

The NTFS File System

Sleuth Kit Implementation Notes (SKINs)

www.sleuthkit.org

Brian Carrier

Last Updated: June 2003

Introduction

The NTFS file system is used in all critical Microsoft Windows systems. It is an advanced file system that makes it different from the UNIX file systems that the original TCT was designed for. This document gives a quick overview of NTFS and how it was implemented. The biggest difference is the use of Alternate Data Streams (ADS) when specifying a meta data structure.

The Sleuth Kit allows one to investigate an NTFS image in the same ways as any UNIX image, including:

- Creation of ASCII timeline of file activity
 - Cluster analysis and mapping between clusters and MFT entries
 - MFT analysis and mapping between MFT entries and file names
 - File and directory level analysis including deleted files
-

NTFS Overview

This provides a quick introduction to the NTFS file system. The terms used are different then with other file systems. For a full overview of the file system, refer to the "Inside Windows 2000" book by Solomon and Russinovich and for details of the file system structures, refer to the NTFS Source Forge project at:

<http://linux-ntfs.sourceforge.net/ntfs/index.html>

MFT

The Master File Table (MFT) contains entries that describe all system files, user files, and directories. The MFT even contains an entry (#0) that describes the MFT itself, which is how we determine its current size. Other system files in the MFT include the Root Directory (#5), the cluster allocation map, Security Descriptors, and the journal.

MFT ENTRIES

Each MFT entry is given a number (similar to inode numbers in UNIX). The user files and directories start at MFT #25. The MFT entry contains a list of attributes. Example attributes include "Standard Information" which stores data such as MAC times, "File Name" which stores the file or directories name(s), \$DATA which stores the actual file content, or "Index Alloc" and "Index Root" which contain directory contents stored in a B-Tree.

Each type of attribute is given a numerical value and more than one instance of a type can exist for a file. The "id" value for each attribute allows one to specify an instance. A given file can have more than one "\$Data" attribute, which is a method that can be used to hide data from an investigator. To get a mapping of attribute type values to name, use the 'fsstat' command. It displays the contents of the \$AttrDef system file.

Each attribute has a header and a value and an attribute is either resident or non-resident. A resident attribute has both the header and the content value stored in the MFT entry. This only works for attributes with a small value (the file name for example). For larger attributes, the header is stored in the MFT entry and the content value is stored in Clusters in the data area. A Cluster in NTFS is the same as FAT, it is a consecutive group of sectors. If a file has too many different attributes, an "Attribute List" is used that stores the other attribute headers in additional MFT entries.

FILES

Files in NTFS typically have the following attributes:

- \$STANDARD_INFORMATION (#16): Contains MAC times, security ID, Owners ID, permissions in DOS format, and quota data.
- \$FILE_NAME (#48): Contains the file name in UNICODE, as well as additional MAC times, and the MFT entry of the parent directory.
- \$OBJECT_ID (#64): Identifiers regarding the files original Object ID, its birth Volume ID, and Domain ID.
- \$DATA (#128): The raw content data of the file.

When a file is deleted, the IN_USE flag is cleared from the MFT entry, but the attribute contents still exist.

DIRECTORIES

Directories in NTFS are indexed to make finding a specific entry in them faster. By default, they are stored in a B-Tree sorted in alphabetical order. There are two attributes that describe the B-Tree contents. Directories in NTFS typically have the following attributes:

- `$STANDARD_INFORMATION` (#16): See above
- `$FILE_NAME` (#48): See above
- `$OBJECT_ID` (#64): See above
- `$INDEX_ROOT` (#144): The root of the B-Tree. The `$INDEX_ROOT` value is one more more "Index Entry" structures that each describe a file or directory. The "Index Entry" structure contains a copy of the `$FILE_NAME` attribute for the file or sub-directory.
- `$INDEX_ALLOCATION` (#160): The sub-nodes of the B-Tree. For small directories, this attribute will not exist and all information will be saved in the `$INDEX_ROOT` structure. The content of this attribute is one or more "Index Buffers". Each "Index Buffer" contains one or more "Index Entry" structures, which are the same ones found in the `$INDEX_ROOT`.
- `$BITMAP` (#176): This describes which structures in the B-Tree are being used.

When files are deleted from a directory, the tree node is removed and the tree is resorted. Therefore, the "Index Entry" for the deleted file maybe written over when the tree is resorted. This is different than what is usually seen with UNIX and FAT file systems, which always have the original name and structure until a new file is created. Also, when the tree is resorted, a file that is on the bottom of the tree can be moved up and a deleted file name will exist for the original location (even though it was never deleted by a user).

Using The Sleuth Kit with NTFS

The Sleuth Kit allows one to view all aspects of the NTFS structure. The biggest difference with using The Sleuth Kit with NTFS instead of UNIX file systems is the attributes. With UNIX you only need to reference the inode number because there is only one piece of content for the file. With NTFS, one can either specify just the MFT number and the default data attribute is used or the type can be specified by adding it to the end of the MFT entry, 36-128 for example. If more than one attribute of the same type exists, then the id can be used after the type, 36-128-5 for example.

All Sleuth Kit tools can take MFT values in any of the above formats and output from the tools will also be in one of the above formats. For example, the 'istat' tool will list all attributes a file has. To get the details of MFT entry 49, use:

```
# istat -f ntfs ntfs.dd 49
MFT Entry: 49
```

```
Sequence: 2
Allocated
UID: 0
DOS Mode: File
Size: 15
Links: 1
Name: multiple.txt
```

```
$STANDARD_INFORMATION Times:
```

```
File Modified: Mon Nov 5 19:58:27 2001
MFT Modified: Mon Nov 5 19:58:27 2001
Accessed: Mon Nov 5 19:58:27 2001
```

```
$FILE_NAME Times:
```

```
Created: Mon Nov 5 19:57:29 2001
File Modified: Mon Nov 5 19:57:29 2001
MFT Modified: Mon Nov 5 19:57:29 2001
Accessed: Mon Nov 5 19:57:29 2001
```

```
Attributes:
```

```
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-2) Name: N/A Resident size: 90
Type: $OBJECT_ID (64-3) Name: N/A Resident size: 16
Type: $DATA (128-1) Name: $Data Resident size: 15
Type: $DATA (128-5) Name: overhere Resident size: 26
```

We see that it has 5 attributes, all of them are resident (notice the small sizes). Two of the attributes are \$DATA attributes (128-1 and 128-5). The full name of 128-1 is 'multiple.txt' and the full name of 128-5 is 'multiple.txt:overhere'.

The following command would display the default data attribute (128-1):

```
# icat -f ntfs ntfs.dd 49
```

The following is the same:

```
# icat -f ntfs ntfs.dd 49-128-1
```

The following displays the other data stream:

```
# icat -f ntfs ntfs.dd 49-128-5
```

As an additional example, the raw format of the \$FILE_NAME attribute can be viewed using:

```
# icat -f ntfs ntfs.dd 49-48-2
```

The output of the above command would be a combination of UNICODE characters and other binary data (I would recommend just using the output of the istat command for this type of data).

The output of the 'fls' command is similar:

```
# fls -f ntfs ntfs.dd
<...>
r/r 48-128-1: test-1.txt
r/r 49-128-1: multiple.txt
r/r 49-128-5: multiple.txt:NEW
r/r 50-128-1: test-2.txt
<...>
```

This allows you to easily identify all data streams.

Note that Autopsy can automate this process for you and allows you to view all attributes.

<http://www.sleuthkit.org/autopsy>

What The Sleuth Kit Cannot Currently Do

There are a few things that The Sleuth Kit is not yet able to do with NTFS:

- UNICODE support for file and directory names is limited to ASCII values.
 - The Security Descriptors are not yet analyzed. Therefore, the exact ACLs of the object can not be displayed.
 - Directories that are indexed by a descriptor other than the file name, are not supported.
 - Compressed & encrypted files are not supported
-

Copyright © 2002-2003 by Brian Carrier. All Rights Reserved