Chapter 8
Deleted File Recovery in NTFS

Learning Objectives
The objectives of this chapter are to:

- Understand principles of file creation and file deletion in NTFS
- Manually recover the deleted files on an NTFS file system based on remaining metadata information

In this chapter, we continue our analysis of NTFS file system. Particularly, we study how to recover the deleted files on an NTFS file system based on remaining metadata information. Also, you will know the challenges facing data recover in NTFS.

8.1 NTFS Deleted Files Recovery

Since it was first introduced in the Windows NT operating system, NTFS has become the default file system for Microsoft operating systems, including Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, and the newest Windows 10 [1]. Due to the popularity of Windows today, it is very often to see the need for data recovery in computers using NTFS file system due to various reasons, such as accidental deletion of files from a forensics investigative perspective. Similar to FAT file system, when a file is deleted, it becomes inaccessible through regular ways available in the operating system. However, the file data itself is not being deleted. Further, some important file system metadata information about the deleted file is not completely wiped out. Unfortunately, recovering deleted files in NTFS could become very challenging because of several factors. One of them is the use of directory index, particularly an index of names of the files and sub-directories within a directory. It uses a B-tree data structure, where a tree node

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_8

contains many index entries (or keys in terms of B-tree). When a file is deleted from the directory, the index entry representing the file is removed and the B-tree has to be adjusted to preserve the B-tree properties if necessary. Therefore, the "Index Entry" for the deleted file may be overwritten when the tree is resorted. This is different than what is usually seen with other file systems, such as FAT file system, which always have the remaining original file name and structure untouched until a new file is created. In this chapter, we will focus on how to recover deleted files in NTFS file system using remaining file metadata information.

8.1.1 File Creation and Deletion in NTFS File Systems

For ease of presentation, we will always discuss the scenarios about files in the root directory. This is because as to files in the other directories, the only extra effort is that we would need to locate the directory which contains the intended file. This can be done by going through the entire file path, starting from the top-most directory (or root directory) to its next sub level until the directory containing the file is reached.

Also, none of special attributes for the file has been set, such as compression, sparse. NTFS utilizes a relational database called Master File Table (MFT) which is similar to FAT directory entries contains an entry for every file on the system. We assume that the location of MFT is already known to the Operating System here.

8.1.1.1 File Creation in NTFS File System (Fig. 8.1)

In an NTFS file system, the procedure for the file creation can be illustrated below:

1. Check if the file system has sufficient disk space to store the contents of the newly created file. If not, an "Insufficient disk space" error message appears and the file creation fails. Otherwise, a certain number of clusters will be allocated to the file, and their status becomes allocated and unavailable to other files or folders. It is worth pointing out that very small files can be wholly contained in the MFT entry allocated to the file, without the need for a separate cluster for its data and ensure fast retrieval times. This will be discussed later.
2. Locate an MFT entry that is currently available and allocate it for the new file.
3. Set the MFT entry as occupied and make it not available for others. Initialize it with standard MFT entry attributes, including $STANDARD_INFORMATION, $FILE_NAME, $DATA attribute, etc.
4. If the newly created file is very small, for example, less than 700 bytes, the file data will be wholly contained as the content of $DATA attribute. Then, save the file content into the $DATA attribute. In this case, $DATA attribute contains file data itself.

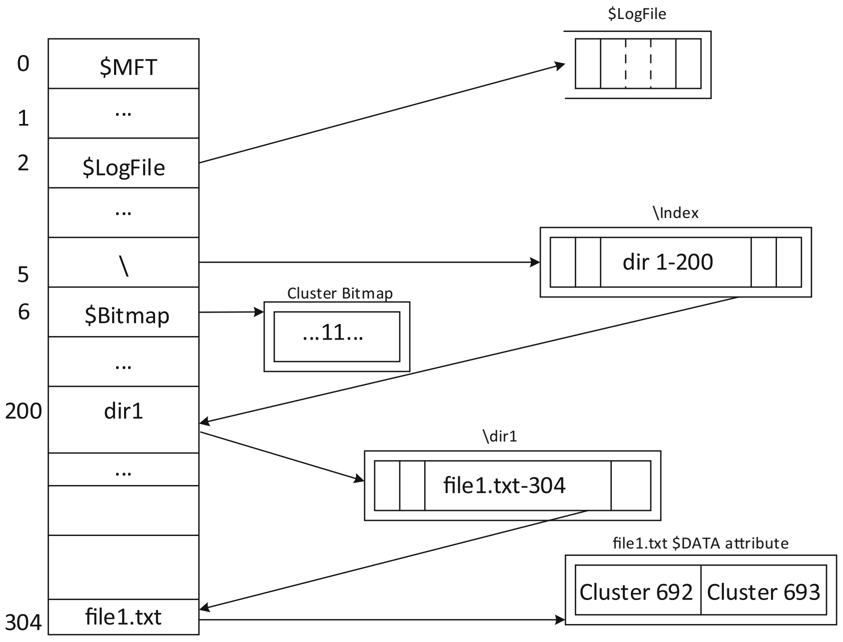


Fig. 8.1 File creation

However, with a larger file, disk space outside the MFT entry or clusters will be allocated to store file content. In this case, $DATA attribute shows where that data is located. As we discussed in previous chapter, NTFS uses data runs to specify the location of file data, where a data run contains starting address of cluster run and run length. Then, check MFT's $Bitmap to find free clusters needed for the file, using best-fit algorithm, and set corresponding $Bitmap bits to 1. Afterwards, write file content to clusters and update the content of $DATA attribute with data runs, which indicates where that data is located.

Note that NTFS allocates clusters of disk space to the file. If the file does not require the entire cluster for storage, the last cluster is left with unused extra space. For example, on a disk with a cluster size of 4 kilobytes and sector size of 512 bytes, clusters will always start at a sector number that is a multiple of 8. NTFS also performs pre-allocation of additional clusters for a file under certain conditions in advance to prevent anticipated future fragmentation as the file grows. This has an effect on fragmentation rates and may actually cause more harm than good on a system with low disk space.

5. Go to root directory (MFT entry 5). NTFS uses B-trees for directory indexing. Read index attributes of MFT entry 5, and traverse the B-tree to determine where the file should go according to its file name. Afterwards, create a new index entry

and add the new index entry to the index node (or INDX record) at the appropriate place alphabetically among the file names already there to maintain the ordering. The B-tree may need to be readjusted so as to preserve the B-tree properties.
6. As each step is taken, record action in $LogFile.

8.1.1.2 File Deletion (Fig. 8.2)

When a file is deleted in an NTFS file system, the operating system will perform the following operations:

1. Go to root directory (MFT entry 5) and read index attributes of MFT entry 5. Traverse the B-tree to locate the index entry representing the file to be deleted. Read the index entry and retrieve the number of the MFT entry number that represents the file. Afterwards, delete the index entry. The B-tree may need to be readjusted so as to preserve the B-tree properties.
2. Set the MFT entry for the deleted file as deleted and make it available for others.
3. If the $DATA attribute is non-resident, it means that the file data is stored in external clusters, as cluster runs. Read data runs and obtains the addresses of these cluster allocated to the deleted file. Then, set corresponding $Bitmap bits to 0, making these clusters available for others.
4. As each step is taken, record action in $LogFile.

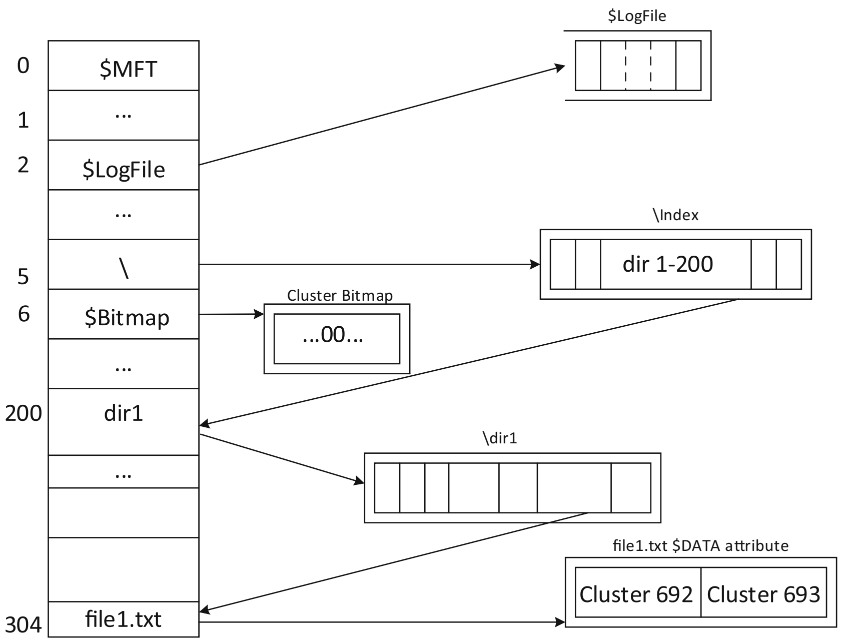
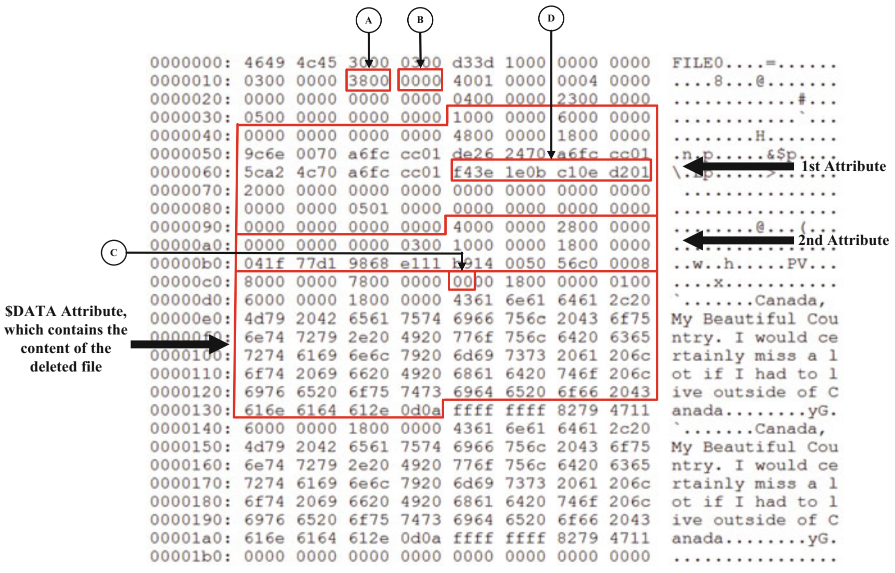


Fig. 8.2 File deletion

Note from the procedure described above that Like FAT, the data contained on disk is not erased as part of the delete process thus increasing the possibility that the deleted file can be wholly recovered. Also, we assume that we know the location of the deleted file, which is in the root directory. In reality, we may not know the exact location of deleted files we want to recover, or in some circumstances we may not even know whether there exists a deleted file. As we already know that in NTFS volumes, deleted files are indicated by a special flag in the MFT entry header. Thus, we will need to scan the MFT as thoroughly as possible to look for the MFT entries of the deleted files. Thus, their $DATA attributes can be further analyzed to obtain the addresses of the cluster allocated to a deleted file. Once these clusters can be retrieved, the only task left is to read and save the contents of each of them in order and verify their contents. Apparently, it is possible to recover the contents of these files as far as they are not overwritten. Because the cluster addressing information is stored in the MFT entry, it is worth noting that unlike FAT, a fragmented file can be recovered with the same ease as contiguous files using remaining metadata information.

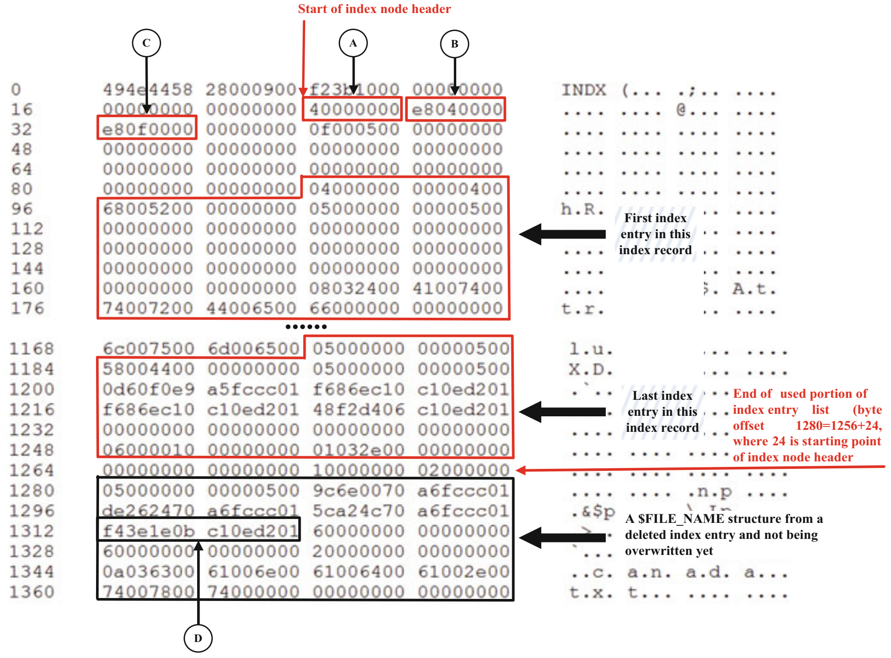
Recall from Chap. 7 the analysis of the root directory, we have a file named canada.txt in the root directory. Canada.txt is a very small file whose content is stored in its $DATA attribute. Figure 8.3 shows a hex dump of the MFT entry **35** pointing to the deleted "Canada.txt". Apparently, the content of the deleted file is still stored in its $DATA attribute.


A
B
D
C
1st Attribute
2nd Attribute
$DATA Attribute, which contains the content of the deleted file

```
0000000: 4649 4c45 3000 0300 d33d 1000 0000 0000  FILE0....=......
0000010: 0300 0000 3800 0000 4001 0000 0004 0000  ....8...@.......
0000020: 0000 0000 0000 0000 0400 0000 2300 0000  ............#...
0000030: 0500 0000 0000 0000 1000 0000 6000 0000  ............`...
0000040: 0000 0000 0000 0000 4800 0000 1800 0000  ........H.......
0000050: 9c6e 0070 a6fc cc01 de26 2470 a6fc cc01  .n.p.....&$p....
0000060: 5ca2 4c70 a6fc cc01 f43e 1e0b c10e d201  \.Lp.....>......
0000070: 2000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0501 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 4000 0000 2800 0000  ........@...(...
00000a0: 0000 0000 0000 0300 1000 0000 1800 0000  ................
00000b0: 041f 77d1 9868 e111 b914 0050 56c0 0008  ..w..h.....PV...
00000c0: 8000 0000 7800 0000 0000 1800 0000 0100  ....x...........
00000d0: 6000 0000 1800 0000 4361 6e61 6461 2c20  `.......Canada,
00000e0: 4d79 2042 6561 7574 6966 756c 2043 6f75  My Beautiful Cou
00000f0: 6e74 7279 2e20 4920 776f 756c 6420 6365  ntry. I would ce
0000100: 7274 6169 6e6c 7920 6d69 7373 2061 206c  rtainly miss a l
0000110: 6f74 2069 6620 4920 6861 6420 746f 206c  ot if I had to l
0000120: 6976 6520 6f75 7473 6964 6520 6f66 2043  ive outside of C
0000130: 616e 6164 612e 0d0a ffff ffff 8279 4711  anada........yG.
0000140: 6000 0000 1800 0000 4361 6e61 6461 2c20  `.......Canada,
0000150: 4d79 2042 6561 7574 6966 756c 2043 6f75  My Beautiful Cou
0000160: 6e74 7279 2e20 4920 776f 756c 6420 6365  ntry. I would ce
0000170: 7274 6169 6e6c 7920 6d69 7373 2061 206c  rtainly miss a l
0000180: 6f74 2069 6620 4920 6861 6420 746f 206c  ot if I had to l
0000190: 6976 6520 6f75 7473 6964 6520 6f66 2043  ive outside of C
00001a0: 616e 6164 612e 0d0a ffff ffff 8279 4711  anada........yG.
00001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

A: Offset to the first attribute 0x38
B: Flags 0x0000, which means deleted file
C: Non-resident flag 0x00, which means resident
D: File access time 0x01D20EC10B1E3EF4=131183561425174260. It means Wednesday, September 14, 2016 7:49:03pm

Fig. 8.3 Example of an MFT entry marked with deleted


Start of index node header
C
A
B
First index entry in this index record
Last index entry in this index record
End of used portion of index entry list (byte offset 1280=1256+24, where 24 is starting point of index node header
A $FILE_NAME structure from a deleted index entry and not being overwritten yet
D

```
0       494e4458 28000900 f23b1000 00000000     INDX (... .;.. ....
16      00000000 00000000 40000000 e8040000     .... .... @... ....
32      e80f0000 00000000 0f000500 00000000     .... .... .... ....
48      00000000 00000000 00000000 00000000     .... .... .... ....
64      00000000 00000000 00000000 00000000     .... .... .... ....
80      00000000 00000000 04000000 00000400     .... .... .... ....
96      68005200 00000000 05000000 00000500     h.R.           .. ....
112     00000000 00000000 00000000 00000000                    .. ....
128     00000000 00000000 00000000 00000000     ....           .. ....
144     00000000 00000000 00000000 00000000     ....           .. ....
160     00000000 00000000 08032400 41007400     ....           $. A.t.
176     74007200 44006500 66000000 00000000     t.r.           .. ....
                         ••••••
1168    6c007500 6d006500 05000000 00000500     l.u.           ... ....
1184    58004400 00000000 05000000 00000500     X.D.           ... ....
1200    0d60f0e9 a5fccc01 f686ec10 c10ed201     .`..           ... 
1216    f686ec10 c10ed201 48f2d406 c10ed201     .              ... 
1232    00000000 00000000 00000000 00000000     ....           ... 
1248    06000010 00000000 01032e00 00000000     .... .... .... 
1264    00000000 00000000 10000000 02000000     .... .... .... ....
1280    05000000 00000500 9c6e0070 a6fccc01     .... .... .n.p ....
1296    de262470 a6fccc01 5ca24c70 a6fccc01     .&$p 
1312    f43e1e0b c10ed201 60000000 00000000     .
1328    60000000 00000000 20000000 00000000     `...
1344    0a036300 61006e00 61006400 61002e00     ..c. a.n. a.d. a...
1360    74007800 74000000 00000000 00000000     t.x. t... .... ....
```

A: Offset to start of index entry list, which is relative to start of node header 0x40
B: Offset to end of used portion of index entry list, which is relative to start of node header 0x04E8=1256
C: Offset to end of allocated index entry list buffer, which is relative to start of node header 0x0FE8=4072
D: File access time 0x01D20EC10B1E3EF4=131183561425174260. It means Wednesday, September 14, 2016 7:49:03pm

Fig. 8.4 Hex dump of **Cluster 44** in the example NTFS volume

Also, the hex dump of **Cluster 44** is shown in Fig. 8.4. It contains the only index record or index node in the $INDEX_ALLOCATION attribute of the root directory after the file “Canada.txt” is deleted.

Bytes 4–7 (E8 04 00 00) of the index node header, labeled as **B** in Fig. 8.4, show offset to end of used portion of index entry list. Here, 0x04E8 = 1256, which is relative to start of node header. It indicates that the used portion of index entries ends at byte offset 1280 = 1256 + 24, where 24 is the size of INDX header coming before the index node header in an INDX record. Apparently, the last index entry is now the one containing the “.” File, which is the second last one from Fig. 7.21 in Chap. 7. In other words, the last index entry has been deleted after the file it points to is deleted. In our example, the deleted last index entry is wiped out, and the $FILE_NAME structure is kept as it was.

For every file on an NTFS volume, there are the following four timestamps included in the $FILE_NAME attribute, where their byte offset locations in the $FILE_NAME structure are included in brackets.

1. File creation time (Bytes 8–15)
2. File modification time (Bytes 16–23)

3. MFT modification time (Bytes 24–31)
4. File access time (Bytes 32–39)

By comparing with Fig. 7.21 in Chap. 7, it can be observed from the Figure above that file access time in the $FILE_NAME structure of the deleted file has been changed to 0x01D20EC10B1E3EF4 = 131183561425174260. In an NTFS file system, timestamps are stored as 8-byte file time values, which represents the number of 100-nanoseconds since 12:00 A.M. January 1, 1601, also known as File Time. It means the file is last accessed on Wednesday, September 14, 2016 7:49:03 pm. In our experiment, we deleted the file canada.txt at that time. Further, note from Fig. 8.3 that another copy of file access time can be found in $STANDARD_INFORMATION attribute, whose data structure is shown in Table 8.1. The first 4 bytes of the first attribute (10 00 00 00) shows that it is an $STANDARD_INFORMATION attribute, here 0x10 = 16. Bytes 0x28-0x2F in $STANDARD_INFORMATION attribute is the file access time. Apparently, it is also updated with the same value.

It is worth noting that there are many types of timestamps. In addition to File Time, another popular one is Unix Timestamp or Epoch, which is the number of seconds since 12:00 A.M. January 1, 1970. It is a 32-bit number.

From the example above, we can see that after a file is deleted in NTFS, the contents of the file are actually not erased. Also, the index entry, which points to the deleted file, was deleted, but the $FILE_NAME structure could not be overwritten yet. Unfortunately, there is no way for us to link them together. In other words, we can only conclude that a file named “Canada.txt” has been removed from the root directory and there is also a deleted file in the NTFS volume. The deleted file was using MFT entry **35**. Nonetheless, recall the above discussion, we have about the timestamps in both $STANDARD_INFORMATION attribute from the MFT entry that points to the deleted file and the $FILE_NAME structure from the deleted index

Table 8.1 Data structure for $STANDARD_INFORMATION attribute (The time values are given in 100 nanoseconds since January 1, 1601, UTC)

Byte range	Bytes	Description
0–15	16	Generic attribute header
16–23	8	File creation time
24–31	8	File modification time
32–39	8	MFT modification time
40–47	8	File access time
48–51	4	Flags
52–55	4	Max versions
56–59	4	Version number
60–63	4	Class ID
64–67	4	Owner ID
68–71	4	Security ID
72–79	8	Quota charged
80–87	8	Updated sequence number

entry which points to the deleted file. The changes to file access time could allow us to link the deleted file's contents to a file name discovered in index attributes, by analyzing and correlating other information,, including other timestamps, file types (or filename extension).

It is worth noting that NTFS uses B-tree for indexing. When a file is deleted from the directory, the index entry representing the file is removed and the B-tree has to be adjusted to preserve the B-tree properties if necessary. Therefore, the "Index Entry" for the deleted file may be overwritten when the tree is resorted. The naming information of the deleted file could be lost.

In next section, we will discuss data recovery in NTFS file system using remaining file metadata information. Specifically, we look for these MFT entries marked as deleted and analyze their $DATA attributes, which contain file data itself if it is a resident attribute or where that data is located if it is a non-resident attribute.

8.1.2 Deleted File Recovery in NTFS File System

As discussed above, NTFS file system handled file deletions by setting a special flag in the MFT entry without actually clearing the data contained in the clusters assigned to the file. Often software products using these techniques caution users to stop using the device and to refrain from creating or modifying any more files on it [2, 3]. This is essential advice because when new storage space is needed to store a new file or to store additional data for a modified file, the device may utilize some of these clusters belonging to deleted files. This will result in the old data being overwritten, making it impossible to recover. However, until this overwriting process occurs, the data remains largely intact and the problem of recovery is simplified.

For simplicity, we assume that the data contents of the deleted file have not been overwritten, and are left completely intact. Then further assume that the MFT entry that points to the deleted file is not overwritten yet and the deleted index entry representing the deleted file hasn't been overwritten.

Suppose that we know the cluster size. Deleted file recovery in NTFS can actually be broken down into two stages. In stage 1, we need to recover the data contents of a deleted file in NTFS file system by performing the following operations.

Firstly, scan the entire MFT one entry at a time and compile a list of entries with a deletion marker (Bytes 22-23 of the MFT entry is 0x0000).

Secondly, extract and analyze these MFT entries' $DATA attributes, which contain the data contents of the deleted files. If it is a resident attribute, it contains the file contents. Then, read and save the contents of the $DATA attribute as a recovered file. If it's a non-resident attribute, its contents are data runs, which specify the addresses of the clusters where the file contents live. Once these clusters can be re-identified, we first need to validate their allocation status. In doing so, we can check cluster allocation file ($Bitmap). The $BitMap is the seventh entry (MFT entry 6) in the MFT. If any bits in the Bitmap representing the allocation status of these clusters are set as 1, it means these data clusters have already been overwritten with new data. In other words, we are unable to successfully recover the deleted file. This is based on the principle that as only one file can occupy any one cluster on a hard drive. If other files are using your deleted files storage space, then it is likely that the original data has been overwritten and permanently destroyed. Otherwise, the only task left is to read and save the contents of each cluster and verify their contents. It is worth noting that both contiguous and fragmented files can be recovered with almost the same amount of effort in NTFS systems.

In stage 2, we search the naming information of the deleted files by performing the following operations:

- Firstly, scan the MFT to look for the MFT entries which represent the directories (or folders), including both deleted and existing ones.
- Secondly, extract and analyze these MFT entries' index attributes, which contain index entries that include the $FILE_NAME structure. Scan these index attributes as thoroughly as possible for the deleted index entries and parse out the naming information of the deleted files.
- Finally, match recovered file contents from the $DATA attributes in MFT entries to their names found in deleted index entries by correlating information, including timestamps, file types (or filename extension).

Review Questions

1. What is the MFT entry number for the root directory of an NTFS? ________________
2. Which of the following types of timestamps is in 100-nanoseconds since 12:00 A.M. January 1, 1601?
   (a) File time
   (b) Unix Timestamp
   (c) Epoch
   (d) None of the above
3. Which of the following attributes in the MFT entry store file contents?
   (a) $DATA
   (b) $FILE
   (c) $INDEX_ROOT
   (d) $INDEX_ALLOCATION
4. Which of the following statements is true?
   (a) If a $DATA attribute is resident, the size of its attribute content is the file size.
   (b) If a $DATA attribute is non-resident, the size of its attribute content is the file size.
   (c) There doesn't exist a $DATA attribute in a MFT entry which points to a directory
   (d) None of the above

5. Suppose the following is a hex dump of a $DATA attribute found in a MFT entry marked with a deleted file

```
0000000: 8000 0000 4800 0000 0100 4000 0000 0200  ....H.....@.....
0000010: 0000 0000 0000 0000 0300 0000 0000 0000  ................
0000020: 4000 0000 0000 0000 0040 0000 0000 0000  @........@......
0000030: 133b 0000 0000 0000 133b 0000 0000 0000  .;.......;......
0000040: 2104 8837 0001 0000                      !..7....
```

What is the size of the file in bytes? ____________________________

What is the size of disk space allocated to the file in bytes? ____________________________

What is the size of slack space for the file in bytes? ____________________________

What are addresses of the clusters allocated to the deleted file? (Listed in the order in which they are allocated) ____________________________________

Is this file fragmented? (Yes/No) ____________________

8.2 Practical Exercise

The objective of this exercise is to manually recover the deleted files on an NTFS file system based on remaining metadata information.

8.2.1 Setting Up the Exercise Environment

We'll continue to use the disk image "thumbimage_ntfs.dd" provided in the book, particularly extracting the first partition image (or a NTFS file system image) within it.

8.2.2 Exercises

Part A: File System Layer Analysis

Now analyze the file system image by answering the following questions using the *fsstat* command in TSK:

Q1. What is the cluster size (in bytes)? ____________________

Q2. What is the MFT entry size (in bytes)? ____________________

Part B: Mounting and Unmounting File Systems

1. Mount the extracted partition into /mnt/forensics with read write access.
2. Change into the "/mnt/forensics" directory and then delete a file named "canada.txt" by using the rm command.
3. Unmounting the extracted partition.

Part C: Recovering the Deleted File "canada.txt"

Q3. After the file canada.txt has been deleted, you can discover that its MFT entry has been marked as deleted, particularly with its byte offset 22-23 replaced with 0x0000.

- Scan the MFT one entry each time and locate the entry pointing to the deleted "canada.txt" file. Particularly, we look for the one with the value of 0x0000 from the byte offset 22–23 of each MFT entry. For simplify, you should be able to locate it by checking the first 50 entries in the NTFS image provided here.
- Analyze the identified MFT entry and extract its $DATA attribute. Parse the $DATA attribute and obtain the following information, including whether resident or non-resident attribute: ________________, file size: ________________
- If it is a resident attribute, it contains the file contents. Then, read and save the contents of the $DATA attribute as recovered file.

You will need to find out the starting point the $DATA attribute content and the content length. Then, you can extract the attribute content by using the dcfldd utility by specifying the block size (*bs*) value as 1 byte, the *skip* value as the starting point of the $DATA attribute content and the *count* value as the content length.

If it's a non-resident attribute, its contents are data runs, which specify the addresses of the clusters where the file contents live. Once these clusters can be re-identified, we first need to validate their allocation status. In doing so, we can check cluster allocation file ($Bitmap). The $BitMap is the seventh entry (MFT entry 6) in the MFT. If any bits in the Bitmap representing the allocation status of these clusters are set as 1, it means these data clusters have already been overwritten with new data. In other words, we are unable to successfully recover the deleted file. Otherwise, read and save the contents of each cluster. It is worth noting that the last cluster may not be fully utilized by the file. In other words, the last cluster is left with unused extra space (or slack space) and should not be included into the recovered file. The size of slack space can be determined by the file size and the size of disk space allocated to the file.

Note that once you have successfully completed this exercise, you can verify the successful recovery of the deleted file by mounting the modified partition image and seeing whether or not you can view the canada.txt properly. Or, you can calculate the MD5 hash value of your recovered file and see whether the resulting hash value is equal to “2af85496e256e2b917e9af38f9c865d7” in hex. Being equal means that you have successfully recovered the deleted file “Canada.txt”.

References

1. New Technology File System (NTFS). http://www.pcguide.com/ref/hdd/file/ntfs/index.htm
2. Brian Carrier. File System Forensic Analysis. Addison-Wesley Professional; 1 edition (Mar 27 2005) ISBN-10: 0321268172
3. M. Alazab, S. Venkatraman, P. Watters. Effective Digital Forensic Analysis of the NTFS Disk Image. Special Issue on ICIT 2009 conference - Applied Computing, 2009