NIST Standards Compliance of Metadata-Based Deleted-File-Recovery (DFR) Tools

Andrew Meyer
Bowling Green State University
Bowling Green, Ohio, USA
apmeyer@bgsu.edu

Sankardas Roy
Bowling Green State University
Bowling Green, Ohio, USA
sanroy@bgsu.edu

ABSTRACT

Digital forensics (DF) tools are used for post-mortem investigation of cyber-crimes. National Institute of Standards and Technology (NIST) has set standards for DF tools. Compliance with the standards by DF tools is critical to ensure integrity of the outcome of forensics analysis. In this paper, we consider standardization of one class of DF tools for Deleted File Recovery (DFR). Via extensive experiments we evaluate frequently-used DFR tools on NIST standards, and find that tools do not satisfy some of the standards. We compile a comparative analysis of these tools, which could help the user choose the right DFR tool.

1. INTRODUCTION

Both in corporate and government settings, digital forensic (DF) tools are used for post-mortem investigation of cyber-crimes and cyber-attacks. Nowadays it is common [6] for law enforcement to use DF tools to follow an electronic trail of evidence to track down a suspect. To maintain the quality and integrity of DF tools, National Institute of Standards and Technology (NIST)'s Computer Forensics Tool Testing Program (CFTT) [9] set standards for these tools. Maintaining the standards of DF tools is especially critical for judicial proceedings: usage of a forensic tool that does not follow the standards may cause evidence to be thrown out in a court case, whereas incorrect results from a forensic tool can also lead improper prosecution of an innocent defendant.

The focus of this paper is about standardization of one class of DF tools that are for Deleted File Recovery (DFR) [2]. As the name suggests, a DFR tool attempts to retrieve deleted files from a file system of a computer. As an example, given a hard disk or a USB drive (which might have been seized from the suspect computer or collected from the crime scene), a forensics professional can use a DFR tool to investigate about (and potentially retrieve) files which a suspect deleted to hide important information. The success or failure of a DFR tool can decide the outcome of a case.

DFR tools are typically classified as one of two varieties, corresponding to two different approaches to file recovery. These varieties are *metadata-based* tools and *file carving* tools. The focus of this paper is metadata-based DFR tools, with file carving left for future work. In the rest of the paper, unless otherwise mentioned, by *DFR tool* we mean metadata-based DFR tool.

Our experiments with a popular DF tool suite named Autopsy [7] show that it does not satisfy all NIST standards for DFR. Furthermore, we extensively experimented with other frequently used DFR tools. We compare those tools' performance and compile a comparative analysis, which could help the user choose the right DFR tool.

Evaluating the performance of a DFR tool is complex because many elements of a forensics scenario determine the success or failure of the file recovery process. A few such factors are the type of the file system (FAT, NTFS, etc.), presence of other active/deleted files in the file system, fragmentation of a file, a deleted file being overwritten by another file, and so on. So, comparison of two DFR tools is scientific only if they are compared while keeping these factors same. Via extensive analysis, we design a set of test file system images (for either of FAT and NTFS) which considers each of the above factors independently. We claim that this list of test cases is exhaustive and thus claim that our evaluation gives a complete picture.

Proceedings of the 2019 Midstates Conference on Undergraduate Research in Computer Science and Mathematics

As there are many file systems (e.g., ext4, HFS, etc.) in addition to FAT and NTFS, one might be interested to know why we chose FAT and NTFS for the current work. Because FAT and NTFS are very widely used on external storage devices and devices running Microsoft Windows, respectively, real-life forensics investigation often involves these two file systems. While we leave other file systems for future study, our current methodology could be extended to other file systems to make a similar study.

The main contributions of the paper are listed as follows:

- We design and build an exhaustive list of canonical test file system (FAT and NTFS) images to evaluate the DFR tool per NIST standards.
- We perform evaluation of frequently-used DFR tools (including free tools as well as proprietary ones) on the test images.
- For the interesting cases of tools' success or failure, we provide logical explanation.
- We provide critique on applicability of some of the NIST standards in a practical setting.

The NIST CFTT portal currently publishes reports of only a subset of DFR tools. However, that set needs to be expanded as many new tools come to market and become popular. Also, existing DFR tools should be retested to ensure their reliability is consistent as new patches and features come out. Adding new reports to the CFTT website will allow tool developers a chance to continually develop their tools for the better. We will submit our study reports to the CFTT portal. At the time of writing, it does not publish test reports for any of the tools we tested.

2. RESEARCH QUESTIONS

A DFR tool is a piece of software that can retrieve residual data of a file that was deleted from a storage device (e.g., computer hard disk, flash drive, and so on). We evaluate a set of popular DFR tools on the scale of CFTT standards. In particular, following are the research questions (RQs) that we target to answer.

RQ1. Do the popular DFR tools (as available in the market) meet the NIST CFTT standards? If not, which tool meets which part of the standard?

RQ2. What factors make the tools fail or succeed?

RQ3. Are the free DFR tools more effective compared to the enterprise-level (proprietary) tools?

The identification of errors, such as for not recovering a deleted file or attempting to recover a file that was never there (Type I and Type II errors, respectively), is an important metric for a DFR tool. Type I and Type II errors account for majority of the standard. Many factors impact the performance of a DFR tool, including file system type, whether the file content is not located in contiguous clusters, whether some part of the deleted file content is overwritten by another file, and more. We consider these variables in the design of experiment when we compare the tools. Note that our current study is limited to exploring the *core features* of NIST standards [2], i.e., we leave the optional features [2] of NIST standards for future study.

3. BACKGROUND

In this section, we present some of design basics (often simplified to aid readability) of FAT file system and NTFS file system, which are relevant to what we discuss in the latter part of the paper. We highlight what information remains in the file system after a file is deleted, which leads to understanding of how metadata-based file-recovery might work. Finally, we present the *core features* of NIST standards for such recovery tools.

3.1 FAT File System

As in many other file systems, a file in FAT system has metadata in addition to the actual file content. The main metadata of a file (say foo.txt) in FAT system is called a *directory entry*, which is of 32 bytes. The directory entry of a file (say foo.txt) has three main *essential* elements: (a) file name, (b) file size, and (c) index of the starting cluster (which holds the actual file content). We can figure out the index of other clusters of the file by reading a global table called the FAT table. The FAT table can determine the chain of clusters which hold the data of a file. In particular, for each cluster (say x) of the file system, the FAT table has an entry, and if FAT(x) is 0, then that means cluster x is currently unallocated. On the other hand, if FAT(x) is y, then that means cluster y is the next cluster after cluster x (as part of the same file). If FAT(x) is

EOF, that indicates cluster x is the last cluster of the file.

As an example, the directory entry of foo.txt and the clusters (holding the actual content of foo.txt) are illustrated in Figure 1. The FAT table is also shown, which tells us that foo.txt's data is stored in contiguous clusters, starting from cluster 100 and ending at cluster 200.

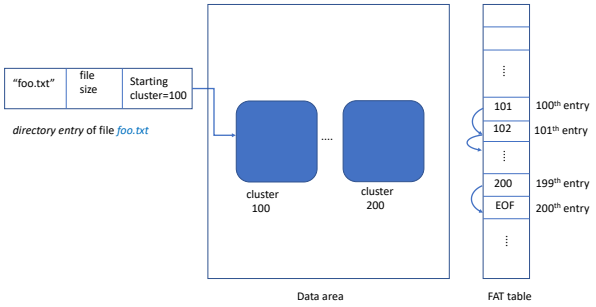


Figure 1: File foo.txt in FAT. The *directory entry* of this file and the actual file content clusters (shaded) are shown. The FAT table is also shown, which determines the chain of clusters (from cluster 100 to cluster 200) of foo.txt.

The change in metadata and actual content of foo.txt is illustrated in Figure 2 after the file is deleted. The first character of the file name (say "foo.txt") is flagged ("_oo.txt") to denote that the file is deleted, but the remaining part of the directory entry can be still available. In the FAT table, the deleted file's corresponding entries are *zeroed*, which denotes that those clusters are available to be allocated to a new file (if necessary). However, the actual content carrying clusters (say cluster 100 to cluster 200) can still be intact until they are *overwritten* by another file.

Furthermore, it is possible that the content of a file is not stored in contiguous clusters in FAT file system, and this phenomenon is called *fragmentation*. If the original file foo.txt has two fragments, it may look as illustrated in Figure 3 where the file's first fragment is from cluster 100 to cluster 101 and the second fragment is from cluster 200 to cluster 298.

3.2 NTFS File System

In an NTFS system, each file has an entry in the Master File Table (MFT) where every entry is typically 1024 bytes. If a file is short, then all of its metadata as well as the actual content can sit inside

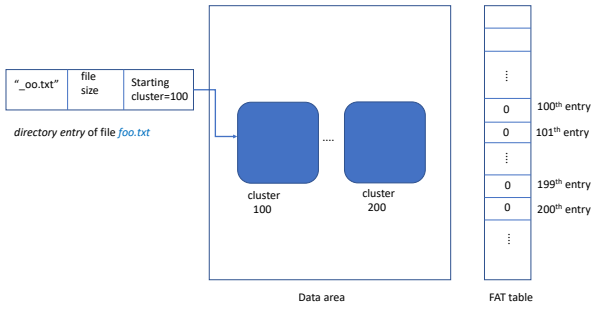


Figure 2: The metadata and actual content of foo.txt are shown after the file is deleted whereas the corresponding entries (i.e., 100-200) in the FAT table are zeroed.

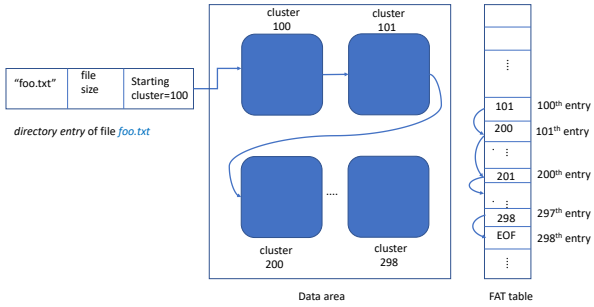


Figure 3: The layout of metadata and actual content of a file foo.txt is shown whereas the file has two fragments (cluster 100-101 and cluster 200-298) as determined by the FAT table.

the MFT entry; otherwise, the file content can be non-resident (i.e., not in MFT entry) and located in other clusters.

As an example, the MFT entry of foo.txt and the actual content carrying clusters are illustrated in Figure 4. The MFT entry indicates that there are two runs of clusters (100-101 and 200-298) which carry the actual file content. In this case, the example file has two fragments.

3.3 Metadata-Based Deleted File Recovery

The previous discussion implies that in many cases a deleted file's metadata (e.g., directory entry in FAT or MFT entry in NTFS) can be still available, and it is possible to recover the file content using this metadata. This is called metadata-based deleted file recovery.

For instance, in the example of Figure 2, we can see

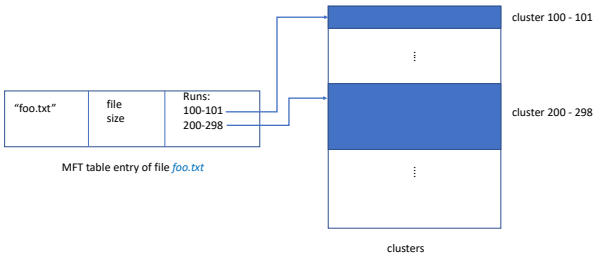


Figure 4: To illustrate NTFS file system, the MFT entry of foo.txt and the actual content carrying clusters are shown. This file has two fragments (cluster 100-101 and cluster 200-298).

from the directory entry of foo.txt that the deleted file starts in cluster 100 and has a size of 101 clusters; thus, we can reason that the deleted file content is in cluster 100 to cluster 200. We can recover the whole file via reading the raw content of these clusters (e.g., by using dd command in Kali Linux).

Note that in FAT system the *directory entry* of a file only refers to the starting cluster, and it does not carry any information about the file fragments. That is why in certain situations where the deleted file is fragmented, metadata-based file recovery in FAT system encounters challenges. On the other hand, if a file is fragmented in NTFS, the corresponding MFT entry does contain information on all the runs (i.e., fragments) of the file (as shown in Figure 4). Consequently, fragmentation does not introduce any extra challenge in file recovery in NTFS.

3.4 NIST Standards

The NIST standards[2] list four *core features* upon which metadata-based DFR tools are to be judged. Following is the text of each core feature as well as our interpretation of each in the context of this work:

1. "The tool shall identify all deleted File System-Object entries accessible in residual metadata."[2] We consider a tool passing this standard if it identifies to the user each file system metadata entry that is marked as deleted.

2. "The tool shall construct a Recovered Object for each deleted File System-Object entry accessible in residual metadata."[2] We consider a tool passing this standard as long as it outputs a file for each deleted file, even if the output file is empty.

3. "Each Recovered Object shall include all non-allocated data blocks identified in a residual metadata entry."[2] For FAT file systems, we consider a tool passing this standard if it recovers at least the first contiguous segment of unallocated sectors starting from the first sector originally allocated to the deleted file. For NTFS file systems, the tool must recover all unallocated sectors originally allocated to the deleted file.

4. "Each Recovered Object shall consist only of data blocks from the Deleted Block Pool."[2] We consider a tool passing this standard if the recovered file consists only of data from the original deleted file, or null data to represent omitted portions.

4. APPROACH

4.1 Overview

To test the DFR tools, we first design hypothetical test scenarios to simulate the challenges of real-world file recovery. We then create each scenario in real file systems and save them as raw images. Using the images as input, we run each DFR tool and attempt to recover all deleted files. Finally, we compare the recovered files to their original versions in order to judge the tools' compliance with the NIST standards. A high-level view of the methodology for a typical test case is illustrated in Figure 5.

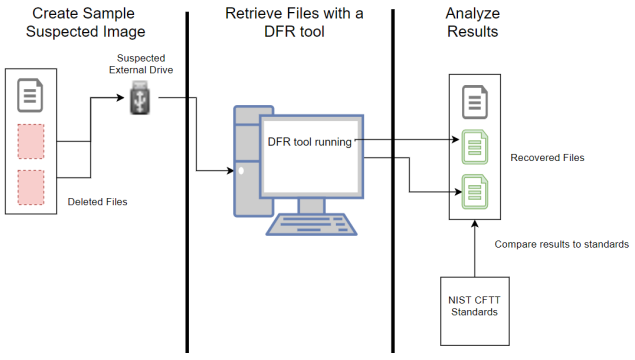


Figure 5: A file system containing deleted files is created on the external drive. That file system is then given as input to a DFR tool, which attempts to recover the deleted files. The recovered files are then analyzed to judge the DFR tool's compliance with the NIST CFTT standard.

4.2 Designing Recovery Scenarios

To test the DFR tools' compliance with the standards, we designed a variety of scenarios in which a tool might have to recover a deleted file. We started with the simplest possible case: a file system containing just one deleted file. This case is ideal and trivial, but by adding more files, we can create a greater variety of scenarios.

The NIST standards limit their scope to recovery of files which were "created and deleted in a process similar to how an end-user would create and delete files,"[2] and exclude "files and file system metadata that is specifically corrupted, modified, or otherwise manipulated to appear deleted."[2] In other words, these standards address situations in which files were deleted via normal file system operations as implemented by a typical operating system, as opposed to direct modification of the file system by a user. Within these constraints, there are two factors which can significantly complicate the file recovery process: fragmentation, and overwriting.

These factors are thus the foundation of our test scenarios, with all cases besides the first involving either fragmented files, overwritten files, or a combination of both. The goal is to create test cases which are canonical; in other words, they constitute the basic elements of a file recovery scenario. We suggest such a canonical list of test cases should be considered representative of all possible scenarios within the scope of the standards.

Following are descriptions of the test cases we designed. When test cases are illustrated in figures, each row represents the file system at a point in time. An arrow indicates a change in the file system, which is illustrated in the following row. Files are distinguished by unique letters, and if they are deleted they are marked as such.

1 The file system contains a single deleted file

2 Fragmented deleted file, with an active file in between the fragments (as illustrated in Figure 6)

3 Fragmented deleted file, with a deleted file in between the fragments

4i Beginning of deleted file overwritten by an active file

4ii Middle of deleted file overwritten by an active file (as illustrated in Figure 7)

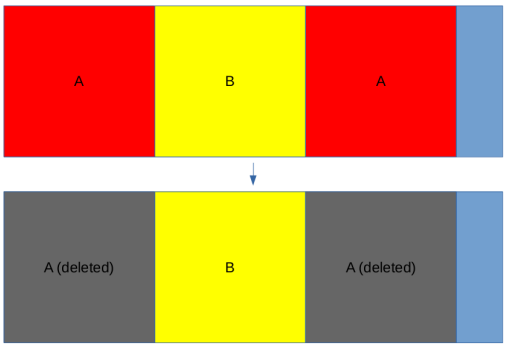


Figure 6: Test Case 2. Fragmented file A is deleted.

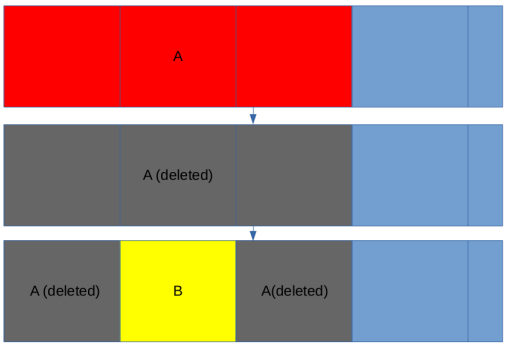


Figure 7: Test Case 4ii. Deleted file A is partially overwritten by active file B.

4iii Deleted file partially overwritten by an active file which doesn't end on a sector boundary

4iv Deleted file entirely overwritten by an active file

5i Beginning of deleted file overwritten by another deleted file (as illustrated in Figure 12)

5ii Middle of deleted file overwritten by another deleted file

5iii Deleted file partially overwritten by a deleted file which doesn't end on a sector boundary

5iv Deleted file entirely overwritten by a deleted file

6 Fragmented deleted file, with an active file in between the fragments, with the second fragment overwritten by another active file (as illustrated in Figure 8)

7 Fragmented deleted file, with an active file in between the fragments, with the second fragment overwritten by another deleted file

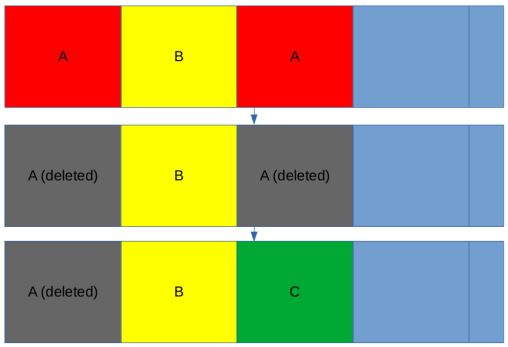


Figure 8: Test Case 6. Fragmented file A is deleted. File A is then partially overwritten by file C.

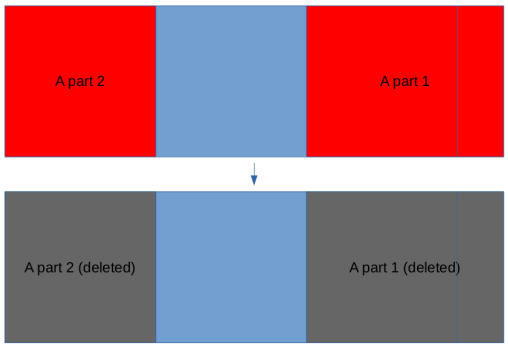


Figure 9: Test Case 8. Fragmented file A, which starts at the end of the file system and wraps around to the beginning, is deleted.

8 Deleted file fragmented from the end of the file system to the beginning (as illustrated in Figure 9)

9 Deleted file fragmented from the end of the file system to after an active file

10 Deleted file fragmented from the end of the file system to after a deleted file

Because NTFS keeps track of the locations of all parts of a file even after deletion, fragmentation is not particularly interesting. Cases 8, 9, and 10 would be redundant with case 2, so we have excluded them for NTFS. Due to how NTFS allocates space for files, cases 4ii and 5ii cannot occur as a result of normal file operations, so they have also been excluded. No cases are excluded for FAT tests.

4.3 Creating Test Images

All test file systems were created in partitions on a 32 GB flash drive. For each test case, the first step is to entirely write over the partition with zeros. This ensures all cases start from identical, reproduceable conditions. A new file system is written to the partition, then new files are written to the file system and deleted. The files used are simple text files containing one letter repeated (e.g., "aa1M" is 1 MiB of the letter 'a'). Files are written to the test file system by simply copying them from another drive. In some cases we also append data to a file in the test file system to create fragmentation. Once the test file system matches the intended scenario, a read-only image of the partition is created. All tests are performed on these images rather than the original drive. Note that when creating FAT test cases we use Ubuntu 18.04 and for NTFS test cases we use Windows 10.

4.3.1 Challenges

It is important to consider when creating test images that the low-level behavior of file operations is not always obvious. For example, when writing a file, there is no guarantee the file's data will be immediately written to the disk. The operating system may cache the operation and wait until the optimal time to perform the write, in order to maximize system performance. We observed this early on, as writing a file and subsequently deleting it would always result in the file's metadata being written, but often left no evidence of the file's data having ever existed. This behavior is obviously undesirable because it leaves nothing meaningful to be recovered. We resolved this by using the *sync* system call, which causes any such cached data to be immediately written to the disk, in between file writes and deletions. Unmounting the file system after writes has a similar effect.

Another type of low-level behavior relevant to the image creation process is the allocation algorithm. The operating system must have some kind of algorithm to decide where in the data area new files should be written. Common allocation algorithms include "first available," "next available," and "best fit." Learning and understanding whatever algorithm the OS uses is very helpful for forcing a specific arrangement of files. We observed that when writing to a FAT file system, Linux uses a "next available" algorithm. After the file system is mounted, the first write will start at the first free space in the data area. The next file will be written starting from the first free space after the previous file. Meanwhile, when writing to an NTFS file system, Windows 10 appears to use a "best fit"

algorithm. In this case, Windows tries to find the smallest space in which the file can fit without being fragmented, and write it there.

4.4 Recovering Files

We selected five popular DFR tools for testing: Autopsy[7], Recuva[8], FTK Imager[4], TestDisk[3], and Magnet AXIOM[5]. Note that Autopsy uses a set of DF tools known as The Sleuth Kit (TSK) for metadata-based recovery, so TSK is also implicitly covered by this study.

The settings we used when testing each tool are as follows: For Autopsy, we performed a standard recovery with all ingest modules disabled. For Recuva we performed a standard recovery using the free version with default settings. For FTK Imager, we performed a standard recovery using the free version with default settings. For TestDisk we used the "file undelete" feature under "Advanced Filesystem Utils." For Magnet AXIOM we performed a "full scan" in AXIOM Process and exported all files accessible in "Filesystem View" in AXIOM Examine.

4.5 Results

After testing each tool, we analyzed the recovered object(s) from each test case. If the recovered file is identical to the original, obviously all standards have been met. While this is ideal, it is often impossible to perfectly recover a file (such as when it is overwritten) so the standards do not require it. In our results, the file is only ever recovered perfectly in FAT cases 1 and 2, and NTFS cases 1-3. For all other cases, the tool is judged on each core feature individually. These judgments are summarized in Figure 10 for FAT test cases and Figure 11 for NTFS test cases.

For cases in which a tool does not fulfill core feature 1, in other words, it cannot find a deleted file, we make no judgment about the remaining core features.

4.5.1 Recovering Fragmented Files

In cases of fragmentation in FAT file systems, we found each tool generally approaches recovery in one of two ways. Recuva and Magnet AXIOM start from the beginning of the file and recover the full length of the file even if an active file exists in that space. Autopsy, FTK, and TestDisk will start from the beginning of the file and recover the full length, but skip over any active files they encounter. Autopsy, FTK, and TestDisk recover all of file A, while Recuva and Magnet AXIOM's recovered images erroneously contain data from file B, causing them to fail core feature 4. When the space in between fragments is unallocated, all tools recover the file as though it was contiguous, pulling some erroneous data and failing core feature 4. When the fragmentation occurs at the end of the file system, Recuva, FTK, and TestDisk recover only the first fragment, while Autopsy returns a short file of null data, and Magnet AXIOM reports an error and returns an empty file. Cases with fragmentation are trivial for NTFS file systems as more information is available from the metadata. Unsurprisingly, no tools had problems with fragmentation cases for NTFS.

4.5.2 Recovering Overwritten Files

In cases where a file has been overwritten by an active file, we found most tools recover the deleted file as though it is not overwritten, failing core feature 4. A few exceptions are FTK Imager, which recovers the file up to the point where it has been overwritten, and Autopsy, which generally recovers only the first cluster of an overwritten file in FAT, and behaves like the other tools for NTFS. TestDisk also exhibits the same behavior as FTK for FAT case 4ii only. Strangely, Magnet AXIOM's recovered objects for FAT cases 4i and 4ii include the overwritten sections, but nothing after them. Other Magnet AXIOM results were similar to the other tools. When the overwriting file has also been deleted, all tools recover the first file as though it is not overwritten.

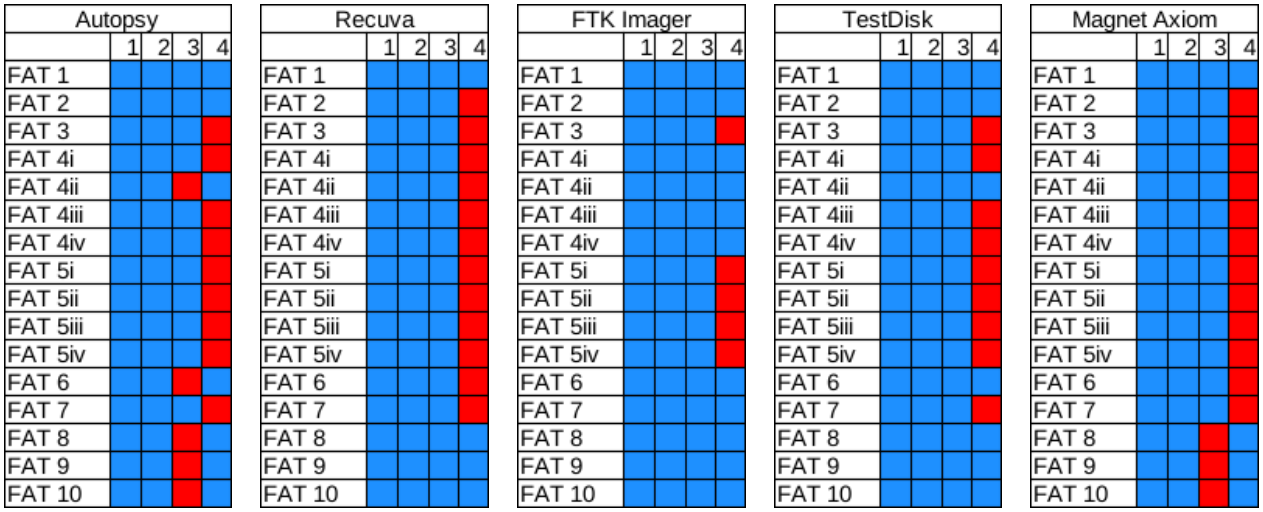
4.5.3 Abnormal Results

A few results stand out as unusual. These are cases for which it is difficult to infer from the recovered object what approach a tool is using.

For FAT cases 4ii, 6, 8, 9, and 10, Autopsy returns a 1.5 KiB file of null data. 1.5 KiB is equivalent to 3 sectors, while a FAT cluster in our cases is defined as 4 sectors (2 KiB).

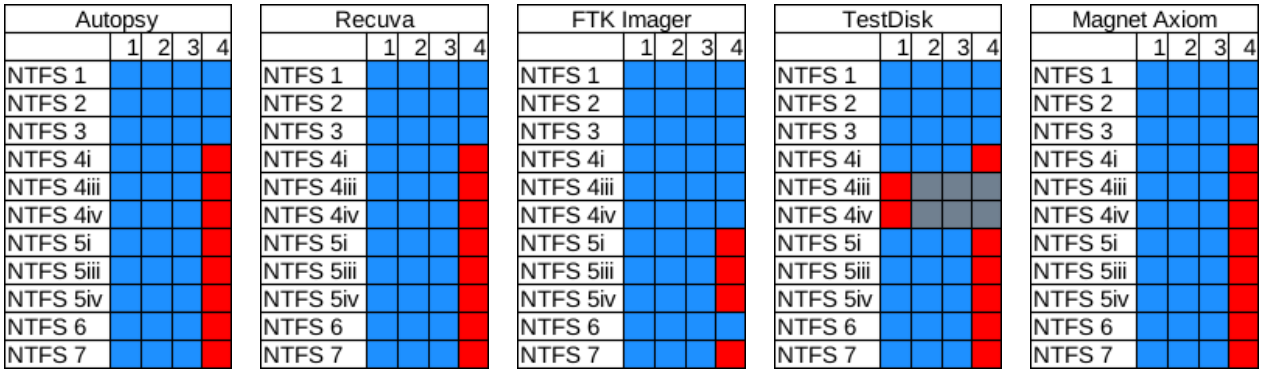
TestDisk fails to identify a file for NTFS cases 4iii and 4iv only. These are the only test cases in which a tool does not fulfill core feature 1.

For FAT cases 4i and 4ii, Magnet AXIOM does not recover the entire length of the deleted file, but it also does not exclude the overwritten sections. In both cases, it recovers up to the end of the overwritten sections, rather than up to the beginning



(a) Autopsy (b) Recuva (c) FTK (d) TestDisk (e) Magnet AXIOM

Figure 10: Test results on FAT test cases for each DFR tool. Rows represent test cases whereas columns represent NIST core features. Blue is passing, red is failing, gray is not tested.



(a) Autopsy (b) Recuva (c) FTK (d) TestDisk (e) Magnet AXIOM

Figure 11: Test results on NTFS test cases for each DFR tool. Rows represent test cases whereas columns represent NIST core features. Blue is passing, red is failing, gray is not tested.

like FTK does.

5. DISCUSSION

5.1 Answering Research Questions

5.1.1 RQ1

Do the popular DFR tools (as available in the market) meet the NIST CFTT standards? If not, which tool meets which part of the standard?

Generally, we found that the DFR tools we tested did not consistently meet the NIST CFTT standards. TestDisk[3] failed to fulfill core feature 1 because it did not identify deleted files in two test cases. All tools fulfilled core feature 2, as they produced a recovered object for every deleted file they identified. Autopsy[7] and Magnet AXIOM[5] failed to fulfill core feature 3 because in several test cases they did not recover data they had access to. All tools failed to fulfill core feature 4 because in many cases they recovered data which was not part of the original file.

5.1.2 RQ2

What factors make the tools fail or succeed?

The most common factor causing tools to fail is when a deleted file has been overwritten. Core feature 4 requires that a tool only recover data which was originally a part of the deleted file. A tool's success regarding this feature is thus a measure of its restraint. The only tool to consistently fulfill core feature 4 is FTK Imager.[4] When it detects that a file has been partially or completely overwritten by another file, it omits the deleted sec-

tions (and everything after them in FAT). However, in cases when the overwriting file has also been deleted, even FTK fails to fulfill this core feature. It is worth noting that Autopsy does appear to react to overwritten files; for some cases of overwriting in FAT, it returns only a single cluster, presumably the starting cluster of the deleted file. However, since that cluster has been overwritten, Autopsy still fails to fulfill core feature 4 in those cases.

Another factor that causes multiple failures is simulated in FAT cases 8, 9, and 10. In FAT, a file can be written starting close to the end of the file system, without enough space to fit contiguously. In such cases, the file must be fragmented, and since it is already at the end of the file system, the second fragment will appear closer to the beginning of the file system (this is illustrated in Figure 9). This scenario could realistically occur when the file system is almost full. In these cases, no tool is able to recover the second fragment of the deleted file; however, because FAT fragmentation is unpredictable, we only require them to recover the first fragment. Interestingly, Autopsy and Magnet AXIOM fail to recover anything at all, with Autopsy returning a short file of null data, and Magnet AXIOM returning an empty file after displaying an error message.

5.1.3 RQ3

Are the free DFR tools more effective compared to the enterprise-level (proprietary) tools?

As can be observed in Table 1, no type of tool clearly outperforms the others. FTK Imager, a proprietary enterprise-level tool, passes the most test cases by a large margin. However, the other enterprise-level tool, Magnet AXIOM, passes the least test cases. Given the available data, we cannot reach a definite conclusion for RQ3.

Table 1: DFR tools sorted by type. "Passes" refers to the number of test cases in which a tool fulfills all 4 core features.

DFR Tool	Type	Passes
Autopsy	free open source	5
TestDisk	free open source	10
Recuva	proprietary freemium	7
FTK Imager	proprietary enterprise	18
Magnet AXIOM	proprietary enterprise	4

5.2 Ambiguity in Standards

While determining how to interpret the NIST standards, we encountered ambiguities in their current wording.

5.2.1 Core Feature 3 and FAT Fragmentation

Core feature 3's requirement that a tool recover "all non-allocated data blocks identified in a residual metadata entry"[2] is not well-defined when considering a FAT file system. In FAT, the only relevant metadata left over after file deletion is the address of the first cluster of file data, and the file's length. If the deleted file is fragmented at any point, no evidence remains in the metadata. Therefore, interpreting the wording very closely, a tool is only required to recover the first cluster of a file's data. As this would not be particularly useful, it is unlikely that this was the intended meaning. For these tests we interpret core feature 3 as requiring the first contiguous segment of unallocated clusters starting from the first cluster originally allocated to the deleted file. In other words, if the file is fragmented, the tool must recover at least the first fragment. If a file is partially overwritten, the tool must recover at least the clusters before the overwritten part. In essence, the tool is only required to recover data for which it does not have to guess what file the data belongs to. However, it should be emphasized this is an assumption and the intent of the standard in this case needs clarification.

5.2.2 Contradictory Core Features

When designing test cases, we found situations in which core features 3 and 4 are entirely incompatible. Core feature 3 specifies "all non-allocated data blocks identified in a residual metadata entry,"[2] but that can sometimes still include data from other files. One such situation is when a deleted file is overwritten, and then the overwriting file is also deleted, such as in case 5i (as seen in Figure 12).

Assuming the file system is NTFS (to avoid the aforementioned ambiguity with core feature 3 and FAT), the residual metadata entry for File A (in this case its Master File Table entry) should list every cluster File A once occupied. All of those clusters are unallocated, so to fulfill core feature 3, the tool must recover them. However, some of those clusters have been overwritten by File B. Core feature 4 requires that a tool only recover "data blocks from the Deleted Block Pool,"[2] and defines the Deleted Block Pool as all blocks which

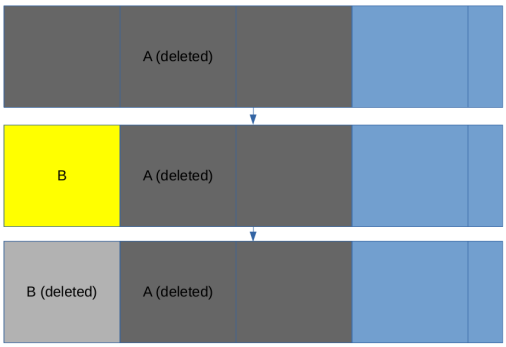


Figure 12: Test Case 5i

"have not been reallocated or reused."[2] Core feature 3 would require tools to recover the clusters reused by File B, while core feature 4 would forbid this. It could be argued that the tool should use File B's metadata to recognize that File B overwrote File A, but this is not always realistic. While the file system stores information such as creation and modification times, this is not "essential metadata," meaning it is not involved in the operation of the file system, so operating systems may implement it inconsistently, or not at all.[1] Since the time information cannot be counted on to be reliable, there is no way to know for sure which file overwrote which. It is also possible for File B's metadata entry to be overwritten at some point before File A's, in which event there is no way for the tool to know File B even existed.

The standards document acknowledges that the "potential for corruption [is] inherent with data that is no longer maintained by a file system,"[2] and that the recovered object "may not completely match the original FS-Object."[2] We propose the standards themselves should be revised to better account for such situations. This would most likely mean adding an exception to either the third or fourth core feature, for cases in which data blocks are overwritten and subsequently deallocated.

5.3 Future Work

The NIST CFTT standard[2] includes several optional features; these features could be explored using a similar methodology. We only created test images for the FAT and NTFS file systems; our process could be expanded to other common file systems such as ext4 and HFS. NIST CFTT has a separate set of standards for file carving DFR tools; future work could involve creating and evaluating test cases for file carving tools.

6. CONCLUSION

We designed an exhaustive set of canonical test cases to determine a metadata-based DFR tool's compliance with the NIST CFTT standards. We tested five popular DFR tools and evaluated their results. We presented a comparison of the tools based on the number of test cases for which each tool meets the standards. We concluded that none of the tested tools consistently fulfilled the NIST standards, and explain the factors which cause them to fail. We also identified potential weaknesses in the standards and suggested improvements.

7. ACKNOWLEDGMENTS

A. Meyer's work has been partially supported by a grant from BGSU's Center for Undergraduate Research and Scholarship (CURS) in summer of 2019. S. Roy's work has been partially supported by a NIST grant (grant number: 70NANB17H321; Year 2017-19) that he has been awarded with as a Co-PI. Quinton Currier at BGSU assisted with the testing process.

8. REFERENCES

[1] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley, 2005.

[2] NIST CFTT. Active file identification and deleted file recovery DFR tool specification: Version 1.1. `https://www.nist.gov/sites/default/files/documents/2017/05/09/dfr-req-1.1-pd-01.pdf`, March 2009.

[3] CGSecurity. TestDisk 7.0. `https://www.cgsecurity.org/wiki/TestDisk`.

[4] Access Data. Forensic Toolkit (FTK) 4.2.0.13. `https://accessdata.com/products-services/forensic-toolkit-ftk`.

[5] Magnet Forensics. Magnet AXIOM 2.7.1.12070. `https://www.magnetforensics.com/products/magnet-axiom/`.

[6] Sharon Gaudin. Digital trail led investigators to alleged craigslist murderer. `https://www.computerworld.com/article/2523694/digital-trail-led-investigators-to-alleged-craigs` `html`.

[7] The Sleuth Kit. Autopsy 4.11.0. `https://www.autopsy.com/`.

[8] Piriform Ltd. Recuva 1.53.1087. `https://www.ccleaner.com/recuva`.

[9] NIST. Computer forensics tool testing program (CFTT). `https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt`.