

ФАЙЛОВАЯ СИСТЕМА NTFS ИЗВНЕ И ИЗНУТРИ

ЧАСТЬ 2

В продолжение знакомства с файловой системой NTFS сегодня мы сосредоточимся на строении атрибутов, исследовав их заголовки и механизмы хранения нерезидентного тела на диске, а также покажем, как рассмотренные нами структуры данных выглядят вживую в дисковом редакторе типа Disk Probe или Sector Inspector.

КРИС КАСПЕРСКИ

Атрибуты

Структурно всякий атрибут состоит из заголовка (attribute header) и тела (attribute body). Заголовок атрибута всегда хранится в файловой записи, расположенной внутри MFT

(см. первую часть статьи, «Файловые записи»). Тела резидентных атрибутов хранятся там же. Нерезидентные атрибуты хранят свое тело вне MFT, в одном или нескольких кластерах, перечисленных в заголовке данного атрибута в

специальном списке (см. «Списки отрезков»). Если 8-разрядное поле, расположенное по смещению 08h байт от начала атрибутного заголовка, равно нулю, атрибут считается резидентным, а если единице – то нет. Любые другие значения недопустимы.

Первые четыре байта атрибутного заголовка определяют его тип. Тип атрибута в свою очередь определяет формат представления тела атрибута. В частности, тело атрибута данных (тип: 80h – \$DATA) представляет собой «сырую» последовательность байт. Тело атрибута стандартной информации (тип: 10h – \$STANDARD_INFORMATION) описывает время его создания, права доступа и т. д. Подробнее см. «Типы атрибутов».

Следующие четыре байта заголовка содержат длину атрибута, выражаемую в байтах. Длина нерезидентного атрибута равна сумме длин его тела и заголовка, а длина резидентного атрибута равна длине его заголовка. Короче говоря, если к смещению атрибута добавить его длину, мы получим указатель на следующий атрибут (или маркер конца, если текущий атрибут – последний в цепочке).

Длина тела резидентных атрибутов, выраженная в байтах, хранится в 32-разрядном поле, расположенном по смещению 10h байт от начала атрибутного заголовка. 16-разрядное поле, следующее за его концом, хранит смещение резидентного тела, отсчитываемое от начала атрибутного заголовка.

С нерезидентными атрибутами в этом плане все намного сложнее и для хранения длины их тела используется множество полей. Реальный размер тела атрибута (real size of attribute), выраженный в байтах, хранится в 64-разрядном (!) поле, находящемся по смещению 30h байт от начала атрибутного заголовка. Следующее за ним 64-разрядное поле хранит инициализированный размер потока (initialized data size of the stream), выраженный в байтах и, судя по всему, всегда равный реальному размеру тела атрибута. 64-разрядное поле, расположенное по смещению 28h байт от начала атрибутного заголовка, хранит выделенный размер (allocated size of attribute), выраженный в байтах и равный реальному размеру тела атрибута, округленному до размера кластера (в большую сторону).

Два 64-разрядных поля, расположенные по смещению 10h и 18h байт от начала атрибутного заголовка, задают первый (starting VCN) и последний (last VCN) номер виртуального кластера, принадлежащего телу нерезидентного атрибута. Виртуальные кластеры представляют собой логические номера кластеров, не зависящие от своего физического расположения на диске. В подавляющем большинстве случаев номер первого кластера тела нерезидентного атрибута равен нулю, а последний – количеству кластеров, занятых телом атрибута, уменьшенном на единицу. 16-разрядное поле, расположенное по смещению 20h от начала атрибутного заголовка, содержит указатель на массив Data Runs, расположенный внутри этого заголовка и описывающий логический порядок размещения нерезидентного тела атрибута на диске (подробнее см. «Списки отрезков»).

Каждый атрибут имеет свой собственный идентификатор (attribute ID), уникальный для данной файловой записи и хранящийся в 16-разрядном поле, расположенном по смещению 0Eh от начала атрибутного заголовка.

Если атрибут имеет имя (attribute Name), то 16-разрядное поле, расположенное по смещению 0Ah байт от атрибутного заголовка, содержит указатель на него. Для безымянных атрибутов оно равно нулю (а большинство атрибутов безымянны!). Имя атрибута хранится в атрибутном заголовке в формате UNICODE, а его длина определяется 8-разрядным полем, расположенным по смещению 09h байт от начала атрибутного заголовка.

Если тело атрибута сжато, зашифровано или разряжено, 16-разрядное поле флагов, расположенное по смещению 0Ch байт от начала атрибутного заголовка, не равно нулю.

Остальные поля не играют сколь-нибудь существенной роли и потому здесь не рассматриваются.

Таблица 1. Структура резидентного атрибута

Смещение	Размер	Значение	Описание
00h	4		Тип (type) атрибута (например, 0x10, 0x60, 0xB0)
04h	4		Длина атрибута, включая этот заголовок
08h	1	00h	Нерезидентный флаг (non-resident flag)
09h	1	N	Длина имени атрибута (ноль, если атрибут безымянный)
0Ah	2	18h	Смещение имени (ноль, если атрибут безымянный)
0Ch	2	00h	Флаги
			Значение
			Описание
			0001h Сжатый атрибут (compressed)
		4000h	Зашифрованный атрибут (encrypted)
		8000h	Разряженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)
10h	4	L	Длина тела атрибута, без заголовка
14h	2	2N+18h	Смещение тела атрибута
16h	1		Индексный флаг
17h	1	00h	Для выравнивания
18h	2N	UNICODE	Имя атрибута (если есть)
2N+18h	L		Тело атрибута

Таблица 2. Структура нерезидентного атрибута

Смещение	Размер	Значение	Описание
00h	4		Тип (type) атрибута (например, 0x20, 0x80)
04h	4		Длина атрибута, включая этот заголовок
08h	1	01h	Нерезидентный флаг (non-resident flag)
09h	1	N	Длина имени атрибута (ноль, если атрибут безымянный)
0Ah	2	40h	Смещение имени (ноль, если атрибут безымянный)
0Ch	2		Флаги
			Значение
			Описание
			0001h Сжатый атрибут (compressed)
		4000h	Зашифрованный атрибут (encrypted)
		8000h	Разряженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)
10h	8		Начальный виртуальный кластер (starting VCN)
18h	8		Конечный виртуальный кластер (last VCN)
20h	2	2N+40h	Смещение списка отрезков (data runs)
22h	2		Размер блока сжатия (compression unit size), округленный до 4 байт вверх
24h	4	00h	Для выравнивания
28h	8		Выделенный размер (allocated size), округленный до размера кластера
30h	8		Реальный размер (real size)
38h	8		Инициализированный размер потока (initialized data size of the stream)
40h	2N	UNICODE	Имя атрибута если есть
2N+40h	..		Список отрезков (data runs)

Типы атрибутов

NTFS поддерживает большее количество предопределенных типов атрибутов, перечисленных в таблице 3. Как уже говорилось выше, тип атрибута определяет его назначение и формат представления тела. Полное описание всех атрибутов заняло бы очень много места и поэтому здесь приводятся лишь наиболее «ходовые» из них, а за информацией об остальных обращайтесь к Linux-NTFS Project.

Таблица 3. Основные типы атрибутов

Значение	ОС	Условное обозначение	Описание
010h	любая	\$STANDARD_INFORMATION	Стандартная информация о файле (время, права доступа)
020h	любая	\$ATTRIBUTE_LIST	Список атрибутов
030h	любая	\$FILE_NAME	Полное имя файла
040h	NT	\$VOLUME_VERSION	Версия тома
040h	2K	\$OBJECT_ID	Уникальный GUID и прочие ID
050h	любая	\$SECURITY_DESCRIPTOR	Дескриптор безопасности и списки прав доступа (ACL)
060h	любая	\$VOLUME_NAME	Имя тома
070h	любая	\$VOLUME_INFORMATION	Информация о томе
080h	любая	\$DATA	Основные данные файла
090h	любая	\$INDEX_ROOT	Корень индексов
0A0h	любая	\$INDEX_ALLOCATION	Ветви (sub-nodes) индекса
0B0h	любая	\$BITMAP	Карта свободного пространства
0C0h	NT	\$SYMBOLIC_LINK	Символическая связь
0C0h	2K	\$REPARSE_POINT	Для сторонних производителей
0D0h	любая	\$EA_INFORMATION	Расширенные атрибуты для HPFS
0E0 h	любая	\$EA	Расширенные атрибуты для HPFS
0F0h	NT	\$PROPERTY_SET	Устарело и ныне не используется
100h	2K	\$LOGGED_UTILITY_STREAM	Используется шифрованной файловой системой (EFS)

Атрибут стандартной информации \$STANDARD_INFORMATION

Атрибут стандартной информации описывает время создания/изменения/последнего доступа к файлу и права доступа, а также некоторую другую вспомогательную информацию (например, квоты):

Таблица 4. Структура атрибута \$STANDARD_INFORMATION

Смещение	Размер	ОС	Описание
~ ~		любая	Стандартный атрибутный заголовок (standard attribute header)
00h	8	любая	С время создания (creation) файла
08h	8	любая	А время изменения (altered) файла
10h	8	любая	М время изменения файловой записи (MFT changed)
18h	8	любая	Р время последнего чтения (read) файла
			Права доступа MS-DOS (MS-DOS file permissions)
			Значение Описание
			0001h Только на чтение (read-only)
			0002h Скрытый (hidden)
			0004h Системный (system)
			0020h Архивный (archive)
			0040h Устройство (device)
			0080h Обычный (normal)
			0100h Временный (temporary)
			0200h Разряженный (sparse) файл
			0400h reparse point
			0800h Сжатый (compressed)
			1000h Оффлайнный (offline)
			2000h Не индексируемый (not content indexed)
			4000h Зашифрованный (encrypted)
24h	4	любая	Старшее двойное слово номера версии (maximum number of versions)
28h	4	любая	Младшее двойное слово номера версии (version number)
2Ch	4	любая	Идентификатор класса (class ID)
30h	4	2K	Идентификатор владельца (owner ID)
34h	4	2K	Идентификатор безопасности (security ID)
38h	8	2K	Количество котируемых байт (quota charged)
40h	8	2K	Номер последней последовательности обновления (update sequence number USN)

Атрибут списка атрибутов \$ATTRIBUTE_LIST

Атрибут списка атрибутов (получился каламбур) используется в тех случаях, когда все атрибуты файла не умещаются в базовой файловой записи и файловая система вынуждена располагать их в расширенных. Индексы расширенных файловых записей содержатся в атрибуте списка атрибутов, помещаемом в базовую файловую запись.

При каких обстоятельствах атрибуты не умещаются в одной файловой записи? Это может произойти, когда:

- файл содержит много альтернативных имен или жестких ссылок;
- файл очень сильно фрагментирован;
- файл содержит очень сложный дескриптор безопасности;
- файл имеет очень много потоков данных (т.е. атрибутов типа \$DATA).

Структура атрибута списка атрибутов приведена ниже:

Таблица 5. Структура атрибута \$ATTRIBUTE_LIST

Смещение	Размер	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	4	Тип (type) атрибута (см. таблицу 3)
04h	2	Длина записи (record length)
06h	1	Длина имени (name length), или ноль, если нет, условно – N
07h	1	Смещение имени (offset to name), или ноль, если нет
08h	8	Начальный виртуальный кластер (starting VCN)
10h	8	Ссылка на базовую/расширенную файловую запись
18h	2	Идентификатор атрибута (attribute ID)
1Ah	2N	if N > 0, то имя в формате UNICODE

Атрибут полного имени \$FILE_NAME

Атрибут полного имени файла хранит имя файла в соответствующем пространстве имен. Таких атрибутов у файла может быть и несколько (например, win32-имя и MS-DOS имя). Здесь же хранятся и жесткие ссылки (hard link), если они есть.

Структура атрибута полного имени приведена ниже:

Таблица 6. Структура атрибута \$FILE_NAME

Смещение	Размер	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	8	Ссылка (file reference) на материнский каталог
08h	8	С – время создания (creation) файла
10h	8	А – время последнего изменения (altered) файла
18h	8	М – время последнего изменения файловой записи (MFT changed)
20h	8	Р – время последнего чтения (read) файла
28h	8	Выделенный размер (allocated size) файла
30h	8	Реальный размер (real size) файла
38h	4	Флаг (см. таблицу 4)
3Ch	4	Используется HPFS
40h	1	Длина имени в символах – L
41h	1	Пространство имен файла (filename namespace)
42h	2L	Имя файла в формате UNICODE без завершающего нуля

Списки отрезков (data runs)

Тела нерезидентных атрибутов хранятся на диске в одной или нескольких кластерных цепочках, называемых отрезками (runs). Отрезком называется последовательность смежных кластеров, характеризующаяся номером начального кластера и длиной. Совокупность отрезков называется списком, run-list или data run.

Внутренний формат представления списков не то, чтобы сложен, но явно не прост, за что получил прозвище brain damage format (формата, срывающего крышу). Для экономии места длина отрезка и номер начального кластера хранятся в полях переменной длины. То есть, если размер отрезка умещается в байт (т.е. его значение не превышает 255), он и хранится в байте. Соответственно, если размер отрезка требует для своего представления двойного слова, он и хранится в двойном слове.

Сами же поля размеров хранятся в 4-байтовых ячейках, называемых nibblami (nibble) или полубайтами. Шестнадцатеричная система исчисления позволяет легко перево-

дить байты в nibblы и наоборот. Младший nibbl равен $(X \times 15)$, а старший – $(X / 16)$. Легко видеть, что младший nibbl соответствует младшему шестнадцатеричному разряду байта, а старший – старшему. Например, 69h состоит из двух nibblов – младший равен 9h, а старший – 6h.

Список отрезков представляет собой массив структур, каждая из которых описывает характеристики «своего» отрезка, а в конце списка находится завершающий нуль. Первый байт структуры состоит из двух полубайт: младший задает длину поля начального кластера отрезка (условно обозначаемого буквой F), старший – количество кластеров в отрезке (L). Поле длины отрезка идет следом. В зависимости от значения L оно может занимать от одного до восьми байт (более длинные поля недопустимы). Первый байт поля стартового кластера файла расположен по смещению $1 + L$ байт от начала структуры (что соответствует $2+2*L$ nibblам). Кстати говоря, в документации Linux-NTFS Project (версия 0.4) поля размеров начального кластера и количества кластеров в отрезке перепутаны местами.

Таблица 7. Структура одного элемента списка отрезков

Смещение в nibblax	Размер в nibblax	Описание
0	1	Размер поля длины (L)
1	1	Размер поля начального кластера (S)
2	$2*L$	Количество кластеров в отрезке
$2+2*L$	$2*S$	Номер начального кластера отрезка

Покажем, как с этим работать на практике. Допустим, мы имеем следующий run-list, соответствующий нормальному нефрагментированному файлу (что может быть проще!): «21 18 34 56 00». Попробуем его декодировать.

Начнем с первого байта – 21h. Младший полубайт (01h) описывает размер поля длины отрезка, старший (02h) – размер поля начального кластера. Следующие несколько байт представляют поле длины отрезка, размер которого в данном случае равен одному байту – 18h. Два других байта (34h 56h) задают номер начального кластера отрезка. Нулевой байт на конце сигнализирует о том, что это последний отрезок в файле. Итак, наш файл состоит из одного-единственного отрезка, начинающегося с кластера 5634h и заканчивающегося кластером $5634h + 18h == 564Ch$.

Рассмотрим более сложный пример фрагментированного файла со следующим списком отрезков: «31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00». Извлекаем первый байт – 31h. Один байт приходится на поле длины и три байта на поле начального кластера. Таким образом, первый отрезок (run 1) начинается с кластера 342573h и продолжается вплоть до кластера $342573h + 38 == 3425ABh$. Чтобы найти смещение следующего отрезка в списке, мы складываем размер обоих полей с их начальным смещением: $3 + 1 == 4$. Отсчитываем четыре байта от начала run-list и переходим к декодированию следующего отрезка: 32h – два байта на поле длины отрезка (равное в данном случае 0114h) и три байта на поле номера начального кластера (0211E5h). Следовательно, второй отрезок (run 2) начинается с кластера 0211E5h и продолжается вплоть до кластера $0211E5h + 114h == 212F9h$. Третий отрезок (run 3): 31h – один байт на поле длины и три байта на поле начального кластера, равные 42h и 0300AAh соответственно. Поэтому третий отрезок (run 3) начинается с кластера 0300AAh и продолжается вплоть до кластера $0300AAh + 42h == 300ECh$.

Завершающий нуль на конце run-list сигнализирует о том, что это последний отрезок в файле.

Таким образом, подопытный файл состоит из трех отрезков, разбросанных по диску в следующем живописном порядке: 342573h – 3425ABh; 0211E5h – 212F9h; 0300AAh – 300ECh. Остается только прочитать его с диска!

Начиная с версии 3.0, NTFS поддерживает разреженные (sparse) атрибуты, т.е. такие, которые не записывают на диск кластеры, содержащие одни нули. При этом поле номера начального кластера отрезка может быть равным нулю, что означает, что данному отрезку не выделен никакой кластер. Поле длины содержит количество кластеров, заполненных нулями. Их не нужно считывать с диска. Вы должны самостоятельно изготовить их в памяти. Кстати говоря, далеко не все дисковые доктора знают о существовании разреженных атрибутов (если атрибут разрежен его флаг равен 8000h), и интерпретируют нулевую длину поля номера начального кластера весьма странным образом. Последствия такого «лечения» обычно оказываются очень печальными.

Пространства имен (name spaces)

NTFS изначально проектировалась как системно-независимая файловая система, способная работать со множеством различных подсистем, как то: Win32, MS-DOS, POSIX и т. д. Поскольку, каждая из них налагает свои собственные ограничения на набор символов, допустимых для использования в имени файла, NTFS вынуждена поддерживать несколько независимых пространств имен (name space).

POSIX

Допустимы все UNICODE-символы (с учетом регистра), за исключением символа нуля (NULL), обратного слеша («/») и знака двоеточия («:»). Последнее, кстати говоря, не ограничение POSIX, а ограничение NTFS, использующей этот символ для доступа к именованным атрибутам. Максимально допустимая длина имени составляет 255 символов.

Win32

Доступны все UNICODE-символы (без учета регистра), за исключением следующего набора: «"» (кавычки), «*» (звездочка), «/» (прямой слеш), «:» (двоеточие), «<» (знак меньше), «>» (знак больше), «?» (вопросительный знак), «\» (обратный слеш), «|» (символ конвейера). К тому же, имя файла не может заканчиваться на точку или пробел. Максимально допустимая длина имени составляет 255 символов.

MS-DOS

Доступны все символы пространства имен win32 (без учета регистра), за исключением: «+» (знак плюс), «,» (знак запятой), «.» (знак точка), «;» (точка с запятой), «=» (знак равно). Имя файла не должно превышать восьми символов за которыми следует необязательное расширение с длиной от одного до трех символов.

Назначение некоторых служебных файлов

NTFS содержит большое количество служебных файлов (метафайлов) строго определенного формата, важнейший из которых – \$MFT – мы только что рассмотрели. Осталь-

ные метафайлы играют вспомогательную роль и для восстановления данных знать их структуру, в общем-то, и необязательно. Тем не менее если они окажутся искажены, штатный драйвер файловой системы не сможет работать с таким томом, поэтому иметь некоторые представления о назначении каждого из них все же необходимо.

У нас нет возможности рассказать о структуре всех метафайлов (да и незачем дублировать Linux-NTFS Project), поэтому эта информация здесь не приводится.

Таблица 8. Назначение основных стандартных файлов

inode	Имя файла	ОС	Описание
0	\$MFT	любая	Главная файловая таблица (Master File Table, MFT)
1	\$MFTMirr	любая	Резервная копия первых четырех элементов MFT
2	\$LogFile	любая	Журнал транзакций (transactional logging file)
3	\$Volume	любая	Серийный номер, время создания, dirty flag (флаг не сброшенного кэша) тома
4	\$AttrDef	любая	Определение атрибутов
5	.	любая	Корневой каталог (root directory) тома
6	\$Bitmap	любая	Карта свободного/занятого пространства
7	\$Boot	любая	Загрузочная запись (boot record) тома
8	\$BadClus	любая	Список плохих кластеров (bad clusters) тома
9	\$Quota	NT	Информация о квотах (quota information)
9	\$Secure	2K	Использованные дескрипторы безопасности (security descriptors)
10	\$UpCase	любая	Таблица заглавных символов (uppercase characters) для трансляции имен
11	\$Extend	2K	Каталоги: \$ObjId, \$Quota, \$Reparse, \$UsnJrnl
12-15	не используется	любая	Помечены как использованные, но в действительности пустые
16-23	не используется	любая	Помечены как неиспользуемые
любой	\$ObjId	2K	Уникальные идентификаторы каждого файла
любой	\$Quota	2K	Информация о квотах (quota information)
любой	\$Reparse	2K	Информация типа reparse point
любой	\$UsnJrnl	2K	Журнал шифрованной файловой системы (journaling of encryption)
> 24	файл пользователя	любая	Обычные файлы
> 24	каталог пользователя	любая	Обычные каталоги

Путешествие по NTFS

Рассказ о NTFS был бы неполным без практической иллюстрации техники разбора файловой записи «руками». До сих пор мы витали в облаках теоретической абстракции. Пора спускаться на грешную землю.

Воспользовавшись любым дисковым редактором, например, Disk Probe, попробуем декодировать одну файловую запись вручную. Найдем сектор, содержащий сигнатуру «FILE» в его начале (не обязательно брать первый встретившийся сектор). Он может выглядеть, например, так:

Листинг 1. Ручное декодирование файловой записи (разные атрибуты выделены разным цветом)

: 00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	
00000000: 46 49 4C 45 2A 00 03 00	60 79 1A 04 02 00 00 00	FILE* y-♦
00000010: 01 00 01 00 30 00 01 00	50 01 00 00 00 04 00 00	0 0 P♦
00000020: 00 00 00 00 00 00 00 00	04 00 03 00 00 00 00 00	♦ ♦
00000030: 10 00 00 00 60 00 00 00	00 00 00 00 00 00 00 00	
00000040: 48 00 00 00 18 00 00 00	B0 D5 C9 2F C6 0B C4 01	н Ff/ b-~
00000050: E0 5A B3 7B A9 FA C3 01	90 90 F1 2F C6 0B C4 01	pz/ (x- K PPe/ b-~
00000060: 50 7F BC FE C8 0B C4 01	20 00 00 00 00 00 00 00	Pd■ b-~
00000070: 00 00 00 00 00 00 00 00	00 00 00 00 05 01 00 00	
00000080: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000090: 30 00 00 00 70 00 00 00	00 00 00 00 00 00 02 00	0 P
000000A0: 54 00 00 00 18 00 01 00	DB 1A 01 00 00 00 01 00	T
000000B0: B0 D5 C9 2F C6 0B C4 01	B0 D5 C9 2F C6 0B C4 01	Ff/ b-~
000000C0: B0 D5 C9 2F C6 0B C4 01	B0 D5 C9 2F C6 0B C4 01	Ff/ b-~
000000D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000E0: 20 00 00 00 00 00 00 00	09 03 49 00 6C 00 66 00	0 P
000000F0: 61 00 6B 00 2E 00 64 00	62 00 78 00 00 00 00 00	a k . d b x
00000100: 80 00 00 00 48 00 00 00	01 00 00 00 00 00 03 00	A H ♦
00000110: 00 00 00 00 00 00 00 00	ED 04 00 00 00 00 00 00	э+
00000120: 40 00 00 00 00 00 00 00	F0 E0 4E 00 00 00 00 00	@ pN
00000130: F0 D1 4E 00 00 00 00 00	F0 D1 4E 00 00 00 00 00	E pN
00000140: 32 EE 04 D9 91 00 00 81	FF FF FF FF 82 79 47 11	2x J C B ByG
000001F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 03 00	
: 00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	

Первым делом необходимо восстановить оригинальное

содержимое последовательности обновления. По смещению 04h от начала сектора лежит 16-разрядный указатель на нее, равный в данном случае 2Ah (значит, это NTFS 3.0 или младше). А что у нас лежит по смещению 2Ah? Пара байт 03 00. Это номер последовательности обновления. Сверяем его с содержимым двух последних байт этого и следующего секторов (смещения 1FEh и 3FEh соответственно). Они равны! Значит, данная файловая запись цела (по крайней мере внешне) и можно переходить к операции спасения. По смещению 2Ch расположен массив, содержащий оригинальные значения последовательности обновления. Количество элементов в нем равно содержимому 16-разрядного поля, расположенному по смещению 06h от начала сектора и уменьшенного на единицу (т.е. в данном случае 03h – 01h == 02h). Извлекаем два слова начиная со смещения 2Ch (в данном случае они равны 00 00 и 00 00) и записываем их в конец первого и последнего секторов.

Теперь нам необходимо выяснить – используется ли данная файловая запись или ассоциированный с ней файл/каталог был удален. 16-разрядное поле, расположенное по смещению 16h, содержит значение 01h. Следовательно, перед нами файл, а не каталог, и этот файл еще не удален. Но является ли данная файловая запись базовой для данного файла или мы имеем дело с ее продолжением? 64-разрядное поле, расположенное по смещению 20h, равно нулю, следовательно, данная файловая запись – базовая.

Переходим к исследованию атрибутов. 16-разрядное поле, находящееся по смещению 14h равно 30h, следовательно, заголовок первого атрибута начинается со смещения 30h от начала сектора.

Первое двойное слово атрибута равно 10h, значит, перед нами атрибут типа \$STANDARD_INFORMATION. 32-разрядное поле длины атрибута, находящееся по смещению 04h и равное в данном случае 60h байт, позволяет нам вычислить смещение следующего атрибута в списке – 30h (смещение нашего атрибута) + 60h (его длина) == 90h (смещение следующего атрибута). Первое двойное слово следующего атрибута равно 30h, значит, это атрибут типа \$NAME и следующее 32-разрядное поле хранит его длину, равную в данном случае 70h. Сложив длину атрибута с его смещением, мы получим смещение следующего атрибута – 90h + 70h == 100h. Первое двойное слово третьего атрибута равно 80h, следовательно, это атрибут типа \$DATA, хранящий основные данные файла. Складываем его смещение с длиной – 100h + 32h == 132h. Мы наткнулись на частотол FFFFFFFh, сигнализирующий о том, что атрибут \$DATA последний в списке.

Теперь, разбив файловую запись на атрибуты, приступим к исследованию каждого из атрибутов в отдельности. Начнем с имени. 8-разрядное поле, находящееся по смещению 08h от начала атрибутного заголовка (и по смещению 98h от начала сектора), содержит флаг нерезидентности, который в данном случае равен нулю (т.е. атрибут резидентный и его тело хранится непосредственно в самой файловой записи). 16-разрядное поле, расположенное по смещению 0Ch от начала атрибутного заголовка (и по смещению 9Ch от начала сектора) равно нулю, следовательно тело атрибута не сжато и не зашифровано. 32-разрядное поле, расположенное по смещению 10h от начала атрибут-

ного заголовка и по смещению A0h от начала сектора, содержит длину атрибутного тела, равную в данном случае 54h байт, а 16-разрядное поле, расположенное по смещению 14h от начала атрибутного заголовка и по смещению A4h от начала сектора, хранит смещение атрибутного тела, равное в данном случае 18h, следовательно, тело атрибута \$NAME располагается по смещению A8h от начала сектора.

Формат атрибута типа \$NAME описан в таблице XX. Первые восемь байт содержат ссылку на материнский каталог данного файла, равную в данном случае 11ADBh:01 (индекс – 11ADBh, номер последовательности – 01h). Следующие 32-байта содержат штампы времени создания, изменения и времени последнего доступа к файлу. По смещению 28h от начала тела атрибута и D0h от начала сектора лежит 64-разрядное поле выделенного размера, а за ним – 64-разрядное поле реального размера. Оба равны нулю, что означает, что за размером файла следует обращаться к атрибутам типа \$DATA.

Длина имени файла содержится в 8-разрядном поле, находящемся по смещению 40h байт от начала тела атрибута и по смещению E8h от начала сектора. В данном случае оно равно 09h. Само же имя начинается со смещения 42h от начала тела атрибута и со смещения EAh от начала сектора. И здесь находится llfak.dbx.

Переходим к атрибуту основных данных файла, пропустив атрибут стандартной информации, который не содержит решительно ничего интересного. 8-разрядный флаг нерезидентности, расположенный по смещению 08h от начала атрибутного заголовка и по смещению 108h от начала сектора, равен 01h, следовательно атрибут нерезидентный. 16-разрядный флаг, расположенный по смещению 0Ch от начала атрибутного заголовка и по смещению 10Ch от начала сектора, равен нулю, значит, атрибут не сжат и не зашифрован. 8-разрядное поле, расположенное по смещению

09h от начала атрибутного заголовка и по смещению 109h от начала сектора, равно нулю – атрибут безымянный. Реальная длина тела атрибута (в байтах) содержится в 64-разрядном поле, расположенном по смещению 30h от начала атрибутного заголовка и по смещению 130h от начала сектора. В данном случае она равна 4ED1F0h (5.165.552). Два 64-разрядных поля, расположенных по смещениям 10h/110h и 18h/118h байт от начала атрибутного заголовка/сектора соответственно, содержат начальный и конечный номер виртуального кластера нерезидентного тела. В данном случае они равны: 0000h/4EDh.

Остается лишь декодировать список отрезков, адрес которого хранится в 16-разрядном поле, находящемся по смещению 20h от начала атрибутного заголовка и 120h от начала сектора. В данном случае оно равно 40h, что соответствует смещению от начала сектора в 140h. Сам же список отрезков выглядит так: 32 EE 04 D9 91 00 00. Два байта занимает поле длины (равное в данном случае 04EEh кластерам) и три – поле начального кластера (0091h). Завершающий ноль на конце говорит о том, что этот отрезок последний в списке отрезков.

Подытожим полученную информацию. Файл называется llfak.dbx, он начинается с кластера 0091h и продолжается вплоть до кластера 57Fh, при реальной длине файла в 5 165 552 байт. За сим все! Теперь уже ничего не стоит скопировать файл на резервный носитель (например, ZIP или стример).

Заключение

Вооруженные джентльменским набором знаний (а также дисковым редактором в придачу), мы готовы дать решительный отпор энтропии, потеснив ее по всем фронтам. Следующая статья этого цикла расскажет о том как восстанавливать удаленные файлы, отформатированные раздели и разрушенные служебные структуры данных.

Что читать

Основными источниками данных по NTFS служат:

- Книга Хелен Кастер (Helen Custer, часто сокращаемая до просто «Helen») «Inside the Windows NT file system» (в русском издании она входит в состав «Основы Windows NT и NTFS»), подробно описывающая концепции файловой системы и дающая о ней общее представление. К сожалению, все объяснения ведутся на абстрактном уровне без указания конкретных числовых значений, смещений и структур. К тому же в операционных системах Windows 2000 и Windows XP с файловой системой произошли значительные изменения, никак не отраженные в книге. Если не найдете эту книгу в магазинах – ищите ее в файлообменных сетях. (Например, www.eMule.ru).
- Хакерская документация от коллектива «Linux-NTFS Project» (<http://linux-ntfs.sourceforge.net>), чьим хобби долгое время была разработка независимого NTFS-драйвера для OS Linux. Однако сейчас энтузиазм команды начал стремительно угасать. Это выдающееся творение, подробно описывающее все ключевые структуры файловой системы (естественно, на английском языке), отнюдь не заменяет книгу Хелен, а лишь расширяет ее! Разобраться в NTFS-project без знаний NTFS очень и очень непросто!
- Документация от Active Data Recovery Software на утилиту Active Uneraser, бесплатную копию которой можно найти на сайте www.NTFS.com. Это своеобразный синтез книги Хелен и Linux-NTFS Project, описывающий важнейшие структуры данных и обходящий стороной все вопросы, которые только можно обойти. Здесь же можно найти до предела выхолощенное изложение методики восстановления данных. Если не найдете Хелен, скачайте демонстрационную версию Active Uneraser и воспользуйтесь прилагаемой к нему документацией (Active Uneraser поставляется в двух вариантах – образе FDD и образе CD, документация присутствует только на последнем из них).
- Контекстная помощь на Runtime Software Disk Explorer также содержит достаточно подробное описание файловой системы, однако, на редкость бестолково организованное. Для упрощения навигации по тексту рекомендуется декомпилировать chm-файл в обычный текст, вручную перебив его в MS Word, pdf или любой другой удобный для вас формат.