A Strategy for Testing Metadata Based Deleted File Recovery Tools

James R. Lyle

National Institute of Standards and Technology
jlyle@nist.gov

Abstract. Deleted file recovery tools use residual metadata left behind after files are deleted to reconstruct deleted files. File systems use metadata to keep track of the location of user files, time stamps of file activity, file ownership and file access permissions. When a file is deleted, many file systems do not actually remove the file content, but mark the file blocks as available for reuse by future file allocations. This paper describes a strategy for testing forensic tools that recover deleted files from the residual metadata that can be found after a file has been deleted.[1]

Keywords: Digital forensics, tool testing, deleted file recovery.

1 Background

A file system is used to store data for access by a computer. The data is normally stored within a tree-structured hierarchy of directories and files. In addition to files and directories, *special objects* (e.g., links and shortcuts) may be defined for a file system. File system *metadata* contains information to describe and locate every file within a given file system, such as, file name, permissions and modify, access, create (MAC) times. Some *metadata* resides in directory entries, but additional *metadata* may reside in special files (e.g., NTFS $MFT) or other locations (e.g., UNIX i-nodes) for a given file system.

When a file or directory is deleted from a file system, the associated *metadata* entry and the stored data are no longer directly accessible to the user and appear to be completely removed. However, in many file systems, e.g., FAT, neither the metadata associated with the file nor the actual content is completely removed. This creates a situation where there is *residual metadata* (metadata remaining after a delete has occurred) that is still accessible and can be used to reconstruct deleted files [1]. However, depending on the original format and structure of the metadata, not all of it may be reachable.

[1] Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the author or the author's employer, nor does it imply that the products are necessarily the best available for the purpose.

P. Gladyshev and M.K. Rogers (Eds.): ICDF2C 2011, LNICST 88, pp. 104–114, 2012.

Many forensic tools exploit the behavior exhibited by file systems of leaving metadata behind after a file is deleted to attempt to recover deleted files. Metadata based deleted file recovery should not be confused with *file carving*, i.e., scanning unallocated file space for the file signatures present within a file itself to identify a deleted file. The scope of this paper is limited to metadata based deleted file recovery tools that use file system metadata from file system structures such as directories or i-nodes to identify recoverable deleted files. A different test strategy is required for testing file carving tools because the test issues that are addressed in file carving are different from the ones for metadata based deleted file recovery. *Directory carving*, scanning unallocated space for deleted directories and i-nodes to locate usable deleted file metadata, is within the scope.

2 Relevant File Systems Design Features

File systems are designed to allow an operating system to have access to secondary storage in a manner that is both efficient and timely. In the past, storage devices have been expensive and slow when compared to random access memory. Accessing the secondary storage efficiently, although implemented differently in each file system, tends to have side effects that can be exploited to recover deleted files. Two of the key relevant design features are the conservative nature of file system activity and contiguous writes [2].

File systems are conservative with storage access operations and avoid unnecessary operations. This characteristic implies that, to be fast and efficient, file systems perform many activities with minimal changes or overhead. In the case of file deletion, in most situations only a *logical deletion* is performed—meaning that the actual data is not erased, but the metadata that indexes the information is changed, flagged or removed. By using this technique, a file, no matter how large, can be deleted by simply modifying or removing entries from file system metadata. The simplest example of this is how a Windows FAT file system deletes files. It locates the directory entry of the file to be deleted, changes the first character in the file name to a '0xE5' hex value, and then zeros the file allocation table making both the file metadata entry and the file data blocks available for reuse. This indicates to the file system that a file has been deleted and is no longer accessible (or maintained) by the file system—yet most of the metadata and the entire file content remain until overwritten.

File systems use contiguous writes if possible. Most operating systems write data to the drive in a contiguous set of data blocks or sectors if available. A given data file, provided it is not modified after being written to the disk, tends to have all the data in sequentially accessible sectors. This speeds up both the write and read processes, since the heads on the drive do not need to move to different areas on the disk to access data. This plays a role in data recovery, in that data from a given file has a high likelihood of being grouped together on the disk in contiguous data blocks. When the residual metadata is incomplete, deleted file recovery tools exploit the contiguous block allocation by file systems as a basis to guess which blocks belonged to the deleted file.

A tool may be able to detect that a file was present on a file system and that the file has been deleted, but it is not possible for a tool to recover the content of a deleted file with complete certainty. A tool may be able to construct a guess at the original file with some limitations. Such a guess may include a file system object metadata record, file name, file size and a set of data blocks that at one time were allocated to the file. Some results that a deleted file recovery tool may produce are the following:

- Some or all of the original blocks allocated may be identified in the file system metadata and assigned to the recovered object.
- If a file size is identified, but not enough data blocks are identified in the file system metadata, then the tool may assign additional blocks to the recovered object based on known block allocation strategies of the file system.
- The file name of the reconstructed file existed at some time but may not have been the name associated with the recovered object.
- A tool may be able to infer from file system metadata that a given block has been overwritten.

3 Deleted File Recovery Tool Requirements

It is difficult to formulate a testable set of deleted file recovery requirements that apply across all file systems because each file system implementation leaves behind a different set of metadata after a file is deleted. A further complication is that different instances of the same file system may have differences in residual metadata due to the settings of optional file system parameters. Experiments with widely used forensic tools determined that metadata elements remaining after a file is deleted is usually a subset of the following items:

- File name (some file systems keep a second short form (8.3) in addition to a long file name),
- File size,
- Location of first data block,
- Location of remaining data blocks,
- MAC times (some file systems may keep additional dates and times; some file systems and operating systems may have different interpretations of MAC times), and
- File attributes such as access permissions and file ownership.

For example, a file deleted from a FAT file system has a partial file name, file size and location of the first data block available, but not the location of the remaining data blocks. Some tools make a guess at the file content by including in the recovered file enough free blocks after the first block so that the recovered object is the same size as indicated by the residual metadata. On other file systems, e.g., ext2, the location of all the data blocks may be available, but the association with the file name is not kept. In this case, an unnamed *lost file* is recovered with the correct size and the

originally allocated data blocks. Of course, there is no guarantee that one or more of the data blocks have not been overwritten by another file.

The lack of uniformity in the residual metadata leads to a complicated set of requirements where each file system type requires slightly different tool behavior because of what is possible to recover varies for each file system type. This also leads to regular revision of any requirements as file systems evolve over time or new file system types are introduced. To make the requirements easier to manage and, more importantly, to make the reports easier to read, we developed a strategy based on idealized requirements. We ignore the differences among the various file system types and instead write the requirements for an ideal file system that leaves in residual metadata all information required to reconstruct a deleted file. Following this strategy has the consequence that sometimes a tool is tested against an impossible-to-meet requirement for a particular file system. This is not really a problem if one keeps in mind that such test results are just a characterization of tool capabilities for each tested file system. It does not really matter for the tool user if a tool cannot do something because the tool failed to implement the feature or if the feature cannot be implemented; the feature is not available to the user and the test report documents the lack of the feature. However, tool test reports using such idealized requirements must clearly state that if a tool does not meet a requirement it may be because it is not possible for any tool to meet the requirement for a given file system. Otherwise, the incorrect conclusion might be made that the tool could meet the requirement if only the tool were better written.

There are several possible formulations of deleted file tool requirements, the following list of requirements is for a tool operating on an ideal file system that after a file is deleted, there is sufficient residual metadata to completely and accurately reconstruct the deleted file:

1. The tool shall report residual metadata for each deleted file entry.
2. The tool shall report file names with the characters that correspond to the representation used by the file system, i.e., a file name stored as one of the hexadecimal strings, D1 87 D0 B0 D0 B9 (UTF-8) or 04 47 04 30 04 39 (UTF-16 big endian), should be rendered as: чай (Russian for *tea*).
3. The tool shall construct a recovered file the same length as the original deleted file.
4. The tool shall construct a recovered file containing the same data blocks as the original deleted file.
5. The blocks of the recovered file shall be in the same order as the original file.
6. The tool shall identify a file as overwritten if and only if at least one block does not contain the same content as the original file at the time of deletion.

4 Verifying Conformance to Requirements

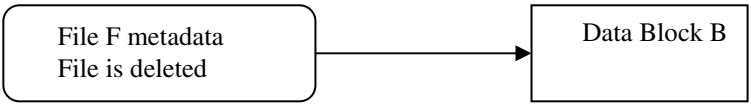
Verifying that a tool meets the deleted file requirements is accomplished by creating a set of test cases with corresponding data sets covering a variety of relationships between data blocks and metadata entries that provide the tool with opportunities to fail. Simply creating a few files, deleting the files and then trying to recover them is not adequate because very few of the actual relationships that could exist between metadata and file blocks are created by such a simple strategy. Test data and test cases need to present a tool with opportunities to examine a wide variety of relationships that can be encountered between metadata and file blocks.

The tool under test is viewed as operating on the image file of either a logical (partition) or physical (entire drive) acquisition. The image is a collection of data blocks, file system metadata and directory entries. The state of the data blocks, file system metadata and directories is determined by a sequence of *file system operations*. A sequence of operations from the following list of abstract operations sets the state of the file system. Most realistic file system operations can be mapped to these abstract operations:

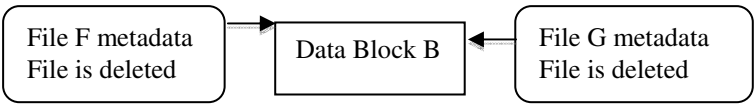
FormatFileSystem (BlockSize, FileSystemSize, FileSystemType)
CreateDirectory (ParentDirectory/Directory)
CreateFile (Directory/File, Size)
CreateOtherObject (Directory/Object, Options)
DeleteObject (Directory/File)
AppendFile (Directory/File, Size)
SetFileAttribute (Directory/File,Attribute, Value)

The tools behavior is described by the content of the recovered object. The core requirement is to construct the recovered object for a given target from any tagged data blocks still associated with the target. We considered the possibility of creating custom tools to manipulate file system metadata to create relationships between metadata and data blocks, but rejected the approach because it would require significant investment in resources to create such a tool for several file systems and might introduce anomalies into the test image file system that would never occur in an unmodified file system. Some basic relationships among blocks and metadata follow:

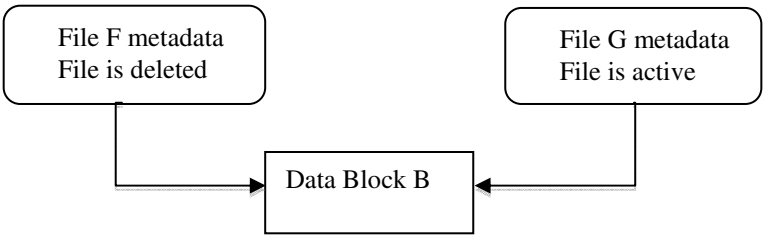
- A block tagged only by one deleted file and no allocated file.



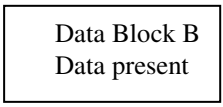
- A block tagged by more than one deleted file and no allocated file.



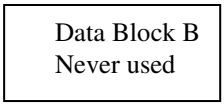
- A block tagged by one deleted file and also an allocated file.



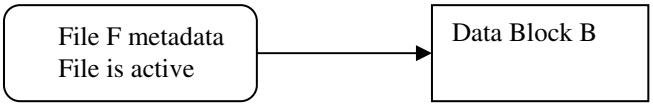
- A block not tagged by any deleted file, but has been allocated in the past.



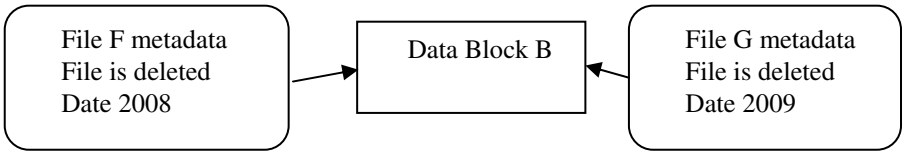
- A block not tagged by any deleted file and has never been allocated in the past.



- An allocated block not tagged by any deleted file.



- A block tagged by more than one deleted file such that time metadata implies that one of the files is more recently written.



If a block is tagged to a directory entry for a deleted file then we know that at some time in the past the data block belonged to the file. The current content of the data block may or may not still be the same as the original content. The block may have been overwritten with new content. While it is not possible to determine the file to which the content belongs, it is possible to determine files to which the content cannot belong. For a given block, B, and a deleted file, D, tagged to B then the following conditions apply:

- If D is the only deleted file entry tagged to B, then B might belong to D.
- If B is allocated to another file, A, that is active, then B cannot belong to D since A has overwritten block B.
- If B is also tagged by another deleted file, F, and the creation date and time of F is more recent than D, then the current content of B cannot belong to D since F has overwritten block B.

5 Creating Test Data and Test Cases

If good forensic examination practices are followed to create test data sets, the test drives are initially wiped with zeros, the latest version of an operating system is installed and realistic files that are encountered in day to day case work are created and deleted. The data set produced by imaging such a drive may be quite realistic. However, trying to identify specific tool behaviors would be difficult due to a fog of details, ambiguities and unknowns. Realistic test cases are valuable, but should not be used exclusively. To tease out specific tool behaviors a precisely controlled data set is required.

There is no difficulty in evaluating a recovered file if it matches the original, but to characterize an imperfect recovery easily requires that the assembled components can be traced to their original location. This can be accomplished by tagging each 512 byte block allocated to a file with the following items:

- File name
- File path
- Block sequence number within the file

Before formatting a file system and creating any files, marking all blocks of the storage device with the string “not used” should initialize the device where the file system will be created. After formatting and creating files, any blocks with other content would indicate metadata blocks. This makes it immediately clear where each block of a recovered file originated.

Part of the setup for each test case is the setting of pretest conditions so that the tool under test has coverage of possible relationships between metadata and data blocks that may be encountered by the tool. The following table lists conditions that may be significant to tool behavior:

Condition	Values
Block size (BS)	$1 \le BS \le BS_{\max}$ Where $BS_{\max}$ is the largest block size allowed.
Fragmentation	None (contiguous), two fragments, multiple fragments.
Overwritten	Overwritten, not overwritten.
Tagged block	None, one deleted object, two deleted objects, both a deleted and an allocated object.
Acquired as	Logical, Physical (one file system image), physical (several file systems in one image).
FS-Object	File, directory, link, shortcut, alternate data stream.
Tag locality	Same directory, different directory.
Recovery Load	Light (a few files), heavy (lots of files).
File System	FAT (3 subtypes: 12, 16, 32), NTFS, EXT (3 subtypes: ext2, ext3, ext4), HFS, HFS+ (4 subtypes: plain, journaling, case sensitive, case sensitive with journaling.)
Character set	ASCII or non-ASCII (left-to-right, right-to-left, CJK, i.e., Chinese, Japanese and Korean).
Operating System	Native or non-native, e.g., delete file from FAT file system using MS Windows vs. from Linux.

Some conditions such as block size are set when the file system is created (formatted) and may or may not be user selectable. Additional images are also required to cover special situations or file system options. One example is the NTFS feature of storing small files within the master file table; another example is setting file system configuration options such as automatic file compression.

Several tools have been created to allow the creation of controlled file layouts and to characterize files for comparison after recovery:

- not-used – tag each sector of a device with the string “not used,”
- mk-file – create a file of tagged blocks,
- ap-file – append more tagged blocks to an existing file,
- fill-fs – allocate all free blocks to a single file,
- layout – categorize all blocks in the image of a file system as: file, unused, fill or metadata, and
- fana – file analysis (characterize and summarize file content to simplify comparison of a recovered file to the original file).

The general process for using these tools to create a test image is as follows:

1. Run the not-used program to mark each sector of a device.
2. Format the device with one or more partitions of the same family.

3. Synchronize the drive state by unmounting all partitions. This ensures that the current state of the drive is on the drive with no parts of the drive state only in memory.
4. Image the drive to capture the base state of the formatted file system. The base image serves as a reference point to identify the initial state of file system metadata.
5. Mount the file systems. The file systems are now ready to be manipulated in a controlled manner. File operations need to be grouped such that a smart operating system does not skip steps for efficient operation. For example, if we create a file and then delete the file, a smart OS may note that nothing needs to be written to secondary storage. This would undermine the effort to have something to actually recover. Operations are grouped into sets of actions such that no action should modify the result of another action within the same set. Between each set of actions, file systems are unmounted, imaged and remounted. The actual state of the file systems can be confirmed by examining the image before continuing to the next set of actions.
6. Use the mk-file program to create some files.
7. Unmount the file systems, image and remount.
8. Do additional actions (create and append) to achieve the relationship between data blocks and metadata required for the specific test image.
9. Use the fana program to characterize every file to be deleted.
10. Set MAC times for every file to be deleted.
11. Unmount, image and remount.
12. Record MAC times for every file to be deleted.
13. Delete the files.
14. Unmount and image the final state of the device. This final image is the test image.

The remainder of this section discusses creation for fragmented and overwritten files.

The above list of steps is just to create a test image of a single deleted file with no complicated relationships between file blocks and file metadata. To create a simple fragmented file steps 6—8 would look something like:

6. Create files A, B1 & C
7. Unmount, image & remount
8. Append B2 to B1

This creates (for a FAT file system) a block layout something like: A B1 C B2. With files A and C each with one block and file B with two fragmented blocks such that C is between the two blocks of B. Three possible tool behaviors (out of several) that have been observed in real forensic tools are the following:

1. Recover one block, B1. Since this is from a FAT file system, B1 is the only block tagged by the residual metadata. The tool quits after one block because that is all it knows for sure.

2. Recover two blocks, B1 B2. The tool sees that there should be two blocks and includes the next free block found.
3. Recover two blocks, B1 C. The tool sees that there should be two blocks and includes the next block after B1, free or (as in this case) not.

As another example, consider the following block layouts, files B and C are both deleted:

1. A B1 C1 C2 B2 D – C is nested within B. B was created first, then C. At a later time something was appended to B.
2. A B1 C1 B2 C2 D – C and B are intertwined. They might be two active log files.

File B will likely be recovered as B1 C1. File C in the first case will likely be recovered as C1C2 and C1B2 in the second. All these tool behaviors have been observed with our test images and widely used forensic tools.

It seems plausible that both of the above layouts would be common and none of the three recovery strategies would be always correct. In other words, these make good test cases because there is a good chance that interesting (incorrect) tool behavior is revealed by the test cases. This is especially true for FAT file system images because when a file is deleted from a FAT file system only the first file block is referenced by the residual metadata (any links to additional blocks are lost when the file allocation table links are cleared) and a tool has to guess to include additional blocks in the recovered file.

File layout in FAT file systems is easy to control. This is not the case in other file systems. For example, some file systems such as ext2 leave gaps for file growth between files. A fragmented file can be created if the size of the gap is known. The layout program can determine the size of the gap and then to ensure fragmentation, a sufficient number of blocks are appended to fill the gap plus a little bit more.

Overwritten files can be created using the fill-fs tool as follows:

1. Use the mk-file and ap-file to create a desired block layout.
2. Run the fill-fs program to allocate all remaining free file blocks.
3. Delete one or more files.
4. Create one or more files. Because the only free blocks are from the files just deleted in step 3, files created in step 4 overwrite these deleted files.

By varying the file sizes and the number of files deleted in step 3 different relationships can be created between residual metadata and data blocks when files are created in step 4 to overwrite deleted files. Some of the overwritten blocks are now referenced by metadata of both a deleted and an active file. Deleting the active file can create another relationship. By deleting the active file we now have a block referenced by two deleted files.

6 Summary

Digital forensic tools can exploit residual metadata remaining after files are deleted to reconstruct deleted files. However, the residual metadata is usually insufficient for a complete reconstruction of file content and metadata. For some file systems, e.g., FAT, a tool may have to guess which data blocks should be associated together in a file. A test strategy needs to identify likely relationships between residual metadata and data blocks, define sequences of file operations for creating these relationships, and provide tools for creating data files such that after a file is deleted and recovered, the source of all data blocks can be identified to ensure a useful characterization of tool behavior. This test strategy has been implemented by the Computer Forensic Tool Testing (CFTT) project at the National Institute of Standards and Technology (NIST) for testing the deleted file recovery feature of widely used forensic tools. As test images are created they are posted to http://www.cfreds.nist.gov.

Acknowledgments. The National Institute of Justice (NIJ), the Department of Homeland Security (DHS), and the National Institute of Standards and Technology's (NIST's) Law Enforcement Standards Office (OLES) and Information Technology Laboratory (ITL) support this work. The work is also supported by other organizations, including the Federal Bureau of Investigation, the U.S. Department of Defense Cyber Crime Center, U.S. Internal Revenue Service Criminal Investigation Division Electronic Crimes Program, the Bureau of Immigration and Customs Enforcement and U.S. Secret Service.

References

1. Carrier, B.: File System Forensic Analysis. Addison Wesley, New York (2005)
2. Garfinkel, S.L.: Carving contiguous and fragmented files with fast object validation. In: DFRWS, pp. S2–S12. Elsevier Ltd. (2007)