ФАЙЛОВАЯ СИСТЕМА NTFS ИЗВНЕ И ИЗНУТРИ

Покажи мне свои структуры данных, и я скажу, кто ты!

Народная программистская мудрость

В этой статье мы рассмотрим основные структуры данных файловой системы NTFS, определяющие ее суть – главную файловую запись MFT, файловые записи FILE Record и последовательности обновления (update sequence или fix-ups), без знания которых осмысленная работа с редактором диска и ручное/полуавтоматическое восстановление данных просто невозможны!

КРИС КАСПЕРСКИ

Файловую систему NTFS принято описывать как сложную реляционную базу данных, обескураживающую грандиозностью своего архитектурного замысла не одно поколение начинающих исследователей. NTFS похожа на огромный, окутанный мраком лабиринт, в котором очень легко заблудиться. Хакеры давно разобрались с основными структурами данных, осветив магистральные коридоры лабиринта светом множества факелов. Боковые ответвления разведаны намного хуже и все еще находятся по власти тьмы, хранящей множество смертоносных ловушек, ждущих своих исследователей. В общем, если NTFS-«читалку» можно запрограммировать буквально за один вечер (с отладкой!), писать на NTFS-тома еще никто не рисковал.

К счастью, никто не требует от нас написания полноценного NTFS-драйвера! Наша задача значительно скромнее – вернуть разрушенный том в состояние, пригодное для восприятия операционной системой (задача-максимум) или извлечь из него все ценные файлы (задача-минимум). Вникать в структуру журналов транзакций, дескрипторов безопасности, двоичных деревьев индексаций для этого совершенно необязательно! Реально нам потребуется разобраться лишь с устройством главной файловой записи – MFT и нескольких дочерних подструктур.

Обзор NTFS с высоты птичьего полета

Основным структурным элементом всякой файловой системы является том (volume), в случае с FAT совпадающий с разделом (partition), о котором мы говорили в прошлой статье [1, 2]. NTFS поддерживает тома, состоящие из нескольких разделов. Подробнее схему отображения томов на разделы мы обсудим в следующей статье этого цикла, а пока же будем для простоты считать, что том представляет собой отформатированный раздел (т.е. раздел, содержащий служебные структуры файловой системы).

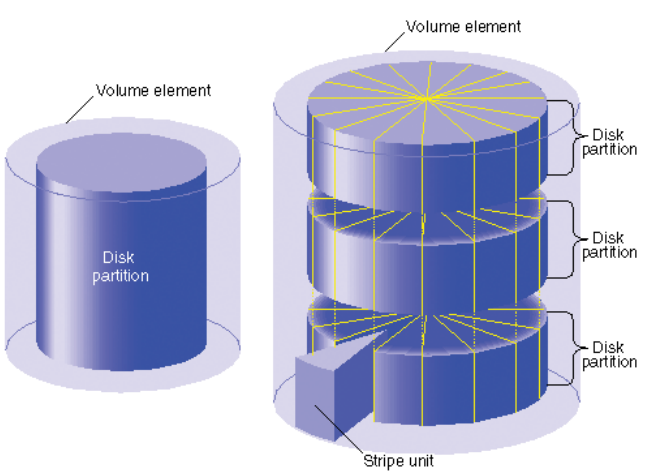

Рисунок 1. Обычный (слева) и разряженный (справа) тома

Большинство файловых систем трактуют том как совокупность файлов, свободного дискового пространства и служебных структур файловой системы, но в NTFS все служебные структуры представлены файлами, которые (как это и положено файлу) могут находиться в любом месте тома, при необходимости фрагментируя себя на несколько частей.

Самым главным служебным файлом является $MFT – Master File Table (главная файловая таблица) – своеобразная база данных, хранящая информацию обо всех файлах тома – их именах, атрибутах, способе и порядке размещения на диске (каталог также является файлом особого типа, со списком принадлежащих ему файлов и подкаталогов внутри). Важно подчеркнуть, что в MFT присутствуют все файлы, находящиеся во всех подкаталогах тома, поэтому для восстановления диска наличия $MTF-файла будет вполне достаточно.

Остальные служебные файлы (кстати говоря, называемые метафайлами или метаданными – metafile/metadata соответственно и всегда предваряются знаком доллара «$») носят сугубо вспомогательный характер, интересный только самой файловой системе. К ним в первую очередь относятся: $LogFile – файл транзакций, $Bitmap – карта свободного/занятого пространства, $BadClust – перечень плохих кластеров и т. д. (Подробнее о назначении каждого из файлов будет рассказано в следующей статье). Текущие версии Windows блокируют доступ к служебным файлам с прикладного уровня (даже с правами администратора!), и всякая попытка открытия/создания такого файла в корневом каталоге обречена на неуспех.

Классическое определение, данное в учебниках информатики, отождествляет файл с именованной записью на диске. Большинство файловых систем добавляет к этому понятие атрибута (attribute) – некоторой вспомогательной характеристики, описывающей время создания, права доступа и т. д. В NTFS имя файла, данные файла и его атрибуты полностью уравнены в правах. Можно сказать, что всякий NTFS-файл представляет собой совокупность атрибутов, каждый из которых хранится как отдельный поток (streams) байтов, поэтому во избежание путаницы атрибуты, хранящие данные файла, часто называют потоками.

Каждый атрибут состоит из тела (body) и заголовка (header). Атрибуты делятся на резидентные (resident) и нерезидентные (non-resident). Резидентные атрибуты хранятся непосредственно в $MTF, что существенно уменьшает грануляцию дискового пространства и сокращает время доступа. Нерезидентные – хранят в $MTF лишь свой заголовок, описывающий порядок размещения атрибута на диске.

Назначение атрибута определяется его типом (type) – четырехбайтовым шестнадцатеричным значением. При желании атрибуту можно дать еще и имя (name), состоящее из символов, входящих в соответствующее пространство имен. Подавляющее большинство файлов имеет по меньшей мере три атрибута: стандартная информация о файле (время создания, модификации последнего доступа, права доступа и т. д.) хранится в атрибуте типа 10h, условно обозначаемом $STANDARD_INFORMATION. Ранние версии Windows NT позволяли обращаться к атрибутам по их условным обозначениям, однако Windows 2000 и Windows XP лишены этой возможности. Полное имя файла (не путать с путем!) хранится в атрибуте типа 30h ($FILE_NAME). Если у файла есть одно или более альтернативных имен (например, MS-DOS-имя), таких атрибутов может быть несколько. Здесь же хранится ссылка (file reference) на материнский каталог, позволяющая разобраться, к какому каталогу данный файл/подкаталог принадлежит. Данные файла по умолчанию хранятся в безымянном атрибуте типа 80h ($DATA), однако при желании прикладные приложения могут создавать дополнительные потоки данных, отделяя имя атрибута от имени файла знаком двоеточия, например:

```
ECHO xxx > file:attr1; ECHO yyy > file:attr2; ↵
   more < file:attr1; more  < file:attr2
```

Изначально в NTFS была заложена способность индексации любых атрибутов, значительно сокращающая время поиска файла по заданному списку критериев (например, времени последнего доступа). Внутренние индексы хранятся в виде двоичных деревьев, поэтому среднее время выполнения запроса оценивается как O(lg n). Однако в текущих NTFS-драйверах реализована индексация лишь по одному атрибуту – имени файла. Как уже говорилось выше, каталог представляет собой особенный файл – файл индексов (INDEX). В отличие от FAT, где файл каталога представляет единственный источник данных об организации файлов, в NTFS он используется лишь для ускорения доступа к содержимому директории и не является обязательным, поскольку ссылка на материнский каталог всякого файла в обязательном порядке присутствует в атрибуте его имени ($FILE_NAME).

Каждый атрибут может быть зашифрован, разряжен или сжат. Однако техника работы с такими атрибутами выходит далеко за рамки первичного знакомства с организацией файловой системы и будет рассмотрена позднее. А пока же мы углубимся в изучение фундамента файловой системы – структуры $MFT.

Главная файловая запись (master file table)

В процессе форматирования логического раздела, в его начале создается так называемая MTF-зона (MFT-zone), по умолчанию занимающая 12,5% от емкости тома (а вовсе не 12%, как утверждается во многих публикациях), хотя в зависимости от значения параметра NtfsMftZoneReservation она может составлять 25%, 37% или 50%.

В этой области расположен $MFT-файл, изначально занимающий порядка 64 секторов и растущий от начала

Версии NTFS

Служебные структуры файловой системы NTFS не остаются постоянными, а слегка меняются от одной версии Windows NT к другой (см. таблицу 1). Этот факт следует принять во внимание при использовании автоматизированных «докторов». Попав на более свежую версию NTFS, «доктор», не оснащенный мощным AI, может запутаться в измененных структурах и разрушить вполне здоровый том.

Таблица 1. Определение версии NTFS по операционной системе

Версия NTFS	Операционная система	Условное обозначение
1.2	Windows NT	NT
3.0	Windows 2000	W2K
3.1	Windows XP	XP

MFT-зоны к ее концу по мере создания новых пользовательских файлов/подкаталогов. Таким образом, чем больше файлов содержится на дисковом томе, тем больше размер MFT. Приблизительный размер $MFT-файла можно оценить по следующей формуле: sizeof(FILE Record) N Files, где sizeof(FILE Record) обычно равен 1 Кб, а N Files – полное количество файлов/подканалов раздела, включая недавно удаленные.

Для предотвращения фрагментации $MFT-файла MFT-зона удержится зарезервированной вплоть до полного исчерпания свободного пространства тома, затем незадействованный «хвост» MFT-зоны усекается в два раза, освобождая место для пользовательских файлов. Этот процесс может повторяться многократно, вплоть до полной отдачи всего зарезервированного пространства. Решение красивое, хотя и не новое. Многие из файловых систем восьмидесятых позволяли резервировать заданное дисковое пространство в хвосте активных файлов, тем самым сокращая их фрагментацию (причем любых файлов, а не только служебных). В частности, такая способность была у DOS 3.0, разработанной для персональных компьютеров типа Агат. Может, кто помнит такую машину?

Когда $MFT-файл достигает границ MFT-зоны, в ходе своего последующего роста он неизбежно фрагментируется, вызывая обвальное падение производительности файловой системы, причем подавляющее большинство дефрагментаторов $MFT-файл не обрабатывают! А ведь API дефрагментации, встроенное в штатный NTFS-драйвер, это в принципе позволяет! Подробности (вместе с самой утилитой дефрагментации) можно найти на сайте Марка Руссиновича. Но, как бы то ни было, заполнять дисковый том более чем на 88% его емкости категорически не рекомендуется!

При необходимости $MFT-файл может быть перемещен в любую часть диска, и тогда в начале тома его уже не окажется. Стартовый адрес $MFT-файла хранится в Boot-секторе по смещению 30h байт от его начала (см. «boot-сектор», описанный в предыдущей статье данного цикла) и в подавляющем большинстве случаев этот адрес ссылается на 4-й кластер.

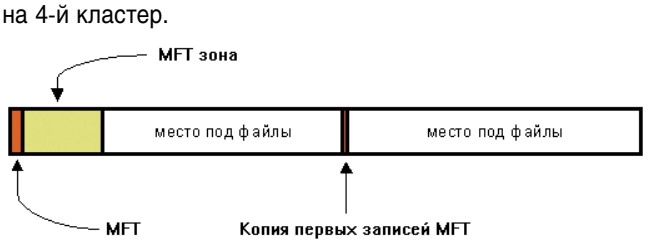

Рисунок 2. Структура дискового тома под NTFS

$MFT-файл представляет собой массив записей типа FILE Record (или в терминологии UNIX – inodes), каждая из которых описывает соответствующий ей файл или подкаталог (подробнее см. раздел «Файловые записи»).

В подавляющем большинстве случаев файл/подкаталог полностью описывается одной-единственной записью FILE Record, хотя теоретически этих записей может потребоваться и несколько.

Для ссылки на одну файловую запись из другой используется ее порядковый номер (он же индекс) в $MFT-файле, отсчитываемый от нуля. Файловая ссылка (file reference) состоит из двух частей (см. таблицу 2) – 48-битного индекса и 16-битного номера последовательности (sequence number).

При удалении файла/каталога соответствующая ему файловая последовательность помечается как неиспользуемая. При создании новых файлов записи, помеченные как неиспользуемые, могут задействоваться вновь, при этом счетчик номера последовательности, хранящийся внутри файловой записи, увеличивается на единицу. Этот механизм позволяет отслеживать «мертвые» ссылки на уже удаленные файлы – очевидно, sequence number внутри file reference будет отличаться от номера последовательности соответствующей файловой записи (этой проверкой занимается утилита chkdsk и автоматически, насколько мне известно, она не выполняется).

Таблица 2. Структура файловой ссылки (file reference)

Смещение	Размер	Описание
00h	6	Индекс файловой записи (FILE record number), отсчитываемый от нуля
06h	2	Номер последовательности (sequence number)

Первые 12 записей в MFT всегда занимают служебные метафайлы: $MFT (собственно, сам $MFT), $MFTMirr (зеркало $MFT), $LogFile (файл транзакций), $Volume (сведения о дисковом томе), $AttrDef (определенные атрибуты), «.» (корневой каталог), $Bitmap (карта свободного пространства), $Boot (системный загрузчик), $BadClus (перечень плохих кластеров) и т. д.

Первые четыре записи настолько важны, что продублированы в специальном $MFTMirr-файле, находящемся приблизительно посередине диска (точное расположение хранится в boot-секторе по смещению 38h байт от его начала). Вопреки своему названию, $MFTMirr – это отнюдь не зеркало всего $MFT-файла, это всего лишь копия первых четырех элементов.

Записи с 12 по 15 помечены как используемые, но в действительности же они пусты (как нетрудно догадаться, это задел на будущее). Записи с 16 по 23 не задействованы и честно помечены как неиспользуемые.

Начиная с 24-й записи располагаются пользовательские файлы и каталоги. Четыре метафайла, появившихся в W2K – $ObjId, $Quota, $Reparse и $UsnJrnl, могут располагаться в любой записи, номер которой равен 24 или больше (как мы помним, номера файловых записей отсчитываются начиная с нуля).

Для знакомства с MFT запустим DiskExplorer от Runtime Software, не забывая о том, что он требует прав администратора, в меню «File» найдем пункт «Drive» и в открывшемся диалоговом окне выберем логический диск, который мы хотим редактировать. Затем в меню «Goto» выберем пункт «Mft», заставляя DiskExplorer перейти к MFT, автоматичес-

Полезный совет

Как быстро узнать тип текущего раздела – FAT или NTFS? Да очень просто – достаточно попробовать создать в его корневом каталоге файл $mft – если он будет создан успешно, то это FAT и соответственно наоборот. Если файл $Extend будет создан успешно, то версия файловой системы 3.0 или выше.

ки меняя режим отображения на наиболее естественный (см. рис. 3).

Как вариант можно нажать <F6> (View as File Entry) и промотать несколько первых секторов клавишей <Page Down>.

Для каждого из файлов DiskExplorer сообщает:

- номер сектора, к которому данная файловая запись принадлежит (обратите внимание, что номера секторов монотонно увеличиваются на 2, подтверждая тот факт, что размер одной файловой записи равен 1 Кбайту, однако вы можете столкнуться и с другими значениями). Для удобства информация отображается сразу в двух системах исчисления – шестнадцатеричной и десятичной;
- основное имя файла/каталога (т.е. имя файла из заголовка файловой записи, некоторые файлы имеют несколько альтернативных имен). Если имя файла/каталога зачеркнуто, значит, он был удален, но соответствующая ему файловая запись все еще цела. Чтобы извлечь файл с диска (не важно, удаленный или нет) подведите к нему курсор и нажмите <Ctrl-T> для просмотра его содержимого в шестнадцатеричном виде или <Ctrl-S> для сохранения файла на диск. То же самое можно сделать и через контекстное меню (раздел «recovery»). При нажатии на <Ctrl-C> в буфер обмена копируется последовательность кластеров, занятых файлом (например, «DISKEXPL:K:1034240-1034240»).
- тип файловой записи – файл это или каталог?
- атрибуты файла/каталога – a = архивный файл, r = только на чтение, h = скрытый, s = системный, l = метка тома, d = каталог, c = сжатый файл;
- размер файла в байтах в десятичной системе исчисления (не для каталогов!);
- дату и время модификации файла/каталога;
- номер первого кластера файла/каталога (или «resident» для полностью резидентных файлов/каталогов);
- перечень типов NTFS-атрибутов, имеющихся у файла/каталога, записанных в шестнадцатеричной нотации (обычно эта строка имеет вид 10 30 80 – атрибут стандартной информации, атрибут имени и атрибут данных файла);
- индекс файловой записи в MFT, выраженный в шестнадцатеричной и десятичной системах исчисления и следующий за словом «No:» (сокращение от Number – номер);
- индекс файловой записи материнского каталога, выраженный в шестнадцатеричной и десятичной системах исчисления (5h – если файл принадлежит к корневому каталогу). Для быстрого перемещения по файловым записям выберите в меню «Goto» пункт «Mft no» и введите требуемый индекс в шестнадцатеричной или десятичной форме;
- для нерезидентных файлов/каталогов – перечень кластеров, занятых файлов в недекодированном виде (а зря – могли бы и декодировать!). Схема кодирования кластеров будет описана в следующей статье.

Прежде чем продолжать чтение статьи, попробуйте поэкспериментировать с $MFT-файлами (особенно фрагментированными). Посмотрите, как создаются и удаляются записи из MFT. Лучше всего это делать на диске, содержащем небольшое количество файлов/каталогов. Чтобы не форматировать логический диск, создайте виртуальный (благо количество оперативной памяти современных компьютеров это позволяет).

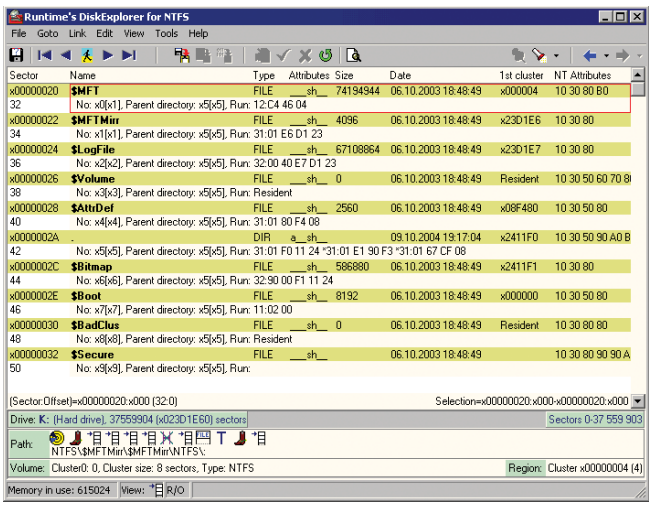

Рисунок 3. DiskExplorer отображает главную файловую запись в естественном виде

Файловые записи (FILE Record)

Благодаря наличию Disk Explorer от Runtime Software с файловыми записями практически никогда не приходится работать вручную, тем не менее знание их структуры нам не помешает.

Структурно файловая запись состоит из заголовка (header) и одного или нескольких атрибутов (attribute) произвольной длины, завершаемых маркером конца (end marker) – четырехбайтовым шестнадцатеричным значением FFFFFFFFh (см. листинг 1). Несмотря на то что количество и длина атрибутов меняется от одной файловой записи к другой, размер самой структуры FILE Record строго фиксирован и в большинстве случаев равен 1 Кбайту (это значение хранится в $boot-файле, не путать с boot-сектором!). Причем первый байт файловой записи всегда совпадает с началом сектора.

Если реальная длина атрибутов меньше размеров файловой записи, ее хвост просто не используется. Если же атрибуты не вмещаются в отведенное им пространство, создается дополнительная файловая запись (extra FILE Record), ссылающаяся на свою предшественницу.

Листинг 1. Структура файловой записи

```
FILE Record
    Header              ; заголовок
    Attribute 1         ; атрибут 1
    Attribute 2         ; атрибут 2
    ...                 ; …
    Attribute N         ; атрибут N
End Marker (FFFFFFFFh)  ; маркер конца
```

Первые четыре байта заголовка оккупированы магической последовательностью «FILE», сигнализирующей о том, что мы имеем дело с файловой записью типа FILE Record. При восстановлении сильно фрагментированного $MFT-файла это обстоятельство играет решающую роль, поскольку позволяет отличить сектора, принадлежа-

щие MFT, от всех остальных. Следом за сигнатурой идет 16-разрядный указатель, содержащий смещение последовательности обновления (update sequence, см. раздел «Последовательности обновления»). Под «указателем» здесь и до конца раздела подразумевается смещение от начала сектора, отсчитываемое от нуля и выраженное в байтах. В NT и W2K это поле всегда равно 002Ah, поэтому для поиска файловых записей можно использовать сигнатуру «FILE*\x00», что уменьшает вероятность ложных срабатываний. Правда, в XP и более старших системах последовательность обновления хранится по смещению 002Dh, и поэтому сигнатура приобретает следующий вид «FILE-\x00».

Размер заголовка (кстати сказать, варьирующийся от одной операционной системы к другой) в явном виде нигде не хранится, вместо этого в заголовке присутствует указатель на первый атрибут, содержащий его смещение в байтах относительно начала FILE Record, расположенный по смещению 14h байт от начала сектора. Смещения последующих атрибутов (если они есть) определяются путем сложения размеров всех предыдущих атрибутов (размер каждого из атрибутов содержится в его заголовке) со смещением первого атрибута. За концом последнего атрибута находится маркер конца – FFFFFFFFh.

В добавок к этому длина файловой записи хранится в двух полях – 32-разрядное поле реального размера (real size), находящееся по смещению 18h байт от начала сектора, содержит совокупный размер заголовка, всех его атрибутов и маркера конца, округленный по 8-байтной границе. 32-разрядное поле выделенного размера (allocated size), находящееся по смещению 1Ch байт от начала сектора, содержит действительный размер файловой записи в байтах, округленный по размеру сектора. Документация Linux-NTFS Project (версия 0.4) утверждает, что allocated size должен быть кратен размеру кластера, однако в действительности это не так. В частности, на моей машине allocated size равен четвертинке кластера.

16-разрядное поле флагов, находящееся по смещению 16h байт от начала сектора, в подавляющем большинстве случаев принимает одно из трех следующих значений: 00h – данная файловая запись не используется или ассоциированный с ней файл/каталог удален, 01h – файловая запись используется и описывает файл, 02h – файловая запись используется и описывает каталог.

64-разрядное поле, находящееся по смещению 20h байт от начала сектора, содержит индекс базовой файловой записи. Для первой файловой записи это поле всегда равно нулю, а для всех последующих, расширенных (extra) записей – индексу первой файловой записи. Расширенные файловые записи могут находиться в любых частях MFT, не обязательно рядом с основной записью. А раз так, необходим какой-то механизм, обеспечивающий быстрый поиск расширенных файловых записей, принадлежащих данному файлу (просматривать весь MFT целиком не предлагать). И этот механизм основан на ведении списков атрибутов $ATTRIBUTE_LIST.

Список атрибутов представляет собой специальный атрибут, добавляемый к первой файловой записи и содержащий индексы расширенных записей.

Остальные поля заголовка файловой записи не столь важны и поэтому здесь не рассматриваются. При необходимости обращайтесь к документации «Linux-NTFS Project».

Таблица 3. Структура заголовка файловой записи (FILE Record)

Смещение	Размер	ОС	Описание	
00h	4	любая	Сигнатура (magic number) 'FILE'	
04h	2	любая	Смещение номера последовательности обновления (update sequence number)	
06h	2	любая	Размер в словах номера последовательности обновления и массива обновления (Update Sequence Number & Array), условно (S)	
08h	8	любая	Номер последовательности файла транзакций ($LogFile Sequence Number или сокращенно LSN)	
10h	2	любая	Номер последовательности (sequence number)	
12h	2	любая	Счетчик жестких ссылок (hard link)	
14h	2	любая	Смещение первого атрибута (attribute)	
16h	2	любая	Флаги (flags)	
			Значение	**Описание**
			0x00	Файловая запись не используется
			0x01	Файловая запись используется и описывает файл (file)
			0x02	Файловая запись используется и описывает каталог (directory)
			0x04	Только Билл Гейтс знает
			0x08	Только Билл Гейтс знает
18h	4	любая	Реальный размер (real size) файловой записи	
1Ch	4	любая	Выделенный размер (allocated size) файловой записи	
20h	8	любая	Ссылка (file reference) на базовую файловую запись (base FILE record) или ноль, если данная файловая запись базовая	
28h	2	любая	Идентификатор следующего атрибута (next attribute ID)	
2Ah	2	XP	Для выравнивания	
2Ch	4	XP	Индекс данной файловой записи (number of this MFT record)	
	2	любая	Номер последовательности обновления (update sequence number)	
	2S-2	любая	Массив последовательности обновления (update sequence array)	

Последовательности обновления (update sequence)

Будучи очень важными компонентами файловой системы, $MFT, INDEX и $LogFile нуждаются в механизме контроля целостности своего содержимого. Традиционно для этого используются ECC/EDC-коды, однако во времена проектирования NTFS процессоры были не настолько быстрыми, как теперь, и расчет корректирующих кодов занимал значительное время, существенно снижающее производительность файловой системы. Поэтому от них пришлось отказаться. Вместо этого разработчики NTFS применили так называемые последовательности обновления (update sequence), так же называемые fix-up.

В конец каждого из секторов, слагающих файловую запись (INDEX Record, RCRD Record или RSTR Record) записывается специальный 16-байтовый номер последовательности обновления (update sequence number), дублируемый в заголовке файловой записи. При каждой операции чтения два последних байта сектора сверяется с соответствующим полем заголовка, и если NTFS-драйвер обнаруживает расхождение, данная файловая запись считается недействительной.

Основное назначение последовательностей обновления – защита от «обрыва записи». Если в процессе записи сектора на диск исчезнет питающее напряжение, может случиться так, что половина файловой записи будет успешно записана, а другая половина – сохранит прежнее содержимое (файловая запись, как мы помним, обычно состоит из двух секторов). После восстановления питания драйвер

файловой системы не может уверенно сказать – была ли файловая запись записана целиком или нет. Вот тут-то последовательности обновления и выручают! При каждой перезаписи сектора update sequence увеличивается на единицу, и потому, если произошел обрыв записи, значение последовательности обновления находящейся в заголовке файловой записи, не будет совпадать с последовательностью обновления, расположенной в конце сектора.

Оригинальное содержимое, расположенное «под» последовательностью обновления, хранится в специальном массиве обновления (update sequence array), находящимися в заголовке файловой записи непосредственно за концом update sequence number. Для восстановления файловой записи в исходный вид мы должны извлечь из заголовка указатель на update sequence number (он хранится по смещению 04h байт от начала заголовка) и сверить лежащее по этому адресу 16-байтное значение с последним словом каждого из секторов, слагающих файловую запись (INDEX Record, RCRD Record или RSTR Record). Если они не совпадут, значит соответствующая структура данных повреждена и использовать ее следует с очень большой осторожностью (а на первых порах лучше не использовать вообще).

По смещению 006h от начала сектора находится 16-разрядное поле, хранящее совокупный размер номера последовательности обновления вместе с массивом последовательности обновления:

```
(sizeof(update sequence number) + sizeif(update sequence array))
```

выраженный в словах (не в байтах!). Поскольку размер номера последовательности обновления всегда равен одному слову, то размер массива последовательности обновления, выраженный в байтах, равен:

```
(update sequence number & update sequence array - 1)*2
```

Соответственно смещение массива оригинального содержимого равно:

```
(offset to update sequence number) + 2
```

В NT и W2K update sequence number всегда располагается по смещению 2Ah от начала FILE Record Header/INDEX Header, а update sequence array – по смещению 2Ch. В XP же и более старших системах – по смещениям 2Dh и 2Fh соответственно.

Первое слово массива последовательности обновления соответствует последнему слову первого сектора FILE Record/INDEX. Второе – последнему слову второго сектора и т. д. Для восстановления сектора в исходный вид мы должны вернуть все элементы массива последовательности обновления на их законные места (естественно, модифицируется не сам сектор, а его копия в памяти).

Продемонстрируем это на следующем примере:

Листинг 2. Оригинальная файловая запись до восстановления

```
--> начало первого сектора FILE Record
00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w→♦☻
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☺ ☻ 0 ☺ (☻   ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11            ♠ ♠   G◄
…
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 06 00                  ♠
<-- конец первого сектора FILE Record
…
000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 06 00  •Ia♪ ○  yyyyBy♠
<-- конец второго сектора FILE Record
```

Сигнатура «FILE» указывает на начало файловой записи. А раз так, по смещению 04h байт будет расположен 16-разрядный указатель на номер последовательности обновления. В данном случае он равен 002Ah. OK! Переходим по смещению 002Ah и видим, что здесь лежит слово 0006h. Перемещаемся в конец сектора и сверяем его с последними двумя байтами. Как и предполагалось, они совпадают. Повторяем ту же самую операцию со следующим сектором. Собственно говоря, количество секторов может и не равняться двум. Чтобы не гадать на кофейной гуще, необходимо извлечь 16-разрядное значение, расположенное по смещению 06h от начала файловой записи (в данном случае оно равно 0003h) и вычесть из него единицу. Действительно, получается два (сектора).

Теперь нам необходимо найти массив последовательности обновления, хранящий оригинальное значение последнего слова каждого из секторов. Смещение массива обновления рассчитывается по следующей формуле: offset of (update sequense number) + 2, т.е. в данном случае 002Ah + 02h == 002Ch. Извлекаем первое слово (в данном случае равное 00h 00h) и записываем его в конец первого сектора, затем делаем то же самое со вторым.

В результате чего восстановленный сектор будет выглядеть так:

Листинг 3. Восстановленная файловая запись

```
--> начало первого сектора FILE Record
00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w→♦☻
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☺ ☻ 0 ☺ (☻   ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11            ♠ ♠   G◄
…
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00                  ♠
<-- конец первого сектора FILE Record
…
000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 47 11  •Ia♪ ○  yyyyBy♠
<-- конец второго сектора FILE Record
```

Заключение

Вооруженные джентльменским набором знаний об устройстве файловой записи, в следующей статье этого цикла мы сможем погрузиться внутрь атрибутов, рассмотрев, как хранятся резидентные и нерезидентные потоки данных на жестком диске.

Литература:

1. Касперски К. Востановление данных на NTFS-разделах. – Журнал «Системный администратор», №9, сентябрь 2004г.
2. Касперски К. Востановление данных на NTFS-разделах. Часть 2. – Журнал «Системный администратор», №10, октябрь 2004г.

Внимание!

FILE Record, INDEX Record, RCRD Record или RSTR Record искажены последовательностями обновления и в обязательном порядке должны быть восстановлены перед их использованием, в противном случае вместо актуальных данных вы получите мусор!