IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555
Vol. 2, No.3, June 2012

NTFS Deleted Files Recovery: Forensics View

Sameer H. Mahant
B.B.Meshram
Department of Computer Engineering Veermata Jijabai Technological Institute,
Mumbai, India

Abstract—Data stored in the files is the main source of evidence in computer forensics. The file system is used to manage these files present on disk. A suspect can remove evidence present on disk by deleting files containing evidences. It is important for forensic investigator to get back the evidences deleted by suspect. Though there are many tools available in market for recovering deleted files, no published documentation is available for their internal working for recovery procedure. It is important for examiner to know details of file system before using available tools, so the results can be verified. In this paper we have described the structure and operation of NTFS file system related to deleted file recovery and proposed a method to recover deleted files on disk formatted using NTFS file system.

Keywords- NTFS, Forensics, Deleted file Recovery, MFT Entry

I. INTRODUCTION

With the advancement of computer technology and internet, use of computers is increased at personal as well as organizational level. There uses for performing cybercrime or illegal activities are also increased. The computers store the data on hard disk using the appropriate file system supported by operating system installed on that computer system. Microsoft's Windows operating systems are the most common amongst other operating systems like Ubuntu, Red Hat Linux, Apple's OSX, etc.

Windows operating systems use NTFS as their default operating system, so it is important to know its internal details and working. Tapping and analyzing the useful data of NTFS file system is of great importance in computer forensics. If the data is deleted, we couldn't directly observe it under Windows Operating System, but it can include some important evidence of crime [1]. Therefore it is necessary to understand how file system handles deletion of file.

This paper describes the NTFS file system's structure and how it handles file deletion which will help in recovering deleted files present on disk. The paper also proposed a method to check validity of recovered data belonging to deleted file.

II. RELATED WORK

The internal details of NTFS file system is described by Brian Carrier in its book "File System Forensic Analysis" [2]. The Basics information about NTFS file system's Boot Sector, Master File Table (MFT), different file types and their attributes is documented by www.NTFS.com [3]. Authors Byeongyeong Yoo, Jungheum Park, Jewan Bang, Sangjin Lee described the problem of the conventional tools for deleted NTFS compressed files and proposes a recovering method for the deleted NTFS compressed files [4].

Through detailed analysis and research on the storage principles of the NTFS file system, the object-oriented method is put forward to design NTFS file parsing system by Zhang Kai, Cheng En, Gao Qinquan [1]. Based on NTFS file system, Liu Naiqi, Wang Zhongshan, Hao Yujie, QinKe proposed an algorithm of reconstructing directory tree for deleted files [5].

Most of the documents have presented a simple recovery procedure for deleted files recovery on NTFS file system, but the recovered files are not verified for correct data. Also the recovery procedure for fragmented Master File Table is not mentioned. This motivated us to create more advanced recovery method for recovering deleted files form NTFS file system and verify the results so they can be used in forensic examinations.

III. NTFS BASIC CONCEPTS

This section describes the basics of NTFS file system, Master File Table attributes and their headers, metadata files present in NTFS file system.

A. Overview of NTFS

NTFS file system is preferred/default file system for Microsoft's various desktops and server operating system. Its new features which are not present in earlier FAT file system make it a very complex file system. Let's take look at NTFS file system structure.

For each user data file, NTFS file system creates a File Record. All the File Records are stored in a special table called as Master File Table (MFT). MFT entry has size of 1024 bytes. Then how can a file having data greater than 1024 bytes fit in the MFT entry? This is where most people may go wrong. Actually MFT does not store the data of file (unless the data is small to be able to fit in MFT Entry), but it contains the information about file which is required for retrieving data and other file operations.

The location of the Master File Table is available in partition boot sector (512 bytes) which is the first sector of the NTFS formatted partition. It is different than disk boot sector which is first sector of entire disk. Disk boot sector contains

information to find start address of partition, where we can find the partition boot sector. The starting cluster address of Master File Table is located at offset 48-55 (8 bytes) in the partition boot sector of NTFS partition. To get the physical address of NTFS partition relative to the start Partition we need to multiply it by 'bytes per sector' (located at offset 11-12) and 'sectors per cluster' (located at offset 13).

MFT Start Address (Physical) = Start Cluster of MFT * Bytes per Sector * Sectors per Cluster

The information about file is stored in MFT Entry as series of attributes. Each attribute has a specific purpose and its own internal structure. Each attribute has an identifier which identifies type of attribute. The list of attributes is given in Table I:

TABLE I. MFT ENTRY ATTRIBUTES WITH THEIR TYPE IDENTIFIER

Type Identifier (Decimal)	Type Identifier (Hexadecimal)	Attribute Name
16	0x10	$STANDARD_INFORMATION
32	0x20	$ATTRIBUTE_LIST
48	0x30	$FILE_NAME
64	0x40	$VOLUME_VERSION
64	0x40	$OBJECT_ID
80	0x50	$SECURITY_DESCRIPTOR
96	0x60	$VOLUME_NAME
112	0x70	$VOLUME_INFORMATION
128	0x80	$DATA
144	0x90	$INDEX_ROOT
160	0xA0	$INDEX_ALLOCATION
176	0xB0	$BITMAP
192	0xC0	$SYMBOLIC_LINK
192	0xD0	$REPARSE_POINT
208	0xE0	$EA_INFORMATION
224	0xF0	$EA
256	0x100	$LOGGED_UTILITY_STREAM
---	0xFFFFFFFF	End of Attributes

Every file does not contain each of the attribute listed in above table. Depending on the file different attributes are added in File Record.

To manage the entire file system, NTFS creates metadata files which contain administrative data. To differentiate NTFS metadata files from user files, each NTFS metadata file's name begin with "$" sign [except '.' (Dot) – root directory for files system]. All these files are kept hidden and not shown to the end user.

To keep file system more manageable NTFS reserves first sixteen entries in MFT only for NFTS metadata files, and their location in the MFT is also fixed. File Records for user created files are added after that reserved entries. Table II. shows the NTFS metadata files with their location number in MFT:

TABLE II. NTFS FILE SYSTEM METADATA FILES

Entry Number	NFTS Metadata File Name
0	$MFT
1	$MFTMirr
2	$LogFile
3	$Volume
4	$AttrDef
5	. (Dot)
6	$Bitmap
7	$Boot
8	$BadClus
9	$Secure
10	$Upcase
11	$Extend

Since each File Record is stored in MFT, we will refer it to as MFT Entry. Each and every file and folder on disk has an entry in MFT [6]. Files and folders are differentiated using simple flag values present in MFT Entry.

Following sections describes the internal structure of attributes and NTFS metadata files related to this paper.

B. *MFT Entry Attributes*

This section describes information of MFT Entry's $STANDARD_INFORMATIN, $FILE_NAME and $DATA attributes.

1) $STANDARD_INFORMATIN Attribute : has the type identifier of 16 (0x10H). It has following structure as shown in Fig. 1.[7].

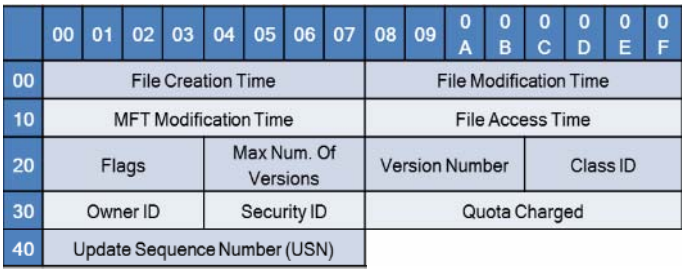

Figure 1. $Standard_Information attribute structure

In the above structure the most important are the four timestamps i.e. File Creation Time, File Modification Time, MFT Modification Time and File Access Time. As we will later see that $FILE_NAME attribute also contains the same four timestamp values which are shown to user. File times of $STANDARD_INFORMATION are not show to user by operating system. All the values are stored as the number of nanoseconds since 1st January 1601 UTC [8].

2) $FILE_NAME Attribute : attribute has type identifier of 48 (0x30H). It has following structure as shown in Fig. 2.

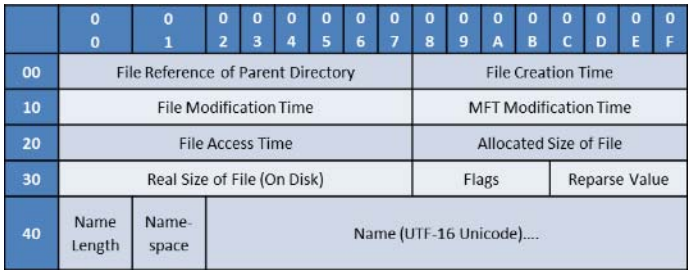

Figure 2. FileName attribute structure

File reference of parent directory contains MFT Entry address of parent folder. The timestamps exists in the $FILE_NAME attribute are the ones that are shown to user when the file properties are viewed. The UTF-16 Unicode encoded name of file is stored in this attribute. A file or folder can have multiple $FILE_NAME attribute to store file name in different formats like POSIX, 8.3 DOS, Win32 format. The value in Namespace field identifies which format is used to store the file name.

- 0 → POSIX format
- 1 → Win32 format
- 2 → DOS format
- 3 → Win32 & DOS format (used when original name already fits in the DOS namespace and two names are not needed)

3) $DATA Attribute: has type identifier of 128 (0x80H). Each file has a default $DATA attribute which does not have any name, additional $DATA attributes must have a name. The folders by default does not have any $DATA in MFT Entry. $DATA attribute holds files data in raw format.

C. MFT Entry Header and Attribute Headers

In this section we will discuss about the header structure of MFT Entry and Attribute, which helps in analyzing the MFT Entry and its attributes. Each MFT Entry's Header contains following information shown in Table III [2].

TABLE III. MFT ENTRY HEADER DETAILS

Byte Range	Information
0-3	Signature ("FILE" or "BAAD")
4-5	Offset To Fixup Array
6-7	Number of entries in Fixup Array
8-15	$Logfile Sequence Number
18-19	Link Count
20-21	Offset to First Attribute
22-23	Flags (about Allocation status of entry)
24-27	Used Size of MFT
28-31	Allocated Size of MFT
32-39	File Reference to base record
40-41	Next Attribute Id

In the MFT Header, Offset to first attribute is given which is relative to start of MFT Entry. This is used to start parsing attributes which are present in MFT Entry. Flags are used to identify whether the entry is for file or folder and also identifies whether the file/folder is deleted or not. The different flag values and there interpretation is given in Table IV.

TABLE IV. MFT HEADER FALG VALUE DETAILS

Value	Description
0x00	Deleted File Entry
0x01	File Entry
0x02	Deleted Folder Entry
0x03	Folder Entry

Each Attribute present in MFT Entry also has header associated with it, which describes its type, name, length, Id and some other information. Because size of MFT Entry is only 1024 bytes, it holds the location information of large file's data. But if the data size is small enough to fit in MFT Entry (normally < 700 bytes) along with other management information then the data is stored in MFT Entry itself. This same is also true for the attributes present in MFT Entry.

Due to this dual behavior the attribute header also has two types: Resident Attribute Header and Non-resident Attribute Header. One of the attribute headers is present at the beginning of attribute before the actual content of attribute starts depending on its content length.

The Resident Attribute Header is used for the attributes whose data size is small and can be stored in MFT Entry itself. The Non-resident Attribute Header is used for the attributes which stores its content into external clusters of file system. Both the headers have a common 16-bytes header which has a flag to identify the type of header. The structure of common attribute header is shown in Fig. 3.

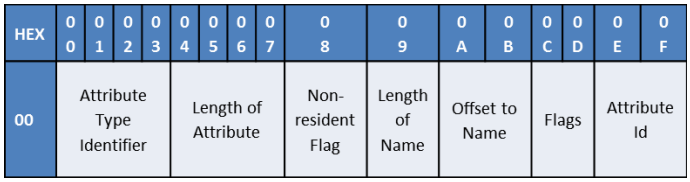

Figure 3. Comman attribute header structure

Attribute Type Identifier field is used to identify the attribute i.e. $STANDARD_INFORMATION, $FILENAME,

$DATA, etc. Non-resident flag field is used to identify the type of header i.e. Resident (Flag = 0) or Non-resident attribute header (Flag = 1).

The structure of Resident attribute header is shown in Fig. 4.

HEX	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	Common Attribute Header															
10	Size of Content								Offset to Content							

Figure 4. Resident attribute header structure

The structure of Nonresident attribute header is shown in Fig 5.

HEX	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	Common Attribute Header															
10	Starting Virtual Cluster Number of RunList											Ending Virtual Cluster Number of RunList				
20	Offset to RunList		Compression Unit Size		Unused							Allocated Size of Attribute Content				
30	Actual Size of Attribute Content											Initialized Size of Attribute Content				

Figure 5. Nonresident attribute header structure

The Non-resident attributes stores the information about data stored in external clusters in special format called as RunList. Offset to the RunList is given in attribute header, but this offset is relative to start of attribute contents excluding header. Fig. 6. shows sample of single cluster Run followed by its interpretation:

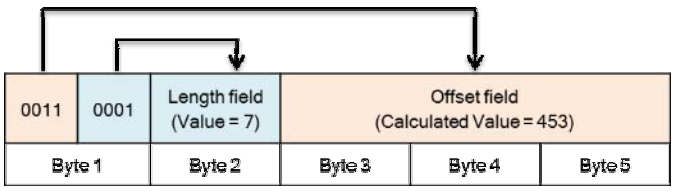

Figure 6. Cluster Run format

RunList has variable length, but must be at list one byte. The first byte of runlist is organized into upper 4-bits and lower 4-bits. The value of lower 4 bits specifies the number of bytes to consider in the calculation of length of a single Run. Run is continuous collection of Cluster Addresses. This field follows the header byte. The value of upper 4 bits specifies the number of bytes to consider in the calculation of start Offset address of cluster. This field follows the Length field. To convert the RunList to actual cluster address list simply consider the Offset field value as first cluster number and continue the numbers till the value specified in Length field.

In above Example the lower 4 bits have value of 1, which specifies that the length field is 1 byte long. The upper 4 bits having value of 3 specify that the Offset field is 3 bytes long. The cluster number list is: 453, 454, 455, 456, 457, 458, 459.

The sample MFT entry showing different attributes and headers is given in Fig. 7.

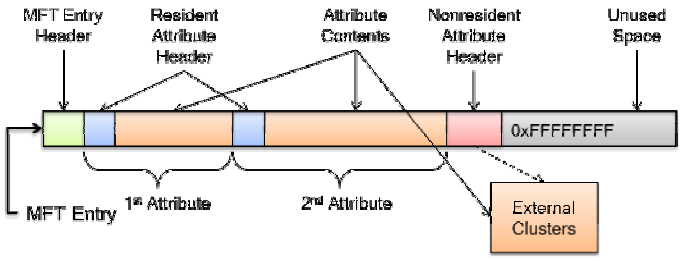

Figure 7. MFT Entry with sample attributes and headers

D. NTFS Metadata Files

In this section we will discuss about $MFT and $BITMAP metadata files. Understanding of these files is required to analyze the structure of MFT on disk and to locate used and unused clusters.

1) $MFT File: Entry 0 in MFT is reserved for the $MFT metadata file. Like any other user file it also has different attributes. The $DATA attribute of this file contains the clusters (group of sectors) used by MFT. These are the locations where we have to search for file records because each and every file has a record in MFT. So in order to get information about every file on disk we have to parse $DATA attribute of $MFT metadata file then for each cluster listed in the $DATA attribute, convert to corresponding sectors and the parse File Records present at that locations.

2) $BITMAP File: The $BITMAP metadata file is located at Entry 6 in MFT and its $DATA attribute has the information about allocation status of disk clusters. The data is organized into 1-byte values. Each Bit of a data byte represents one cluster. We can get allocation status of 8 clusters using single byte. But in each byte the cluster numbers are interpreted in reverse order. Consider byte with binary values 00000101. Here the LSB value '1' denotes that 'cluster 0' is allocated. The MSB is used to denote allocated status of 'cluster 8' which is unallocated as it has value '0'.

IV. NTFS DELETED FILE RECOVERY

According to Davis et. al., "one of the most common tasks requested in any investigation is to find and recover the files that have been deleted from the system. This will often be a prime indicator of what the suspect is trying to hide if you find mass deletions before your imaging occurred" [9].

In this section we will discuss the file system level changes that happen when a file is deleted. Then we will discuss about how we can recover deleted file using MFT Entry analysis. Finally we will take a look at the tool named "NTFS True Recovery", that we have developed for forensic examiner to

automate recovery of deleted files on NTFS file system. The test results for tool verification are also given.

A. *File Deletion on NTFS*

When we delete a file on NTFS file system:

Step 1: File's MFT Entry is made unallocated by changing the flag values in MFT Entry Header. For files it is changed from 0x01 to 0x00, and for folder it is changed from 0x03 to 0x02.

Step 2: $Bitmap attribute of $MFT metadata file is processed and value 0 is set for the file's MFT Entry.

Step 3: The non resident attributes of file's MFT Entry are processed and their clusters are set to unallocated in $BITMAP metadata file.

From above steps it is clear that when file is deleted on NTFS files system, actual data content of the file is not deleted. Only the changes to the MFT Entry Header and some metadata files are made.

B. *Recovering Deleted Files on NTFS*

To process deleted files authors Zhang Ka, Cheng En, Gao Qinquan used following technique [1]:

1. According to the normal file analysis, parse all the files, and store the flag bit information of the deleted file.
2. Sort order of all file by address space.
3. When analysis is over, find the deleted file and folder through the flag byte.
4. When the data area of deleted file is not covered by the adjacent two files, the file date is fine, so do not clip.
5. When the area is partly covered by the adjacent two files, we should cut out the covered space, and then the remainder is right content after analysis.
6. When the area is totally covered by the adjacent files, then the data is lost and is totally non-recoverable.

Fig. 8, Fig. 9. and Fig. 10. shows totally recoverable model, partial recoverable model and non-recoverable model respectively which are used in above technique.

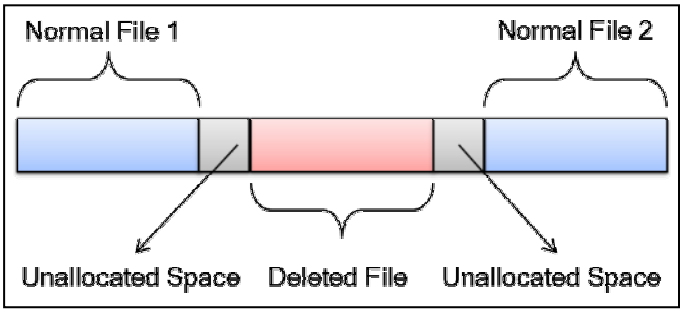

Figure 8. Totally recoverable model

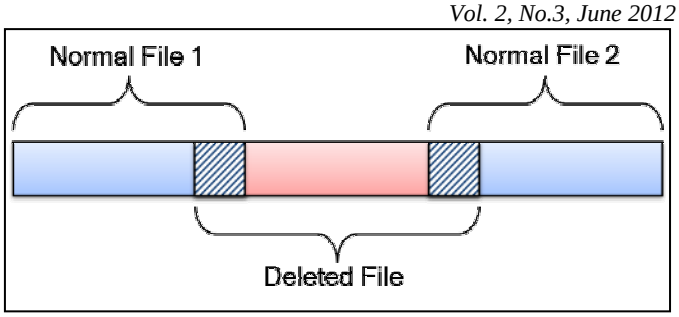

Figure 9. Partial recoverable model

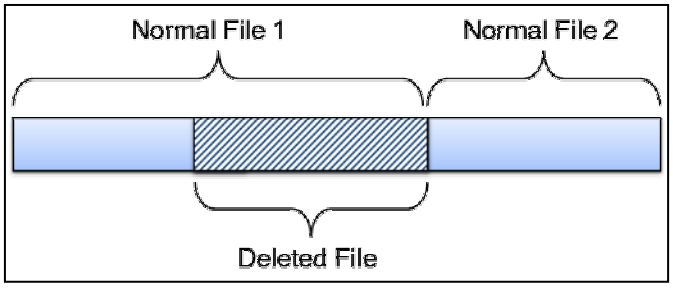

Figure 10. Nonrecoverable model

The above technique works well if the file's data is stored as series of clusters which are continuous. The above mentioned technique did not consider fragmented file's data which is stored in multiple clusters runs which is required for more detailed file recovery of deleted files. Also keeping address information of all files and sorting them is additional overhead and may affect performance of recovery tool.

Each file and folder stored in NTFS has its entry in Master File Table. Even deleted files and folders also have there entry in Master File Table until it is reallocated to other file or folder. We also know that $BITMAP metadata file has information about allocation status of disk clusters. Using this information we have proposed the technique to recover deleted files from NTFS partition which is given below:

Steps followed in deleted file recovery are:

1. Search Files MFT Entry
2. Process it's $DATA attribute
3. If $DATA attribute is resident, file content is present in MFT Entry. Just copy it to external location to complete recovery process.
4. If $DATA attribute is non-resident, file's contents are present in external cluster. Then parse the RunList present in the attribute.
5. Check if each cluster listed in Runlist is allocated or not. If all clusters have allocated status as 0, means that the files data is still present on disk. Therefore complete recovery is possible. Copy file's contents to external location to complete recovery process.
6. If some clusters have allocated status as 1, then the partial recovery is possible. Copy file's contents to external location with reallocated cluster's values as zeros to complete partial recovery process.

7. If all clusters have allocated status as 1, then the file's contents are lost and recovery is impossible.

The above technique can handle the file's data which is fragmented across number of cluster runs so gives more granular results in recovery of deleted files. The technique does not require any external storage or use of sorting therefore can increase the performance in recovery process of deleted files.

The complete flowchart for deleted file recovery process is shown in Fig.11.

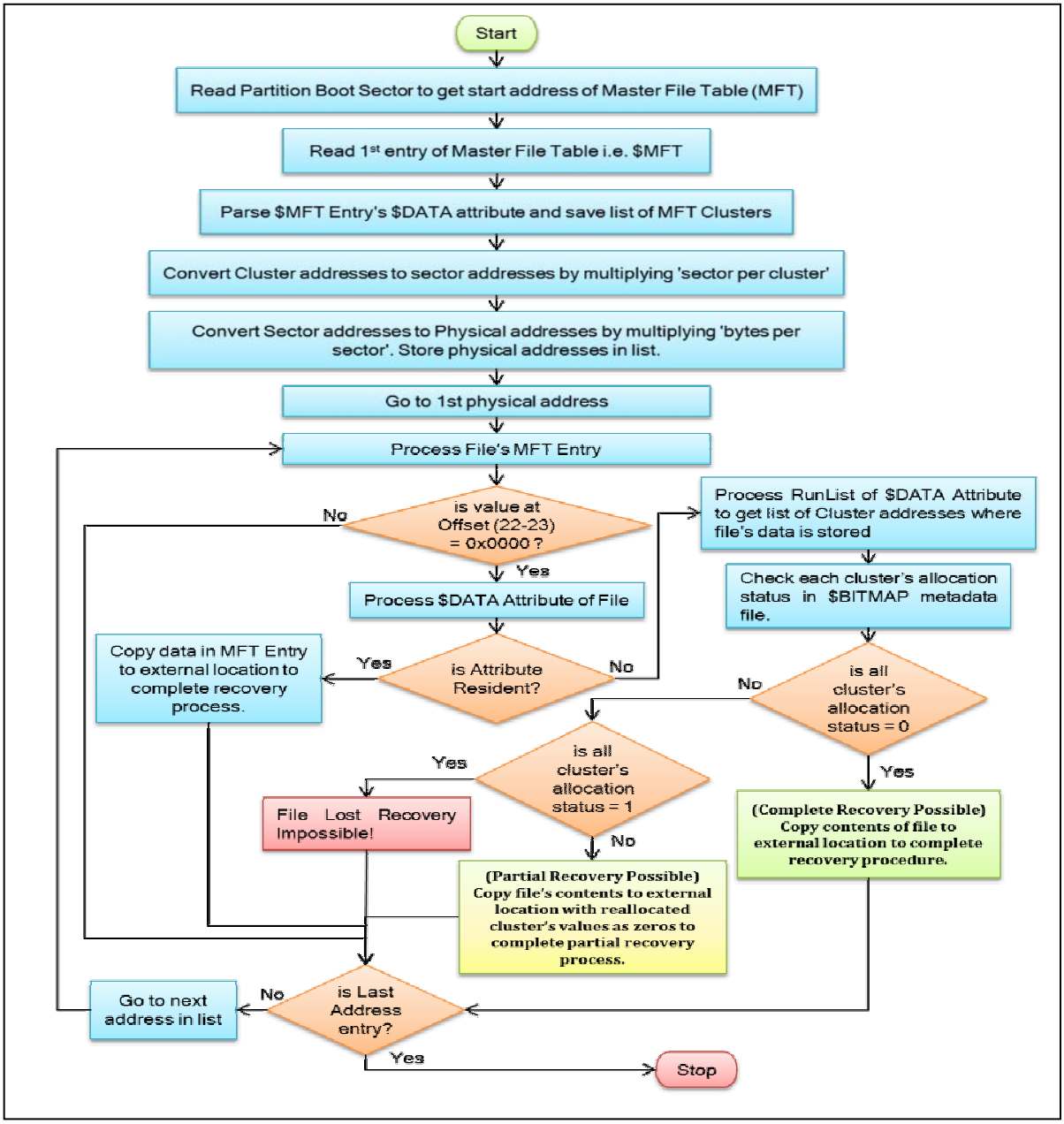

Figure 11. Deleted file recovery process

V. CONCLUSION

Data stored in the files is the main source of evidence in computer forensics. The files are stored on hard disks or on some removable devices. As the size of storage devices is increasing, time take in analyzing them is increasing. Finding evidences in deleted files and recovering them is important for investigation. This paper described the internal details on NTFS file system and how it handles the delete files, which can be used to verify the results of recovery tools. In this paper we have also proposed a technique which will help in faster recovery of deleted files.

REFERENCES

[1] Z. Kai, C. En and G. Qinquan, "Analysis and Implementation of NTFS File System Based on Computer Forensics," in The Second International Workshop on Education Technology and Computer Science, Wuhan, Hubei, China, 2010.

[2] B. Carrier, File System Forensic Analysis, Addison Wesley Professional, 2005.

[3] www.NTFS.com. [Online]. [Accessed 22 3 2012].

[4] B. Yoo, J. Park, J. Bang and S. Lee, "A Study on a Carving Method for Deleted NTFS Compressed Files," in Human-Centric Computing (HumanCom), 2010 3rd International Conference, Cebu, Philippines, 2010.

[5] L. Naiqi, W. Zhongshan, H. Yujie and Q. Ke, "Computer Forensics Research and Implementation Based on NTFS File System," in ISECS International Colloquium on Computing, Communication, Control, and Management, Guangzhou, 2008.

[6] E. Huebner, D. Bem and C. K. Wee, "Data hiding in the NTFS filesystem," Digital Investigation The International Journal of Digital Forensics & Incident Response, vol. 3, no. 4, pp. 211-226, December 2006.

[7] J. Bang, B. Yoo, J. Kim and S. Lee, "Analysis of Time Information for Digital Investigation," in Fifth International Joint Conference on INC, IMS, and IDC, Seoul, Korea, 2009.

[8] E. Casey, "Chapter 5: Windows Forensic Analysis," in Handbook of Digital Forensics and Investigation, United States of America, Elsevier Inc., 2010, pp. 209-300.

[9] A. Philipp, D. Cowen and C. Davis, Hacking Exposed Computer Forensics, 2nd ed., McGraw-Hill, 2005.

AUTHORS PROFIL

Sameer Mahant is graduated in Computer Engineering from Mumbai University, India. Currently pursuing Master's Degree in Computer Technology From, Veermata Jijabai Technological Institute, Matunga, Mumbai. His research interests are computer security, web security and digital forensics.

Dr. B. B. Meshram is working as Professor in Computer Technology Dept., VJTI, Matunga, Mumbai. He is Ph.D. in Computer Engineering and has publication of 25 international journals, 70 international conference and 39 national conference papers to his credit. He has taught various subjects such as Object Oriented Software Engg., Network Security, Advanced Databases, Advanced Computer Network (TCP/IP), Data warehouse and Data mining, etc. at Post Graduate Level. He has guided several projects at graduate and post graduate level. He is the life member of CSI and Institute of Engineers.