMin:PSec:PFrame, принадлежащих point'у `01h` (стартовый адрес первого трека), необходимо добавить ту же самую величину, которую вы добавили к соответствующим полям Lead-Out. Отредактированный вариант может выглядеть, например, так: `74:01:42`. (74:30:00 /* новый адрес Lead-Out */ − 00:29:33 /* старый Lead-Out */ + 00:01:00 /* старый

стартовый адрес первого трека */ == 74:01:42 /* новый стартовый адрес */.) Короче говоря, новая версия ccd-файла должна выглядеть так, как показано в листинге 10.7.

Листинг 10.7. Ключевой фрагмент "реаниматора" 75-минутных дисков CD-RW

```
PMin=74                    PMin=74
PSec=30                    PSec=01
PFrame=00                  PFrame=42
```

Вообще-то, для приличия следовало бы скорректировать и поля PLBA (LBA-адрес связан с абсолютным адресом следующим соотношением: `LBA == ((Min*60) + Sec)*75 + Frame`, однако текущие версии работают исключительно с абсолютными адресами и LBA-адреса игнорируют. Теперь все, что находится между концом Lead-In-области и началом первого сектора, и будет называться областью Pre-gap. При "прожиге" диска область Pre-gap остается нетронутой и позже может быть прочитана на секторном уровне (а это как раз то, что нам нужно!). Сказать по чести, чрезмерное увеличение Pre-gap первого трека — не самая лучшая идея, т. к. не все приводы способны читать такой "жирный" Pre-gap. С точки зрения совместимости, было бы лучше увеличивать Pre-gap *второго* трека, однако при этом первый трек придется располагать в самом начале диска и его тело неизбежно затрет восстанавливаемые сектора. И хотя это не такая уж большая проблема (в первых секторах диска все равно ничего ценного нет), к такой мере без особой необходимости все же лучше не прибегать. На крайний случай действуйте так: запишите на диск две сессии и вместо стартового адреса point'a `01h` меняйте стартовый адрес point'a `02h` (он будет находиться в разделе `session=2`).

Теперь наскоро очистим наш подопытный диск и до отвала забьем его какими-нибудь файлами (предпочтительнее всего использовать текстовые файлы — т. к. в этом случае будет сразу же видно: извлекается ли с восстановленного диска "мусор" или полезная информация). Записав файлы на диск, тут же выполним его быструю очистку.

Убедившись, что диск действительно очищен и его содержимое уже недоступно, запустим CloneCD и запишем только что созданный нами "лечебный" образ. Запись должна проводиться в режиме RAW DAO, иначе ничего хорошего у вас не получится (поэтому прежде чем восстанавливать скольнибудь ценный диск на еще не известном вам приводе, попробуйте потренироваться на "кошках", — диске, не содержащем ничего интересного).

Вот наконец мы держим в руках свежевосстановленный диск. Но действительно ли он восстановлен? А вот сейчас и убедимся! Вставляем "воскресшего из пепла" в привод NEC и с замиранием сердца пробуем прочитать один

из наугад взятых секторов из середины диска (начальные сектора обычно содержат нули, потом — файловую систему, и их очень легко принять за бессмысленный "мусор"). О чудо!!! Оригинальное содержимое очищенного диска читается как ни в чем не бывало!!! Правда, при попытке прочесть оглавление диска средствами операционной системы, привод может впасть в глухую задумчивость, граничащую с полным зависанием (ведь стартовый адрес первого трека расположен не в начале диска, а совсем в другом месте), но это все ерунда! Главное, что на секторном уровне диск все-таки доступен, пускай и не на всех приводах. Так, в частности, привод ASUS вообще отказывается читать такой диск, возвращая ошибку, а PHILIPS читает один мусор (к счастью, этот мусор можно восстановить, — достаточно просто на битовом уровне выполнить EFM-перекодировку с более "правильной" позиции. Поскольку возможных позиций всего 14, перебор обещает не затягиваться на длительное время. Тем не менее, лучше не извращаться, а просто приобрести более качественный привод).

Остается лишь привести диск в состояние, пригодное для "переваривания" операционной системой (что толку в работе с диском на низком уровне?). Последовательно читая все сектора диска один за один, мы будем собирать их в один IMG-файл, для определенности именуемый recover.img. Сектора, которые не удалось прочитать даже с нескольких попыток, мы будем просто пропускать. Теперь скопируем "лечебный" CCD-файл в recover.ccd и вернем стартовый адрес первого трека на прежнее место. Запишем сформированный образ диска на новую болванку, и (если все сделано правильно) любой привод должен читать ее правильно. Сеанс демонстрационного восстановления окончен, и мы, малость освоившись с этой технологией, можем приниматься за вещи куда как более серьезные. Например, откроем собственную компанию по восстановлению очищенных дисков. Шутка! Хотя почему бы и нет?

Хорошо, а как быть, если очищенный диск был многосессионным? Ведь описанные выше приемы рассчитаны на работу лишь с одной сессией! На самом деле, можно восстановить и многосессионный диск. Это лишь чуть-чуть труднее. Но, чтобы это сделать, мы должны предварительно познакомиться с остальными полями TOC. А это уже тема следующего раздела!

Постой, а если после очистки диска на него что-то писалось, — возможно ли тогда его восстановление или нет? Разумеется, непосредственно затертые места утеряны безвозвратно, но остальную часть информации по-прежнему можно спасти. Если диск до очистки был многосессионным, то нам даже не придется корпеть над восстановлением файловой системы, т. к. файловая система каждой последующей сессии обычно дублирует предыдущую ("обычно" это в смысле "за исключением удаленных файлов") и последняя сессия диска оказывается достаточно далеко от его начала, а потому и риск ее затирания — минимален (если, конечно, схватиться вовремя, а не тогда, когда весь диск перезаписан до отказа). Восстановление односессионных

дисков с затертой файловой системой — намного более трудная, но все-таки разрешимая задача. Во-первых, этих файловых систем на типовом диске целых две: ISO-9660 и Joliet, правда, в силу их близкого географического положения при затирании диска они обычно гибнут обе. Во-вторых, указанные файловые системы не поддерживают фрагментации и всякий файл, записанный на лазерный диск, представляет собой единый информационный блок. Все, что нужно для его восстановления — определить точку входа и длину. Точка входа в файл всегда совпадает с началом сектора, а подавляющее большинство типов файлов позволяют однозначно идентифицировать свой заголовок по уникальной сигнатуре (в частности, для ZIP-файлов характерна следующая последовательность: `50 4B 03 04`). Конец файла, правда, определяется уже не так однозначно, и единственная зацепка — структура самого восстанавливаемого файла. Впрочем, большинство приложений довольно лояльно относится к "мусору" в хвосте файла, и потому точность определения его длины с погрешностью в один сектор на практике оказывается вполне достаточной. Поскольку файлы располагаются на диске вплотную, без "зазоров", конечный сектор всякого файла надежно вычисляется путем вычитания единицы из стартового сектора следующего за ним файла.

Вообще же говоря, техника восстановления лазерных дисков намного проще и незатейливее искусства врачевания их прямых коллег — дискет и жестких дисков. Правда, поговорку "семь раз отмерь — один раз отрежь" еще никто не отменял, и одна из пренеприятнейших особенностей работы с CD-RW как раз и состоит в том, что вы не можете гарантированно управлять процессом записи. Дискеты и жесткие диски в этом смысле полностью прозрачны, — что вы пишете, то вы и получаете. Перезаписываемые же носители, напротив, представляют собой "черный ящик", и вы никогда не можете быть уверенными в том, что данный конкретный привод будет правильно интерпретировать отдаваемые ему команды (увы, восстановление дисков CD-RW никак не вписывается в рамки Стандарта, а все нестандартные махинации могут интерпретироваться приводом неоднозначно). Единственно, что остается посоветовать, — не пускайте все на самотек, а бесконечно экспериментируйте, экспериментируйте и еще раз экспериментируйте, накапливая бесценный опыт, который вам когда-то очень пригодится.

Как восстановить нечитающийся CD?

Bad sector не волк — в лес не убежит.

Народная мудрость

Лазерные диски — не слишком-то надежные носители информации. Даже при бережном обращении с ними вы не застрахованы от появления царапин и загрязнения поверхности (порой диск "фрезерует" непосредственно сам привод, и вы бессильны этому противостоять). Но даже вполне нормальный

на вид диск может содержать внутренние дефекты, приводящие к его полной или частичной нечитаемости на штатных приводах. Особенно это актуально для CD-R/CD-RW-дисков, качество изготовления которых все еще оставляет желать лучшего, а процесс записи сопряжен с появлением различного рода ошибок.

Однако даже при наличии физических разрушений поверхности лазерный диск может вполне нормально читаться за счет огромной избыточности хранящихся на нем данных, но затем, по мере разрастания дефектов, корректирующей способности кодов Рида-Соломона неожиданно перестает хватать и диск без всяких видимых причин отказывает читаться, а то и вовсе не опознается приводом.

К счастью, в подавляющем большинстве случаев хранимую на диске информацию все еще можно спасти, и этот раздел рассказывает как.

Общие рекомендации по восстановлению

Не всякий нечитающийся (не стабильно читающийся) диск — дефектный. Зачастую в этом виновен отнюдь не сам диск, а операционная система или привод. Прежде чем делать какие-либо заключения, попробуйте прочесть диск на всех доступных вам приводах, установленных на компьютерах с "девственно" чистой операционной системой. Многие приводы, даже вполне фирменные и дорогие (например, мой PHILIPS CD-RW 2400), после непродолжительной эксплуатации становятся крайне "капризными и раздражительными", отказывая в чтении тем дискам, которые все остальные приводы читают безо всяких проблем. А операционная система по мере обрастания свежим софтом склонна подхватывать различные "глюки", подчас проявляющиеся самым загадочным образом (в частности, привод TEAC установленный в систему с драйвером CDR4_2K.SYS, доставшийся в наследство от привода PHILIPS, конфликтует с CD-плеером, не соглашаясь отображать содержимое дисков с данными, если тот активен, после же удаления CDR4_2K.SYS все идет как по маслу).

Также не стоит забывать и о том, что *корректирующая способность* различных моделей приводов очень и очень неодинакова. Как пишет инженер-исследователь фирмы ЕПОС Павел Хлызов в своей статье "Проблема: неисправный CD-ROM": *"в зависимости от выбранной для конкретной модели CD-ROM стратегии коррекции ошибок и, соответственно, сложности процессора и устройства в целом, на практике тот или иной CD-ROM может либо исправлять одну-две мелкие ошибки в кадре информации (что соответствует дешевым моделям), либо в несколько этапов восстанавливать, с вероятностью 99,99%, серьезные и длинные разрушения информации. Как правило, такими корректорами ошибок оснащены дорогостоящие модели CD-ROM. Это и есть ответ на часто задаваемый вопрос: "Почему вот этот диск читается на машине товарища, а мой ПК его даже не видит?"*

Вообще-то, не совсем понятно, что конкретно господином инженером-исследователем имелось в виду: корректирующие коды C_1, C_2, Q- и P-уровней корректно восстанавливают все известные мне приводы, и их корректирующая способность равна: до 2 ошибок на каждый из C_1 и C_2 уровней и до 86 и 52 ошибок на Q- и P-уровни соответственно. Правда, количество обнаруживаемых, но уже математически не исправимых ошибок составляет до 4 ошибок на C_1 и C_2 уровней и до 172/104 ошибок на Q/P, но гарантированно определяется лишь позиция сбойных байт во фрейме/секторе, но не их значение. Впрочем, зная позицию сбойных байт и имея в своем распоряжении исходный HF-сигнал (т. е. *аналоговый* сигнал, снятый непосредственно со считывающей головки), кое-какие крохи информации можно и вытянуть, по крайней мере теоретически, так что приведенная ранее цитата в принципе *может быть* и верна, однако, по наблюдениям автора данной статьи, цена привода очень слабо коррелирует с его "читабельной" способностью. Так, относительно дешевые приводы ASUS читают практически все, а дорогие приводы PHILIPS даже свои родные диски с драйверами опознают через раз.

Другая немаловажная характеристика — *доступный диапазон скоростей* чтения. В общем случае — чем ниже скорость вращения диска, тем мягче требования, предъявляемые к его качеству. Правда, зависимость эта не всегда линейна. Большинство приводов имеют одну или несколько наиболее *предпочтительных* скоростей вращения, на которых их читабельная способность максимальна. Например, на скорости 8x дефектный диск читается на ура, а на всех остальных скоростях (скажем, 2x, 4x, 16x, 32x) — не читается вообще. Предпочтительная скорость легко определяется экспериментально, необходимо лишь перебрать полный диапазон доступных скоростей.

При покупке привода CD-ROM выбирайте тот, у которого скоростной диапазон максимален. Например, уже упомянутый выше PHILIPS CDRW 2400 умеет работать лишь на: 16x, 24x, 38x и 42x. Отсутствие скоростей порядка 4x—8x ограничивает "рацион" привода только высококачественными дисками.

По непонятным причинам, штатные средства операционной системы Windows не позволяют управлять скоростью диска, и потому приходится прибегать к помощи сторонних утилит, на недостаток которых, впрочем, жаловаться не приходится. Вы можете использовать утилитой *Slow CD*, *Ahead Nero Drive Speed* и т. д. Вообще-то, большинство приводов самостоятельно снижают скорость, натолкнувшись на нечитающиеся сектора, однако качество заложенных в них алгоритмов все еще оставляет желать лучшего, и "ручное" управление скоростью дает значительно лучший результат.

Если же ни на одном из доступных вам приводов диск все равно не читается, можно попробовать отшлифовать его какой-нибудь полировальной пастой. Технике полирования оптических поверхностей (и лазерных дисков в частности) посвящено огромное количество статей, опубликованных как в печатных изданиях, так и в Интернете (особенно полезны в этом смысле

астрономические книги по телескопостроению), поэтому здесь этот вопрос будет рассмотрен лишь кратко. Да, действительно, поцарапанный диск в большинстве случав можно отполировать, и если все сделать правильно, диск с высокой степенью вероятности возвратится из небытия. Во-первых, полировка восстанавливает лишь царапины нижней поверхности диска и бессильна противостоять разрушениям отражающего слоя. Во-вторых, устраняя одни царапины, вы неизбежно добавляете другие, и после иной полировки лазерному диску может очень сильно "поплохеть". В-третьих, полировке дисков невозможно научиться за раз, — вам понадобится уйма времени и куча "подопытных" дисков. Нет уж, благодарю покорно! Лучше мы пойдем другим путем!

А вот что вашему диску действительно не помешает — так это протирка обычными салфетками, пропитанными антистатиком (ищите их в компьютерных магазинах). Прежде чем вытирать диск, сдуйте все частицы пыли, осевшие на него (иначе вы его только больше поцарапаете) и ни в коем случае не двигайтесь концентрическими мазками! Вытирать поверхность диска следует радиальными движениями от центра к краям, заменяя салфетку на каждом проходе.

Диск не опознается приводом

Вы вставляете диск в привод. Привод раскручивает диск, судорожно мигая при этом индикатором активности, затем, убедившись в том, что на заданной скорости диск не читается, начинает снижать обороты вплоть до полной остановки диска. Индикатор "DISK IN" (если он присутствует на лицевой панели привода) печально тухнет, давая тем самым понять, что кусок пластика, засунутый в привод, с точки зрения привода, представляет собой все что угодно, но только не компакт-диск. При попытке обращения к диску выдается сообщение об отсутствии диска в дисководе и вежливое предложение его туда вставить.

Неспособность привода опознать диск в подавляющем большинстве случаев — это свидетельство неисправности CD-ROM-привода. Реже — дефективности самого лазерного диска. Даже если вчера этот диск вполне уверенно опознавался и даже если привод опознает все остальные диски — не спешите уверять себя в его, привода, работоспособности! Попробуйте прочитать диск на другом накопителе. На худой конец — уменьшите скорость вращения диска до минимальной, однако будьте готовы к тому, что привод вас не послушается. Дело в том, что большинство приводов автоматически сбрасывают прежние установки скорости при смене диска и не позволяют изменять скорость вплоть до тех пор, пока диск не будет опознан (особенно этим "славятся" приводы TEAC, приводы от ASUS обычно ведут себя более демократично).

Если же подопытный диск отказывается опознаваться всеми доступными вам приводами, то причина скорее всего в том, что приводы не могут прочесть

оглавление диска (также называемое *TOC*), хранящееся в области Lead-In. Выньте диск из привода и внимательно рассмотрите узкое блестящее кольцо, расположенное у внутреннего края диска — это и есть Lead-In. Нет ли на нем глубоких царапин или загрязнений? Загрязнения удалите чистой салфеткой (к слову сказать, при очистке диска про вводную область зачастую как-то забывают, вероятно, принимая ее за бесполезное декоративное украшение). Бороться с царапинами намного труднее и без надлежащего опыта полировки лазерных дисков за это дело лучше не браться. Лучше всего было бы отнести такой диск в сервисный центр, специализирующийся на восстановлении информации, однако далеко не во всяком городе такие центры вообще есть и далеко не всегда они выполняют такое восстановление оперативно и грамотно. Опять-таки: конфиденциальность, стоимость восстановления и прочее, прочее, прочее.

Можно ли восстановить такой диск самостоятельно? Да, можно, но для этого вам понадобится определенное оборудование, стоящее около 1000 рублей (~30$). Конкретно — отдельный CD-ROM, над которым будет не жалко поизмываться и потерей которого вы окажетесь не слишком сильно огорчены (очень хорошо подходят для этих целей низкоскоростные приводы, оставшиеся от последнего апгрейда системы).

Весь фокус в том, что для работы с диском на секторном уровне TOC не так уж и нужен и без него вполне можно обойтись. Фактически это не аппаратная, а программная проблема. Обнаружив, что в процессе чтения оглавления диска возникли неустранимые ошибки, микропрограмма, зашитая в ПЗУ привода, отказывает такому диску в обработке, несмотря на то, что содержимое TOC дублировано в Q-канале подкода и размазано по всей спиральной дорожке. Причем привод реально нуждается лишь в трех основных полях TOC: *адресе выводной области диска* (чтобы знать: до сих пор можно "дергать" головкой), *стартовом адресе первого трека* (чтобы знать, откуда начинать чтение данных) и *адресе следующей вводной области* (только для многосессионных приводов). Со стартовым адресом первого трека разобраться проще всего — он по жизни равен 00:02:00 (что соответствует нулевому LBA-адресу). Адрес Lead-Out, напрямую зависящий от объема лазерного диска, не обязательно указывать точно, достаточно выбрать его таким, чтобы он был не меньше адреса настоящего Lead-Out, иначе все, расположенные за ним сектора окажутся недоступными. Установив адрес Lead-Out на 80 или даже 90 минут, мы можем гарантировать, что вся поверхность диска будет доступна приводу. Короче говоря, имей мы доступ ко внутренним структурам прошивки привода, восстановление разрушенного TOC было бы плевым делом. Автор использует для этих целей специальным образом модифицированную им прошивку обыкновенного CD-ROM (старенькая 8x модель от no name), которая позволяет манипулировать любыми служебными данными и потому читает все, что только физически можно прочесть.

Если же "взлом" микропроцессорных программ вам не по зубам, можно пойти другим путем. Аккуратно разберите привод CD-ROM и извлеките его

начинку из корпуса (теперь вы поняли, почему автор порекомендовал купить для этих целей отдельный — максимально дешевый — привод?). Теперь открутите болты, удерживающие металлическую планку, на которой закреплен эдакий "пятачок", прижимающийся к верхнему краю лазерного диска и тем самым уберегающий его от проскальзывания. Вместо этой нехитрой конструкции вы можете использовать металлическое кольцо или иную тяжесть. Главное — получить свободный доступ к лазерному диску и возможность его "горячей" смены на ходу *без* выдвижения лотка.

Подключите привод к компьютеру и, включив питание последнего, нормальным путем вставьте в привод специальным образом подготовленный диск, адрес выводной области которого лежит в районе 80—90 минут (можно просто вставить любой компакт-диск с видеофильмом от 700 Мбайт). Убедившись, что диск нормально опознан, дождитесь его полной остановки и, не выключая компьютера, аккуратно снимите его с привода, ни в коем случае не открывая лоток. Теперь установите в привод тот диск, который вы собираетесь восстанавливать. Поскольку TOC старого диска уже находится в кэше, а замену диска, совершенную таким варварским способом, привод обнаружить не в состоянии, он будет работать с новым диском точно так же, как и со старым. Только не пытайтесь читать содержимое диска средствами операционной системы — это ни к чему хорошему ни приведет (ведь она тоже умеет кэшировать и сколько бы вы ни жали на "обновить", в окне проводника будет неизменно прежнее оглавление). Лучше возьмите любой "грабер", читающий диск на секторном уровне и не задающий при этом лишних вопросов (например, можно воспользоваться утилитой cd_raw_read, бесплатно распространяемой автором), и скопируйте все содержимое диска от первого сектора до последнего в файл-образ, а затем, используя любую подходящую программу "прожига", "залейте" его на CD-R или CD-RW. Пусть вы не восстановите сам диск, но зато — его содержимое! Эта методика с одинаковым успехом применима как для аудиодисков, так и для дисков с данными.

Как вариант: можно не откручивать прижимную планку, а найти датчик смены диска и на время сделать ему "харакири", заставляя привод "думать", что восстанавливаемый диск не был заменен (дешевые приводы используют простые механические датчики, сразу же бросающиеся в глаза, в более дорогих моделях отдельного датчика вообще нет, и признаком смены диска считается нажатие кнопки Eject; в этом случае с некоторым риском можно воспользоваться отверстием для аварийного извлечения диска, однако имейте в виду, что извлечение диска на работающем приводе может необратимо искалечить его механическую часть).

К слову сказать, существуют и такие приводы, которые ухитряются читать диск даже при полностью разрушенном TOC'е. К ним, в частности, относятся некоторые модели пишущих устройств от фирмы MSI. Обладателям этих приводов незачем развинчивать свой CD-ROM — сбойный диск он прочтет и так.

Также, при восстановлении многосессионных дисков можно попробовать просто зачернить вводную область диска черным маркером, — содержимое первой сессии при этом окажется утраченным, но вот все последующие сессии большинство приводов прочтут на ура. Напоминаю, что вводная область диска выглядит как блестящее кольцо, расположенное вокруг внутренней кромки диска.

Диск опознается приводом, но не опознается операционной системой

Вы вставляете диск в привод. Привод раскручивает диск, зажигает индикатор DISK IN (если он есть), однако попытка просмотра содержимого диска штатными средствами операционной системы приводит к сообщению о той или иной ошибке. Сканирование поверхности диска утилитой Ahead Nero CD Speed (или любой другой утилитой аналогичного назначения) выявляет один или несколько разрушенных (damaged) секторов.

Это — явный симптом повреждения файловой системы, а точнее — ее корневого каталога. Если это произошло — не хватайтесь за сердце. Восстановление корневого каталога лазерных дисков в отличие от винчестеров и дискет не представляет большой проблемы. Подавляющее большинство лазерных дисков содержат не одну, а сразу **две** файловых системы, дублирующих друг друга — ISO-9660 и Joliet (таковыми являются все диски, выпущенные после 1995 года). Согласитесь, одновременное разрушение сразу двух корневых каталогов — событие крайне маловероятное. К тому же, в силу отсутствия фрагментации, вложенные подкаталоги не разбросаны по всей поверхности лазерного диска, а сосредоточенны в одном месте, благодаря чему даже при полностью разрушенном корневом каталоге их достаточно легко восстановить. Наконец, каждая последующая сессия многосессионого диска включает в себя содержимое файловых систем всех предыдущих сессий (исключая, разумеется, удаленные файлы). А потому, при "смерти" файловой системы последней сессии, мы без труда можем спасти содержимое всех остальных.

К сожалению, штатные средства Windows не предоставляют возможности выборочного монтирования ни предпочтительной файловой системы, ни предпочтительной сессии, принудительно "подсаживая" нас на корневой каталог Joliet последней сессии диска. Самое простое, что можно сделать , — попробовать прочитать диск под "голой" MS-DOS с установленным драйвером MSCDEX, работающим исключительно с системой ISO-9660 и игнорирующим существование Joliet. Как вариант, вы можете воспользоваться утилитой ISO 9660.dir, разработанной автором специально для работы с порушенными файловыми системами и восстанавливающей все, что только можно восстановить.

Естественно, в силу того, что максимальная длина файловых идентификаторов в системе ISO-9660 составляет всего лишь 11 символов, длинные файловые имена оказываются необратимо искажены, однако, согласитесь, это все же лучше, чем совсем ничего.

При вставке диска в привод компьютер "зависает"

Вы вставляете диск в привод, привод раскручивает диск, интенсивно мигая индикатором активности, и зависает, зачастую "подвешивая" вместе с собой и операционную систему. В легких случаях положение спасает кнопка Eject, в тяжелых — Reset.

Такое поведение характерно для защищенных дисков, защита которых основана на искаженном TOC'е. Большинство приводов к искаженному TOC'у относятся довольно лояльно (хотя это смотря еще что искажать), но встречаются и такие, которые при этом просто виснут. Если прочесть защищенный диск все же необходимо — попробуйте сменить привод.

Другой возможный вариант — зацикленная файловая система. При "прожиге" дисков CD-R/CD-RW "кривым" софтом — такое часто случается. Удерживая клавишу <Shift> во время загрузки диска, запретите операционной системе читать его содержимое (или же просто временно отключите автозапуск) и посредством той же утилиты ISO 9660.dir вытяните из диска все, что только с него можно вытянуть.

Диск читается с ошибками

Если, несмотря на все ухищрения типа снижения скорости или очистки поверхности, диск все равно читается с ошибками и сбойные сектора приходятся как раз на область, занятую ценнейшими файлами, — "дело труба". Но все же, шансы успешного восстановления данных есть, пускай и небольшие.

Прежде всего: ошибка ошибке рознь. Редко бывает так, чтобы сектор не читался весь целиком. Как правило, речь идет об искажении одного или нескольких принадлежащих ему байт. Причем корректирующая способность избыточных кодов такова, что до 392 сбойных байт исправляется уже в декодере первого уровня (CIRC-декодере). Еще до 86 ошибок способны исправлять P-коды и до 52 ошибок — Q-коды. Таким образом при наиболее благоприятном распределении ошибок удается восстановить вплоть до 530 ошибок или до ~25% общей емкости сектора. Лишь чудовищная ненадежность оптических носителей приводит к тому, что даже такая колоссальная избыточность данных иной раз не в силах противостоять сбоям.

В зависимости от установочных параметров, накопитель, обнаружив неустранимый сбой, либо отдает сектор в том виде, в котором его удалось прочесть, либо же просто рапортует об ошибке, оставляя содержимое выходного буфера в неопределенном состоянии. Идея восстановления состоит в том, чтобы заставить привод выдавать все, что он только способен прочесть. Конечно, искаженные байты уже не вернуть назад, однако многие форматы файлов вполне лояльно относятся к небольшим разрушениям. Музыка в формате MP3/WMA, видеофильмы, графические изображения — все

они будут вполне нормально воспроизводиться, и только непосредственно на месте самого искажения возникнет щелчок той или иной громкости или мелькнет "артефакт". С архивами ситуация обстоит значительно хуже, однако в подавляющем большинстве случаев вы потеряете всего один-единственный файл, а все остальное содержимое архива распакуется нормально (кстати, некоторые архиваторы, такие например, как RAR, поддерживают собственные корректирующие коды, позволяющие при минимальной избыточности восстанавливать "битые" архивы).

"Постойте! — возразят мне иные читатели. — Как же, было дело! Пробовали мы восстанавливать нечитающиеся диски теми или иными утилитами. Ну и что? "Вылеченный" mpg- или avi-файл система наотрез отказалась считать видеофайлом!" "Так все дело в том, — резонно возражу я, — что эти самые утилиты просто выкидывали все сектора, которые они не могли прочесть, в результате чего размер файла, а значит, и относительные смещения всех его структур изменились! Неудивительно, что после такой "кастрации" он перестал воспроизводиться!"

Воспользуйтесь любым копировщиком защищенных дисков, предоставляющим выборочное управление режимом обработки ошибок, и выберите режим 24h (максимально возможная коррекция ошибок без прерывания передачи данных в случае невозможности их восстановления). Среди прочих утилит для этой цели подойдет тот же cd_raw_read, разработанный автором. Как альтернативный вариант вы можете использовать Alcohol 120% и/или CloneCD.

Спрашиваете: а по каким причинам сектор может перестать читаться? Прежде всего это глубокие и широкие радиальные царапины со стороны верхней части. Преодолев тонкий барьер защитного лакового слоя, царапины "выедают" непосредственно сам отражающий материал, а вместе с ним — и полезные данные.

Немногочисленные *узкие царапины* в общем-то не опасны — содержимое сектора размазано вдоль спиральной дорожки и потому выпадения нескольких байт легко компенсируются за счет избыточности. Правда, тут есть одно "но". Откуда приводу знать, сколько именно "питов" и "лендов" было пропущено? Поскольку "питы" и "ленды" напрямую не соответствуют двоичному нулю и единице, и единица кодируется переходом от "пита" к "ленду" или наоборот, а нуль — отсутствием переходов на данном участке, становится понятно, что пропадание нечетного числа "питов"/"лендов" как бы переворачивает весь хвост фрейма с ног на голову, т. е., другими словами, "гробит" его. Отсюда: даже одна-единственная царапина способна породить целый каскад ошибок, не устранимых штатными корректирующими кодами, но в принципе, поддающимися ремонту вручную. Ну, не то, чтобы совсем вручную, — необходимая для этой цели утилита уже написана автором и сейчас проходит стадию альфа-тестирования, на ура читая те диски, которые не читались нормальным путем. Не исключено, что к моменту выхода

данной книги она перейдет в стадию бета-тестирования и станет бесплатно доступна всем желающим. Впрочем, поскольку длина одного фрейма составляет всего 24 байта, разрушение нескольких подряд идущих фреймов может быть реконструировано и штатными корректирующими кодами и к помощи моей утилиты придется прибегать лишь на сильно поцарапанных дисках.

Широкие царапины — другое дело. Мало того, что они "съедают" несколько фреймов целиком, так они еще и оптическую головку сбивают с дорожки. Попав в образованную царапиной дыру, головка совершенно дезориентируется (ей становится попросту не на что "опираться"!) и "вылетает" в одну из соседних дорожек. Умные приводы автоматически распознают такую ситуацию и позиционируют головку на нужное место. Приводы поглупее (коих, кстати, подавляющее большинство) самоуверенно продолжают чтение как ни в чем ни бывало. В результате, голова одного сектора скрещивается с хвостом другого, и естественно, при попытке восстановления такого сектора штатными корректирующими кодами ничего, кроме "мусора", не получается и привод уныло диагностирует неисправимую ошибку. Выход — читать такой сектор до тех пор, пока головка не попадает на ту же самую дорожку, с которой начиналось чтение сектора. Количество попыток чтения при этом должно быть достаточно велико (от 100 и больше). Ведь с точки зрения вероятности отклониться от спиральной дорожки намного проще, чем удержаться на ней!

Концентрические царапины — самый деструктивный тип разрушений, который только может быть. Размазывание информации вдоль спиральной дорожки теперь не в силах противостоять сбою, поскольку искажение затрагивает весь сектор целиком (радиальные царапины, напротив, искажают лишь небольшую часть сектора). К тому же концентрические царапины сбивают систему слежения, поскольку следящие лазерные лучи слегка расфокусированы и потому оказываются весьма чувствительны к подобным дефектам поверхности.

Царапины, расположенные с нижней стороны диска, в большинстве случав устраняются полировкой, а вот царапины, "высверлившие" рабочий слой, ликвидировать — увы — невозможно.

Описание компакт-диска

Компакт-диск, прилагаемый к этой книге, содержит все утилиты, необходимые для анализа и копирования защищенных компакт-дисков. Хотелось бы еще раз обратить внимание читателя на то, что обход защиты CD от копирования — это совсем не то же самое, что нарушение авторских прав! Так, например, действующее законодательство многих стран явным образом предписывает потребителям выполнять резервное копирование лицензированных носителей информации. Еще один пример — корпорация PHILIPS, являющаяся одним из основоположников технологии записи на компакт-диски, является ярым противником любых отклонений от стандарта и настаивает на том, что диски, защищенные от копирования при помощи любого из механизмов, основанных на отклонениях от стандарта, не должны снабжаться логотипом "Compact Disc". Опять же, надо отметить, что действующее законодательство многих стран поддерживает это требование. Таким образом, любой диск, защищенный от копирования нестандартным форматом, должен иметь соответствующую маркировку, явным и недвусмысленным образом предупреждающую потенциальных потребителей о том, что, несмотря на внешнее сходство, кусок пластика, который они собираются приобрести, настоящим компакт-диском не является.

Что же касается разработки механизмов защиты CD, программисты, берущиеся за эту работу, должны хотя бы иметь представление о методах работы взломщиков и арсенале технических средств, имеющихся в их распоряжении. Будет еще лучше, если разработчики защит овладеют этим инструментарием как минимум на таком же уровне, что и хакеры. Это позволит вам быстро обнаружить наиболее слабые звенья защитного механизма, которые являются наиболее вероятной мишенью для атаки. Обнаружив такие слабости, их следует ликвидировать, направив на решение этой задачи максимум доступных интеллектуальных ресурсов. Попросту говоря, разработчик защит должен проникнуться хакерской психологией и начать думать как хакер! В конце концов, если вы не знаете, как работают защитные механизмы, в чем заключаются достоинства и недостатки каждого из них, то вы не сможете создать достаточно сильный защитный механизм, который был бы устойчив ко взлому и одновременно недорог и прост в реализации.

Каталог SRC данного компакт-диска содержит исходный код всех демонстрационных примеров, приведенных в данной книге. Здесь собраны многие небольшие, но очень полезные утилиты, в том числе копировщик Pinch of File, позволяющий выполнять пофайловое копирование содержимого CD, включая файлы с некорректными длинами и неправильными стартовыми секторами. Подкаталоги каталога SRC содержат следующую информацию:

- ETC — демонстрационные примеры, иллюстрирующие низкоуровневый доступ к приводам CD-ROM;
- RS.LIB — библиотеки CloneCD и Ahead Nero для низкоуровневой работы с секторами CD, интерфейсы к этим библиотекам, а также практические примеры, иллюстрирующие их использование;
- RS.SIMPLE — элементарные примеры, иллюстрирующие принципы работы кодов Рида-Соломона;
- SCSI.ALT — исходный код драйвера, позволяющего исполнять машинные команды IN/OUT с прикладного уровня;
- SCS.LIB — различные утилиты, разработанные автором книги для работы с защищенными CD.

В каталоге DOC компакт-диска располагаются описания стандартов и спецификаций, а каталог Image содержит файлы цветных и черно-белых рисунков книги.

Предметный указатель

A

Adaptec 183, 184
Additional Sense Code 171
Additional Sense Code Qualifier 171
Ahead Nero 242
Ahead Nero Burning ROM 246
Alcohol 120% 4, 242
AMD 143
API 242
API-функция 151, 180, 404, 405
ASPI 152, 187, 224, 404
ATA/ATAPI 204
ATAPI 176
ATIP 10, 400

B

BSOD 109, 137, 141, 146
Buffer overfull 126
Bug Check-параметр 138,
  142—144, 148

C

CCD-файл 343, 346, 349, 352,
  374, 376, 432
  создание 344
CDB 165
CDB-блоки 165
CD-Enhanced 372
CDFS-драйвер 405
CD-плеер 371, 434
CIRC-декодер 440
CIRC-кодер 42
CloneCD 4, 32, 242
Cooked-Mode 160
CRC 17

D

DDCD 359
Directory Record 421
DDK 136, 137, 146, 153, 164
DLL 187
DMP-файлы 146
DPMI 222
DPMI-функция 405
Driver Development Kit 136
DSV 28, 40, 415, 416, 417
DumpChk 146

E

EFM 16
EFM-перекодировка 432
EFM-слово 16
ElbyECC.dll 32, 356
Error Flag(s) 177

F

F1-фрейм 39, 48
FAR 229, 344
FAT 379
Flag Bits 177
Floppy 243
FPO-функции 123
FSCTL/IOCTL-команды 152

G

GNU 53
GUI 127, 187

H

HBA 200
Header(s) code 177
HEX-редактор 148, 368, 380, 393, 394, 396, 414
HF-сигнал 357, 435

I

IBM 79
IBM PC 62, 166, 382
IDA Pro 12
IMG-файл 343, 432
Intel 144
Interrupt List 221
IOCTL 210, 224, 405
IOCTL-код 168, 184
IOCTL-команды 153, 154
IOPM 210
IRP 140
IRP-запросы 179
IRQ 139
ISO 9660 420

J

Joliet 420, 439

K

KAnalyze 146

L

LBA 22, 176, 380
LBA-адрес 255
Lead-In 49, 276
Lead-Out 50, 376

M

Mac 373
MBR 195
Merging-бит 25, 415
MFM 245
Microsoft 164
Microsoft Kernel Debugger 118
Microsoft Knowledge Base 405
Microsoft Visual Studio 118, 127, 405
Microsoft Visual Studio Debugger 112, 118, 120, 123, 132
MOD 242
MP3, 373
MP3-плеер 384, 389, 396, 423
MS WDB 12
MSCDEX 152
MSDN 164, 187, 405
MS-DOS 183, 196, 221, 243, 270, 381, 403
MSF 22, 294
MSI 438

N

Norton Utilities 119
NTDDK 138

P

Platform SDK 164, 376
PMA 49
Pool-request 144
Post-gap 259, 270, 277
Pre-gap 259, 267, 270, 276
  extended 277
Program area 51
PTE 141

Q

Q-канал 247, 249, 261, 437
  поле CONTROL 246
  поле INDEX 277

R

RAID 81
  программный 82
RAW 404
RealAudio 393
Red Book 22
Regedit 111
Roxio Easy CD Creator 242
Run-in/Run-out 359

S

SCSI 165
SCSI-контроллер 143
SCSI-порт 195
SDK 187
Sense Info 171, 367
Sense Key 171, 172, 367
SMART 203
Soft-Ice 228
SP2, 144
SPTI 152, 168, 404
SRB 166
Sub-Channel Data Selection Bits 178
SUB-файл 343
SYNC0 48
SYNC1 48
Synch Field 177

T

TAO 278
TDB 10, 278
TDI 280
TDT 279
TDU 279, 280
TOC 50, 151, 246, 249, 259, 261, 270, 271, 275, 426, 432, 437, 438
  искажение 246—248
  основные поля 437
  чтение 158, 260
Transition area 277
TSS 213, 214
Twin-sectors 10

U

UNIX 373
User Data 24

W

Windows 73, 373, 381, 382, 404, 419, 421, 423, 435, 439
Windows 2000, 245, 353, 418
Windows 2000, RUS 134
Windows 3.1, 221
Windows 95, 221
Windows 9x 110, 151, 183, 225, 418
Windows 9x/NT 404
Windows NT 110, 143, 151, 160, 196, 214, 225, 403—405, 418
Windows NT 4.0, ENG 134
Windows XP 245
WM_DESTROY 376
Word 73

X

X-сектор 295, 360, 367
  копирование 366
  понятие 366

Y

Yellow Book 22

Z

Zip-накопитель 414
ZIP-файл 433
ZX-Spectrum 353

А

Адрес:
LBA 255
абсолютный 255
порта (относительный) 211
Алгоритм:
Берлекэмпа-Месси 88, 92
Форни 88, 92
Ченя 88
Альфа-тестирование 441
Аргумент:
this 130
Арифметика:
полиномиальная 68
Аудиопроигрыватель 371, 376
Аудиосессии 371
Аудиотреки 350, 371, 375

Б

Байт:
контрольный 48
субканала 17
субкода 17
терминатор 213
управляющий 17
чередование 81
"Белый шум" 30
Бета-тестирование 441
Библиотека:
ElByECC.DLL 83, 100
MFC 130
MFC42.DLL 131, 132
MFC42.LIB 132
NEWTRF.DLL 83
NTOSKRNL 214
OWL 130
Wnaspi.dll 186
Wnaspi32.dll 184, 186, 231
обертка 184
Биты:
запрещенные комбинации 55
информационные 57
контрольные 57
объединяющие 16, 25
Блок 17, 351
SRB 179
вбега/выбега 359
Буфер:
sense_buf 171

В

Взлом 373
Врапперы 110

Г

Генератор:
псевдослучайных чисел 397
случайных чисел 397
Голубой экран смерти 109, 135, 137, 138, 144—148
Группы Галуа 71

Д

Дамп памяти 113, 134
малый 135
полный 135, 136, 145
ядра 135, 145
Данные 17
Декодер:
Рида-Соломона 31
Дескремблирование 30
Дизассеблер 414
IDA Pro 34, 128, 150, 392
Диск:
CD-DA 153
CD-DATA 153
Imation 407, 414
TDK 414
логический 419
многосессионный 51, 285, 371, 418, 439
нестандартная разметка 243
оглавление 50, 275
трудновоспроизводимые дефекты 243

Доктор Ватсон 111, 112
Дорожка 8, 49
Доступный диапазон
  скоростей 435
Драйвер 151
  APIX.VxD 186
  APIX.VXD 184
  ASPI.SYS 184, 195
  ASPI32.SYS 229
  ATAPI.SYS 197, 200, 201, 203,
    205, 207
  AXSAKI.SYS 197
  BDOS.EXE 136
  CDFS 420
  CDFS.VxD 186
  CDR4_2K.SYS 434
  CDROM.SYS 195
  CDVSD.VxD 186
  DISK.SYS 200, 201
  ESDI_506.PDR 186
  MSCDEX 183, 221, 222, 439
  NTFS.SYS 140
  PORTIO 210
  W2K_KILL.SYS 136
  низкоуровневый 183
  убийца 136, 137

Ж

"Желтая книга" 22, 361

З

Заголовок 17
  поле адреса 17
  поле режима 17
  поле синхронизации 17
Запись:
  произвольная 359
Защита:
  аудиодисков 376
  диска
    от проигрывания 244
  нейтрализация 274
  типа "Волк" 300
  типа "Лиса" 370
  типа "Шакал" 292
  уровня файловой
    системы 378
Защитный слой 15

И

Идентификатор:
  KeBugCheckEx 146
  PspUnhandledException
    InSystemThread 146
Имитация ключевых
  особенностей 242
Индекс полинома 74
Индикатор:
  "DISK IN" 436, 439
Интерполяция 6
Интерфейс:
  ASPI 110, 183, 275
  SPTI 110, 183, 275

К

Канал:
  Q 19
  P 19
  подкода 10
  субкода 18
    секции 18
Карта разрешения
  ввода/вывода 213
Клавиша:
  <F8> 145
  <Shift> 440
Класс CwinThread 130—133
Классификация защит 241
  критерии 241
Кнопка:
  Eject 234, 247, 308, 325, 426,
    438, 440
  Reset 440
  выброса диска 407

Код 54
Bug Check 137, 138
ECC/EDC 19, 161
EDC & ECC 177
Боуза-Чоудхури-Хоквингема 66
основание 54
ошибки:
ASC 171, 172
ASCQ 171, 172
Sense Key 171, 172
Рида-Соломона 60, 62, 63, 66, 68, 351
Хемминга 57
эффективность 59
Кодер/декодер:
Рида-Соломона 63, 65, 70, 72, 74, 79
Кодирование:
несистематическое 84
систематическое 62, 84
Кодировщик:
несистематический 84
систематический 84
Кодовые:
комбинации 54
символы 54
слова 54
Кодовый алфавит 54
Команда:
BLANK 232
bpx SendASPI32Command 231
DIR 382, 422
GET CONFIGURATION 247
GET PERFORMANCE 232
IN 212
INQUIRY 189
IOCTL_CDROM_GET_LAST_SESSION 153
IOCTL_CDROM_MEDIA_REMOVAL 234
IOCTL_CDROM_RAW_READ 153
IOCTL_CDROM_READ_Q_CHANNEL 153
IOCTL_CDROM_READ_TOC 153
IOCTL_SCSI_GET_CAPABILITIES 169
IOCTL_SCSI_GET_INQUIRY_DATA 196
IOCTL_SCSI_MINIPORT_IDENTIFY 203
IRP_MJ_READ 153
mod 229
MODE SENSE 10, 46
Modules 131
OUT 212
P RET 161, 229
POP 123, 125
PUSH 123
READ 292
READ CD 46, 176, 281, 292, 295, 298, 303, 333, 343, 351, 363, 365, 367, 368
READ CD MSF 281
READ HEADER 295, 299, 303, 321
READ SUBCHANNEL 242, 299, 321, 327, 336, 343, 368—370
READ TOC 270, 271, 274, 310, 320, 327, 330, 333, 336, 343, 349, 363
READ TOC/PMA/ATIP 51
RET 122, 126, 127
SCSI_INQUIRY 168, 187
SCSI_READ10, 187
SEEK 295, 298, 299, 303, 321, 327, 333, 336, 343, 368—370
SendASPI32Command 188
SET CD SPEED 236
операции 166
параметры 166
Консоль:
Checked Build Environment 146
Free Build Environment 146
Контрольные суммы 17
"Красная книга" 22, 246

Л

Ленды 7, 16, 357, 441
Линии задержки 42

М

Метки 400
Методики защитных механизмов 5
Мини-порт 196, 200
  управление 200, 201
Модуль-нарушитель 138

О

Область:
  Lead-In 259, 428, 429, 437
  Lead-Out 428, 429
  Pre-gap 427, 428, 429, 431
  диска:
    вводная 49,
    выводная 50
  переходная 277
  постзазора 277
  предзазора 276
    расширенная 277
Обработка запросов:
  асинхронный режим 190
Операция:
  XOR 63, 70, 80, 88, 397
  взятия остатка (%) 72
"Оранжевая книга" 360, 366
Отверстие для аварийного выброса CD 247
Отладчик:
  i386kd 136, 138
  i386kd.exe 145, 146
  Kernel Debugger 146
  Soft-Ice 12, 138, 196, 231, 237, 387, 392
  Windows Debugger 145
Отражающий слой 15
Отсроченная ошибка 171
Очистка диска CD-RW:
  быстрая 426
  полная 426
Ошибки:
  CIRC 372
  MEDIUM ERROR 365, 367
  двухсимвольные 47
  исправимые 47
  неисправимые 47
  односимвольные 47
  текущие 171
  трехсимвольные 47
  уровня C1/C2, 371

П

Пакет 19, 351, 359
Переменная:
  unsigned long 291
Питы 7, 16, 357, 441
Поле:
  Absolute CD Address 336
  A-FRAC 21
  ALBA 340
  A-MIN 21
  Application Code 266
  A-SEC 21
  ATIP 266, 309
  Control 21
  CRC 21
  EDC/ECC 358
  Error Flags 46
  Expected Sector Type 176
  FRAC 21
  Header 24
  Highest Track Number 280
  INDEX 21
  Lowest Track Number 280
  MIN 21
  MODE 358, 360, 361, 363
  Packet Size 280
  PFrame 376
(окончание рубрики см. на стр. 452)

Поле *(окончание)*:
PLBA 431
PMin 376
Point 374
PSec 376
q-Data 21
q-Mode 21
SEC 21
Session 373
Sync 23, 358
TNO 284, 326, 327, 377
TocEntries 373
Type of Track Description Unit 280
ZERO 21
Полином:
локатора ошибки 90
порожденный 61, 84
сгенерированный 84
Поля Галуа 62, 63, 71
арифметика 68
деление 78
сложение по модулю 2, 72
умножение 78
Порт ввода/вывода 210
Прерывание:
INT 2Fh 222
INT 31h 222
Привод:
ASUS 4, 41, 258, 261, 293, 296, 304, 308, 309, 315, 316, 319, 321, 327, 335, 339, 367, 435, 436
CD-RW:
PHILIPS 41
Plextor 41
MSI 49, 50
NEC 4, 258, 259, 261, 293, 296, 304, 308, 315, 316, 319, 321, 327, 335, 344, 347, 353, 356, 367, 431
PHILIPS 4, 50, 190, 234, 247, 258, 261, 432, 434, 435
PHILIPS CDRW 24, 000, 407
PHILIPS CDRW 2400, 46, 435
Plextor 352, 358, 417
Primary 220
Secondary 220
TEAC 4, 236, 261, 293, 296, 304, 308, 315, 316, 319, 321, 327, 333, 339, 351, 356, 434, 436
корректирующая способность 434
предпочтительные скорости вращения 435
Приемы защиты CD 282
Программа:
"Pinch of File" 384
Ahead Nero 261, 277, 278, 283, 290, 294, 296, 300, 309, 311, 316, 321, 331, 352
динамическая библиотека newtrf.dll 284
Ahead Nero Burning ROM 83
Alcohol 120% 2, 197, 230—232, 237, 246, 256, 258, 261, 267, 269, 270, 274, 281, 283, 291, 295, 296, 300, 303, 304, 306, 309, 310, 312, 313, 324, 328, 333, 352, 366, 373, 376, 380, 382, 406, 407, 426—428, 441
ASPack 227
Blind Write 283, 291
c2u 354
Cactus Data Shield 2
CD_RAW_SECTOR_READ 361, 362
CD-Cops 2
CDRWin 230, 283, 427, 430
chkdsk.exe 141, 142
CloneCD 2, 83, 100, 229, 230, 237, 238, 246—248, 255, 258, 259, 261, 262, 266, 267, 269, 274, 281, 291—293, 298, 304—306, 309, 310, 312, 313, 316, 323, 333, 352, 356, 358, 361, 366, 369, 373, 376, 380, 406, 417, 426—429, 441
CRWin 300
DeXOR 396
Easy CD Creator 237, 238, 246, 260

fc.exe 354
HIEW 269, 368, 380, 394, 396
i386kd.exe 136
Nero Burning Rom 230, 246, 421, 425
Pinch of File 269
raw.TOC.exe 260
Roxio CD Copier 378
Roxio Easy CD Creator 248, 260, 373, 380, 419, 421
SafeDisk 2
SecuROM 2
StarForce 2
Stomp Record Now 246, 260, 261, 290, 294, 296, 309, 311, 352, 359, 360, 373, 419, 421, 425
Программная область 51
Проигрыватели аудиодисков 371
Процедура Ченя 92
Процессоры IA-32, 85

Р

Расстояние Хемминга 54, 55
Регистр:
сдвигающий 85
Редактор реестра 111
параметр Auto 111
параметр Debugger 111
параметр UserDebuggerHotKey 111
Режим:
DAO 278
RAW DAO 246, 247, 248, 256, 257, 259, 269, 376, 427, 431
RAW SAO 246, 256
TAO 278
TEST ONLY 103
Рекордер PHILPS 373

С

Сегмент состояния задачи 213
Сектора 8, 17, 260
ключевые 358
"слабые" 8
Секция 48
Сессия 419
закрытая 51
Сигнал:
HF 26
аналоговый 435
Сигнатура 201
Символы кодовые 54
Синдром 89
Синхрогруппа 16, 17, 26
Синхропоследовательность 411
Скоростная диаграмма чтения 407
Скремблирование 30, 352, 356
Сложение:
по модулю два 36
по модулю четыре 72
Сообщение 54
алфавит 54
дискретное 54
Стандарт:
ECMA-130, 19, 24, 31, 48, 305, 361, 416
IEC 908, 22
Стойкость к взлому 241
Структура:
IDEREGS 204
IO_SCSI_CAPABILITIES 169
IOCTL_SCSI_PASS_THROUGHT 183
IOCTL_SCSI_PASS_THROUGHT_DIRECT 171, 183
SCSI_PASS_THROUGH 168
SCSI_PASS_THROUGH_DIRECT 168, 169, 188
SCSI_REQUEST_BLOCK 168
SENDCMDINPARAMS 203, 204
SRB_ExecSCSICmd 188, 232
SRB_IO_CONTROL 201, 203
Субкод 17
Сумматоры 85

Т

Таблица lock-up 76
Такты 85
Том 419
Точка останова условная 231
Трек 49
  блок описания 278
  идентификатор описателя 280
  модуль описателя 279
  нестандартный номер 343—346
  стандартная организация 275
  таблицы описателя 279

У

Узловые точки 407
Указатель null-pointer 138
Умножители 85
Уникальные характеристики носителя 401
Устройство:
  MbDlDp32, 238
  MbMmDp32, 184, 229
Утилита:
  Ahead Nero CD Speed 295, 439
  Ahead Nero Drive Speed 435
  ASPI32.raw 271
  CD.lock.exe 328
  cd_raw_read 438, 441
  CD_READ_TOC 324
  CDRTOOLS 230
  drivers.exe 138
  dumpbin 132
  EDITBIN.EXE 207
  implib 101
  ISO9660.dir 422, 423, 424, 439, 440
  ISO9660.dir.exe 422, 424
  objdir.exe 196
  Scan Disk 262
  seek_and_Q 347, 348
  Slow CD 435
  SPTI.raw 271

Ф

Файл:
  asm.drf.zip 424
  boot.ini 143
  chkdsk.exe 145
  Drwtsn32.exe 112
  file name.ccd 430
  IMAGE.CCD 249, 270, 284, 285, 290, 292, 293, 307, 314—316, 426
  IMAGE.DIF 354
  IMAGE.IMG 249, 270, 355
  IMAGE.SUB 249, 284, 368, 370
  Memory.dmp 136
  NTDDSCSI.H 200
  pagefile.sys 142, 144
  recover.ccd 432
  recover.img 432
  SaveDump.exe 145
  scsi.h 202
  See You.mp3, 422, 423
  testt.exe 120
  windbg.exe 145
  подкачки 142
  символьных идентификаторов 136
Файловая система:
  High Sierra 161, 186
  ISO-9660, 161, 186, 380, 381, 421, 433, 439
  Joliet 161, 380, 381, 421, 433, 439
Файловый менеджер FAR 146, 229, 424
ФАПЧ 245
Фильтры 85
Флаг:
  dwCreationDisposition 154
  GENETIC_WRITE 156
Формат:
  ASF 393
  AVI 378, 393
  BMP 378
  MP2, 393

MP3, 373, 393, 396, 440
WAV 373, 378, 393
WMA 373, 393, 440
Формат сектора:
область пользовательских данных 24
поле EDC 24
поле Header 24
поле Intermediate 24
поле P-parity 24
поле Q-parity 24
поле Sync 23
поле заголовка 24
поле синхронизации 23
Фреймовый кадр 17
Функция:
ABSOLUTE_READ 405
ASPI32Post 187
CalcSubChannelCRC 284
CheckSector 102
CreateFile 154, 161, 183, 196, 200, 227, 275, 392, 404
CreateFileA 237, 238, 389
CWinThread
DeviceIoContorl 275, 405
DeviceIoControl 152, 154, 155, 167, 171, 180, 183, 184, 200, 203, 212, 224, 227, 228, 230, 238, 404
DispatchMessage 128
DPMI Simulate Real Mode Interrupt 222
EnumWindows 376
ExitProcess 127
FindWindows 376
fopen 405
fread 405, 408
FreeASPI32Buffer 184
GenECCAndEDC_Mode1, 101
GetASPI32Buffer 184
GetASPI32DLLVersion 184
GetASPI32SupportInfo 184
GetLastError 156
gf_sum 73
IoCreareDevice 229
IsIdleMessage 131—133
Ke386IoSetAccessProcess() 214, 215
Ke386QueryIoAccessMap() 214, 215
Ke386SetIoAccessMap() 214, 215
LoadLibraryA 231
PsGetCurrentProcess() 215
PumpMessage 131—133
Rand() 397
ReadFile 161, 384, 392
ReadFile/WriteFile 160
Run(void) 130
Run 130, 131, 133
SendASPI32Command 184, 227, 230, 231, 275
SetFilePointer 160, 161, 384, 389, 392
Simulate Real Mode Interrupt 405
Threshing 392
TranslateASPI32Address 184
TranslateMessage 128
WriteProcessMemory 110
XOR 416

Ц

ЦАП 244
Царапины CD:
концентрические 442
радиальные 442
узкие 441
широкие 442
Цилиндр 9

Ш

Шифр Вернама 33
Шифровка файлов 392

Э

Эмуляция носителя 242