

Lecture 5 File Systems: NTFS

CS 414 Digital Forensics

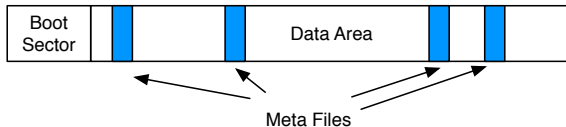
Dr Changyu Dong

`<changyu.dong@cis.strath.ac.uk>`

Department of Computer and Information Sciences
University of Strathclyde

- NTFS is a file system designed for the NT OS families
- Much more complex than FAT
- Designed for reliability , scalability and security

NTFS Volume Layout



- Boot Sector: starts at sector 0 and can be up to 16 sectors, gives the starting location of the MFT, cluster size (1 to 128 sectors, but typically 8), size of each MFT entry (usually 1024 bytes)
- Data Area: All sectors after the boot sector are data area. File system information are stored in meta files.

File System Meta Data Files

- Name begins with a \$
- All in the root directory and are hidden
 - 0 \$MFT: The Master File Table . Holds information about all files and directories on the volume.
 - 1 \$MFTMirr: a mirror record of the MFT. Read if the first MFT record is corrupt
 - 2 \$LogFile: log file used for file recovery
 - 3 \$Volume: volume information e.g. label, identifier
 - 4 \$AttrDef: attributed information
 - 6 \$Bitmap: bitmap allocation status of each cluster
 - 7 \$Boot: boot sector and boot code
 - 8 \$BadClus: list of all bad clusters
 - 9 \$Secure: security and access control
 - 10 \$Upcase: all uppercase unicode characters
 - 11 \$Extend: a directory for optional extensions

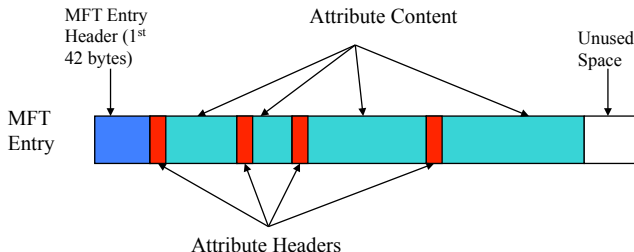
A Few Differences Between NTFS and FAT

- In FAT, the boot sector is not a file, in NTFS, the boot sector is a file (\$Boot)
- In FAT, cluster allocation is tracked by the FAT table, in NTFS, it is tracked by a \$Bitmap file and \$Badclus file.

MFT Entries

- The MFT contains entries, a.k.a. file records
- Each entry is 1 KB in size (default) and is numbered
- First 42 bytes contain 12 fields, the remaining 982 bytes are unstructured and can be filled with attributes
- Each entry corresponds to a file or a directory
- A file or directory may have multiple entries to store information about the file or directory
- Each entry has a 48-bit address which is based on its sequential order in the MFT. For example, 0 means it is the first entry in the MFT, 10 means it is the 11th entry.
- The first 16 entries are reserved for system meta data files and directory, e.g.
 - Entry 0 is \$MFT
 - Entry 5 is the root directory

MFT Entry Structure



- MFT entry = MFT header + attributes
- An attribute consists of a header and its content.
- A file or directory can have up to 65,536 attributes
 - That's why you might need more than one entries for a file or directory.
- Each attribute has a type

Attribute Type Examples

- \$FILE_NAME (48): file name in unicode, a reference to the parent directory, and the last accessed, written and created times
- \$STANDARD_INFORMATION (16): general information, e.g. flags, the last accessed, written and created times and the owner and security ID
- \$DATA (128): file content. Small files (<~700 bytes) this is resident, large files this is non-resident.
- \$INDEX_ROOT (144): Directory only. The root node of an index tree (for files and subdirectories in it)
- \$INDEX_ALLOCATION (160): Nodes of an index tree rooted in \$INDEX_ROOT attribute.
- \$BITMAP (176): a bitmap for the \$MFT file and for indexes

\$Bitmap File and \$BITMAP Attribute

- A meta file called \$Bitmap is used to track cluster allocation
 - Before write data, use \$Bitmap file to find unallocated cluster
- \$MFT file uses \$BITMAP attribute to track MFT entry allocation
 - Before create a file (directory), use \$BITMAP attribute to find an unallocated entry in \$MFT file.
- Directories use \$BITMAP attribute to track which index entries are in use

Attribute Headers

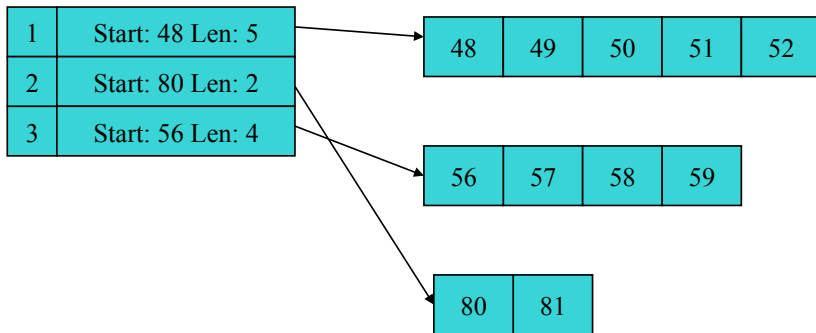
- Contain the type of the attribute, size and name
- Also flag whether the content is compressed or encrypted
- And whether the attribute content is stored within the entry or stored externally (resident or non-resident).
- An MFT entry can have multiple attributes of the same type
 - an entry wide unique ID is used to distinguish them
 - NTFS Files (and directories) can have multiple \$DATA attributes. These are called Alternate Data Streams (ADS).
 - One could have a file of 0b size but there can be any number of bytes hidden in a separate data stream for that file.
 - Windows Explorer only displays the primary data stream

Attribute Content

- Various format and size
- If the attribute content is stored in the MFT entry, then the attribute is a **resident** attribute
- If the attribute content is too large to be stored in the MFT entry, then it can be stored in external clusters. Then the attribute is a **non-resident** attribute.
- The header identifies whether this is resident or non-resident attribute
- For non-resident attribute, the offset of the cluster runlist is given in the attribute header. A starting and last virtual cluster number (VCN) are also recorded in the attribute header
 - Virtual clusters are logical cluster numbers independent from their physical location on disk. In most cases, the starting VCN is zero and the last one is the number of clusters occupied by the attribute content minus one.

Large Non-resident Attribute

- The content of a non-resident attribute can be very large
 - The file content
 - The index of a directory
- Need to be stored in multiple external clusters
- Cluster runlists are used to record cluster allocation
 - The start cluster address and the run length (how many cluster in this run)



Attribute Structure (Resident)

Offset	Size, bytes	Value	Description
00h	4		Attribute type (e.g., 10h, 60h, B0h)
04h	4		Attribute length including the header
08h	1	00h	Non-resident flag
09h	1	N	Attribute's name length (0 if the attribute is nameless)
0Ah	2	18h	Name offset (0 if the attribute is nameless)
0Ch	2	0001h 4000h 8000h	Flags: - compressed attribute - encrypted attribute - sparse attribute
0Eh	2		Attribute ID
10h	4	L	Length of attribute body without header
14h	2	2N+18h	Attribute's body offset
16h	1		Index flag
17h	1	00h	For alignment
18h	2N	UNICODE	Attribute name (if any)
2N+18h	L		Attribute body

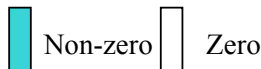
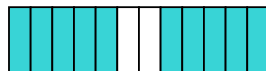
Attribute Structure (Non-resident)

Offset	Size, bytes	Value	Description
00h	4		Attribute type (e.g., 0x20, 0x80)
04h	4		Attribute length including the header
08h	1	01h	Non-resident flag
09h	1	N	Attribute's name length (0 if the attribute is nameless)
0Ah	2	40h	Name offset (0 if the attribute is nameless)
0Ch	2	0001h 4000h 8000h	Flags: - compressed attribute - encrypted attribute - sparse attribute
0Eh	2		Attribute ID
10h	8		Starting VCN
18h	8		Last VCN
20h	2	2N+40h	Offset of data runs list
22h	2		Compression unit size rounded up to 4 bytes
24h	4	00h	For alignment
28h	8		Allocated size rounded up to cluster size
30h	8		Real size
38h	8		Initialized data size of the stream
40h	2N	UNICODE	Attribute name (if any)
2N+40h	..		List of data runs

Sparse Attributes

- NTFS can save disk space by making non-resident \$DATA attribute sparse
- Clusters that contains all zeros are not written to disk
- A special run is created for the zero clusters: no start clusters, only run length
- A flag is set to indicate the attribute is sparse,

1	Start: 48 Len: 5
2	Start: _ Len: 2
3	Start: 56 Len: 4

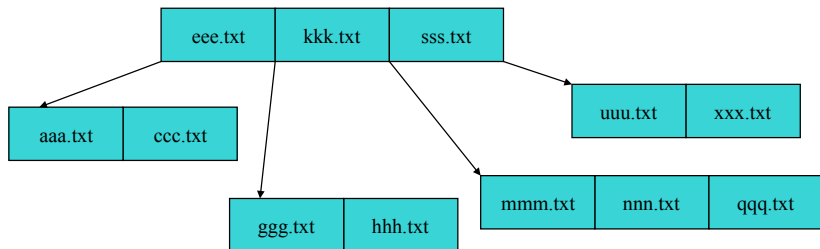


Compressed and Encrypted Attribute

- File system level compression and encryption
- Apply to the \$DATA attribute
- Cannot have both at the same time
- Can be on directory or file level



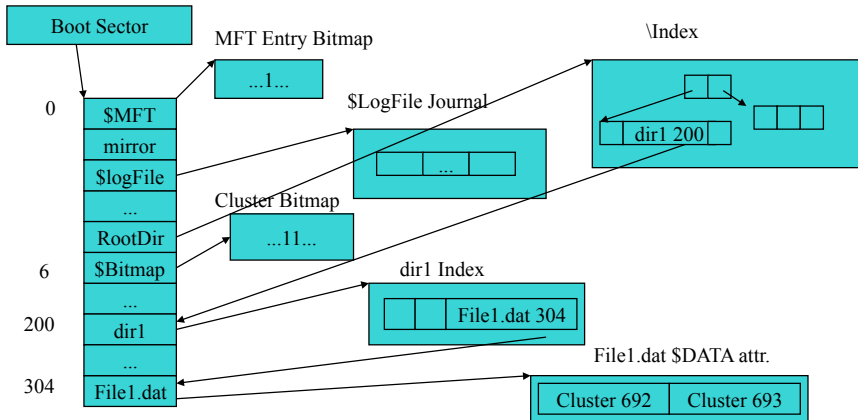
- NTFS uses balanced-tree indexes to speed up file look up
- Recall that FAT uses linear search to locate a file in a directory (from the first directory entry until encounter the file name)
- $O(\log n)$ vs $O(n)$
- The following shows a B-Tree sorted under file names, each node also contains the MFT address of the file.



File Allocation Example

- Create the file `\dir1\file1.dat` where `filesize = 4,000 bytes` and `clustersize = 2, 048 bytes`
 - Read and process boot sector to determine cluster size, location of MFT
 - Read 1st entry of MFT which is `$MFT` file and determine layout of MFT
 - Allocate an MFT entry for the new file by using `$BITMAP` to find an unused entry, say 304, and set corresponding bit to 1
 - Go to entry 304, create `$STANDARD_INFORMATION` and `$FILE_NAME` and set times to current time
 - Read `$Bitmap` file to allocate 2 clusters for the file (clusters 692 and 693) and write file content, update `$BITMAP` file.
 - Then need to update the index of `dir1`. First find the MFT entry for the root directory (5), then MFT entry for `dir1` by reading the `$INDEX_ROOT` and `$INDEX_ALLOCATION` and traverse sorted tree. Say the entry for `dir1` is 200
 - Go to MFT entry 200 and update the index of `dir1` to add a index entry to MFT entry 304 (the entry for `file1.txt`).
 - `$LogFile` is updated when necessary.

File Allocation Example



File Delete Example

- Delete the file `\dir1\file1.dat`
 - Read and process boot sector to determine cluster size, location of MFT
 - Read 1st entry of MFT which is \$MFT file and determine layout of MFT
 - Then need to find dir1. First find the MFT entry for the root directory (5), then MFT entry for dir1 by reading the \$INDEX_ROOT and \$INDEX_ALLOCATION and traverse sorted tree. The entry for dir1 is 200
 - Go to MFT entry 200 and search the index of dir1 for file1.dat. The MFT address for the file is found to be entry 304
 - Remove the node from the index
 - Go to MFT entry 304, unallocate the entry by cleaning the in-use flag, set the MFT entry bitmap to 0 for this entry
 - Unallocate disk clusters by setting the cluster bits for cluster 692, 693 in the \$Bitmap file to 0
 - \$LogFile is updated and carry out other house keeping tasks when necessary .

- When a file is deleted, the name is removed from the parent directory index, the MFT entry is unallocated and the clusters are unallocated
- But the entry and clusters are not cleared
- Read the unallocated entries in the MFT and you can recover the file (if the entry and clusters are not reallocated)
 - Small files: the content is resident
 - Large files: need to read external clusters
- The entire path can be determined by the parent directory reference in \$FILE_NAME attribute

```
sansforensics@SIFT-Workstation:~$ fsstat -o 128 ntfs_dd
```

FILE SYSTEM INFORMATION

```
-----  
File System Type: NTFS  
Volume Serial Number: 9E2AD4CD2AD4A395  
OEM Name: NTFS  
Volume Name: New Volume  
Version: Windows XP
```

METADATA INFORMATION

```
-----  
First Cluster of MFT: 17237  
First Cluster of MFT Mirror: 2  
Size of MFT Entries: 1024 bytes  
Size of Index Records: 4096 bytes  
Range: 0 - 1792  
Root Directory: 5
```

CONTENT INFORMATION

```
-----  
Sector Size: 512  
Cluster Size: 4096  
Total Cluster Range: 0 - 51710  
Total Sector Range: 0 - 413694
```

\$AttrDef Attribute Values:

```
$STANDARD_INFORMATION (16)  Size: 48-72  Flags: Resident  
$ATTRIBUTE_LIST (32)       Size: No Limit  Flags: Non-resident  
$FILE_NAME (48)            Size: 68-578   Flags: Resident,Index  
$OBJECT_ID (64)            Size: 0-256    Flags: Resident  
$SECURITY_DESCRIPTOR (80)   Size: No Limit  Flags: Non-resident  
$VOLUME_NAME (96)          Size: 2-256    Flags: Resident  
$VOLUME_INFORMATION (112)   Size: 12-12   Flags: Resident  
$DATA (128)                Size: No Limit  Flags:  
$INDEX_ROOT (144)          Size: No Limit  Flags: Resident  
$INDEX_ALLOCATION (160)     Size: No Limit  Flags: Non-resident  
$BITMAP (176)              Size: No Limit  Flags: Non-resident
```

```
sansforensics@SIFT-Workstation:~$ fls -o 128 ntfs_dd
r/r 4-128-4:      $AttrDef
r/r 8-128-2:      $BadClus
r/r 8-128-1:      $BadClus:$Bad
r/r 6-128-4:      $Bitmap
r/r 7-128-1:      $Boot
d/d 11-144-4:     $Extend
r/r 2-128-1:      $LogFile
r/r 0-128-1:      $MFT
r/r 1-128-1:      $MFTMirr
d/d 35-144-1:     $RECYCLE.BIN
r/r 9-128-8:      $Secure:$SDS
r/r 9-144-11:     $Secure:$SDH
r/r 9-144-14:     $Secure:$SII
r/r 10-128-1:     $UpCase
r/r 3-128-3:      $Volume
d/d 38-144-7:     backup
d/d 39-144-7:     pdf
-/d * 1776-144-1:      MSI26e58.tmp
d/d 1792:         $OrphanFiles
```

```
sansforensics@SIFT-Workstation:~$ lsstat -o 128 ntfs_dd 39-144-7
MFT Entry Header Values:
Entry: 39          Sequence: 1
$LogFile Sequence Number: 1073346
Allocated Directory
Links: 1

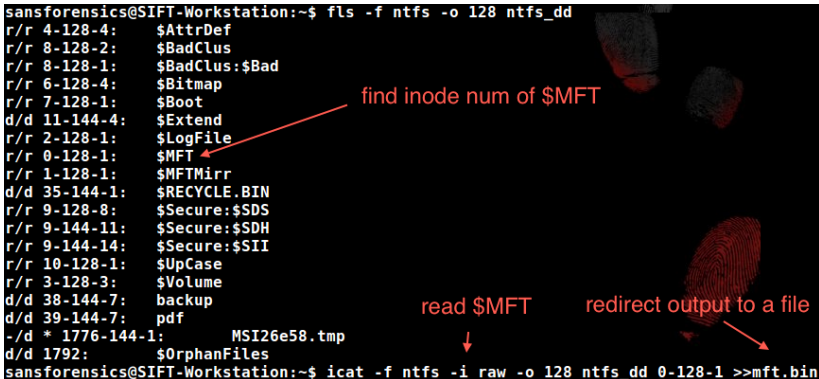
$STANDARD_INFORMATION Attribute Values:
Flags:
Owner ID: 0
Security ID: 264  ()
Created:      Thu Aug  2 11:07:18 2012
File Modified: Thu Aug  2 11:11:01 2012
MFT Modified:  Thu Aug  2 11:11:01 2012
Accessed:     Thu Aug  2 11:11:01 2012

$FILE_NAME Attribute Values:
Flags: Directory
Name: pdf
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      Thu Aug  2 11:07:18 2012
File Modified: Thu Aug  2 11:07:18 2012
MFT Modified:  Thu Aug  2 11:07:18 2012
Accessed:     Thu Aug  2 11:07:18 2012

Attributes:
Type: $STANDARD_INFORMATION (16-0)  Name: N/A  Resident  size: 72
Type: $FILE_NAME (48-4)  Name: N/A  Resident  size: 72
Type: $INDEX_ROOT (144-7)  Name: $I30  Resident  size: 56
Type: $INDEX_ALLOCATION (160-5)  Name: $I30  Non-Resident  size: 4096  init_s
size: 4096
42
Type: $BITMAP (176-6)  Name: $I30  Resident  size: 8
```


Extract MFT Using icat

- **icat** read the content of a file (specified using a inode number) from an image and output to stdout.
- `icat [-f fstype] [-i imgtype] [-b dev_sector_size] [-o imgoffset] image [images] inode`
- Can be used to extract MFT content



```
sansforensics@SIFT-Workstation:~$ fls -f ntfs -o 128 ntfs_dd
r/r 4-128-4:  $AttrDef
r/r 8-128-2:  $BadClus
r/r 8-128-1:  $BadClus:$Bad
r/r 6-128-4:  $Bitmap
r/r 7-128-1:  $Boot
d/d 11-144-4:  $Extend
r/r 2-128-1:  $LogFile
r/r 0-128-1:  $MFT
r/r 1-128-1:  $MFTMirr
d/d 35-144-1:  $RECYCLE.BIN
r/r 9-128-8:  $Secure:$SDS
r/r 9-144-11: $Secure:$SDH
r/r 9-144-14: $Secure:$SII
r/r 10-128-1: $UpCase
r/r 3-128-3:  $Volume
d/d 38-144-7: backup
d/d 39-144-7: pdf
-/d * 1776-144-1: MSI26e58.tmp
d/d 1792:     $OrphanFiles
sansforensics@SIFT-Workstation:~$ icat -f ntfs -i raw -o 128 ntfs_dd 0-128-1 >>mft.bin
```

find inode num of \$MFT

read \$MFT

redirect output to a file