Open University of the Netherlands
faculty of Management, Science & Technology
Bachelor Computer Science

Interpreting NTFS Time-stamps

Author:
Jelle Bouma
851887770

Researcher:
Ir. Vincent van der Meer
Supervisor:
Dr. Ir. Hugo Jonker

August 1, 2019

ABSTRACT: Digital forensic investigators try to reconstruct events that have taken place on a computer system, to find evidence for use in court. To convict a suspect, it must be proven that the suspect has interacted with some illegal files. The most basic indication of when a file was used, are the time-stamps attached to the file. Time-stamps are important to digital forensics as they save the exact time that some operation occurred. However, from time-stamps alone it is not clear which file operations have happened at those times. This study aids digital forensic investigators as it explores the different kinds of file operations and how time-stamps can be used to find out which file operations may have happened at which times. The ways that time-stamps might have been tampered with to hinder digital forensics are also explored. Finally these findings are implemented in a tool that can match time-stamps with operations.

Contents

1 Introduction **5**

2 Background **8**
2.1 NTFS . . . 8
2.2 Master File Table . . . 8
2.2.1 $STANDARD_INFORMATION attribute . . . 9
2.2.2 $FILE_NAME attribute . . . 9
2.2.3 $DATA attribute . . . 9
2.2.4 $MFTMirr . . . 9
2.3 File operations . . . 9
2.3.1 File operations considered in this research . . . 10
2.3.2 Time-stamp changes . . . 11

3 Related Work **13**
3.1 File operations and their effects on time-stamps . . . 13
3.1.1 Does Windows version affect time-stamps? . . . 13
3.1.2 Does file type affect time-stamps? . . . 14
3.1.3 Consensus on the effect of operations on time-stamps . . . 14
3.2 Detection of forgery by comparing time-stamps . . . 14

4 Methodology **15**
4.1 MFT or $LogFile . . . 15
4.2 Tooling . . . 15
4.2.1 Reading the MFT . . . 16
4.3 Performing file operations and inferring their effects . . . 16
4.4 Validating file operations . . . 16

5 The types of resulting time-stamps for file operations **18**
5.1 The time at which the file operation starts: opStartTime . . . 19
5.2 The time at which the file operation ends: opEndTime . . . 19
5.3 An unchanged time-stamp . . . 19
5.4 A time-stamp transferred from the SI attribute . . . 20
5.5 A time-stamp transferred from another file entry . . . 20
5.6 A time-stamp transferred by file tunneling . . . 21
5.7 A time-stamp set to an arbitrary value . . . 21
5.8 Result modifier: rounding . . . 21
5.9 Result modifier: timezone difference . . . 22

6 Time-stamps as a Result of File Operations **23**
6.1 Extracting zip archives using Windows Explorer . . . 23
6.2 Overwriting move within volume . . . 24
6.3 Update MS Word/Office . . . 25
6.4 Overview of time-stamp changes by operations . . . 25

7 Modifiers to the Effects of File Operations on Time-stamps **27**
7.1 Last access updating . . . 27
7.2 File tunneling . . . 29
7.3 Transferring files from other file systems . . . 30
7.4 Unexpected behaviour of SI.E . . . 32
7.5 Operations per file type . . . 32
7.5.1 Executable files . . . 32
7.5.2 Directories . . . 32
7.6 Operations per operating system . . . 33
7.7 Accuracy of time-stamp names . . . 33

8 Time-stamp Forgery and Degraded Time-stamps **35**
8.1 Changing time-stamps to user-specified values . . . 35
8.1.1 The SetFileTime system call . . . 35
8.1.2 the undocumented NtSetInformationFile system call . . . 36
8.1.3 Editing the raw time-stamps data in the MFT to change time-stamps . . . 37
8.1.4 Detection . . . 37
8.2 Changing the system clock . . . 37
8.3 Time-stamp degradation . . . 38
8.4 Origins of irregular time-stamps . . . 38
8.5 Conclusion . . . 39

9 Investigating Time-stamps in the Wildfrag Database **40**
9.1 WildFrag database . . . 40
9.2 Future and far past time-stamps . . . 40
9.2.1 Volume 59 . . . 41
9.2.2 Amount of future or far past time-stamps per type . . . 41
9.3 The quirk: SI.C > SI.E . . . 41

10 Automating Comparison of Time-stamps with Operations **43**
10.1 Examples of rules derived from file operation overviews . . . 43
10.2 Structure . . . 44
10.2.1 Time-stamp result types . . . 44
10.3 Method . . . 45
10.3.1 Comparing an array of time-stamps to an operation . . . 45
10.3.2 Finding a possible sequence of operations for an array of time-stamps . . . 46
10.3.3 Finding all possible sequences of operations for an array of time-stamps . . . 47
10.4 Time-stamp Analyser . . . 48
10.4.1 Use . . . 48
10.4.2 Output . . . 48
10.4.3 Use case . . . 48
10.4.4 Testing . . . 50

11 Discussion **51**
11.1 MFT entry allocation . . . 51
11.2 The effects of file operations on the sequence number . . . 52
11.2.1 Possible use of the sequence number: uncover the original ordering of file entries . . . 52

12 Future work **53**

13 Conclusions **54**

A Reflection on Process **56**
A.1 Communication 56
A.2 Planning 56
A.3 Writing 57

B Overviews of File Operations, Modifiers and Their Effect on Time-stamps **58**
B.1 Base File Operations 58
B.2 Modifiers 59

C Diagrams **61**

Chapter 1

Introduction

Digital forensic investigators try to reconstruct events that have taken place on a computer system, to find evidence for use in court. Based on this evidence, a suspect might be declared either innocent or guilty of possessing and using some illegal files. The existence of illegal files on the suspect's hard drive is not enough evidence to convict someone, it must be proven that the suspect had some interaction with the files. For this, digital forensic investigators uncover when the illegal files were used and how. The most basic indication of when a file was used, are the time-stamps attached to the file. In this study we will explore how time-stamps are formed in the Windows operating system, and how the events leading to the time-stamps can be reconstructed.

A time-stamp is a date and time value recording when an event took place. However, information relating to the event is not recorded. Microsoft does not provide any details as to how the time-stamps are formed. In the documentation of Windows and NTFS (the file system used by Windows) they only state: "Time stamps are updated at various times and for various reasons. The only guarantee about a file time stamp is that the file time is correctly reflected when the handle that makes the change is closed."[1]. While Windows does update the time-stamps (which they also call "file times") of a file for various reasons, Microsoft does not specify how or why the time-stamps are updated, only that this is guaranteed to be done correctly. This study will show how to interpret time-stamps to uncover what may have happened at those times. A digital forensic investigator will then be able to decipher from the time-stamps of a file, what happened to that file at those times, using the results of this study. Consider the following eight time-stamps belonging to some file under investigation:

		a. $STANDARD_INFORMATION	b. $FILE_NAME
1	Creation	2019-07-02 21:33:35.3624443	2019-07-02 21:33:35.3624443
2	Write	2009-07-14 05:32:31.6745400	2019-07-02 21:33:35.3624443
3	Entry Modification	2019-07-02 21:33:35.3654445	2019-07-02 21:33:35.3624443
4	Access	2019-07-02 21:33:35.3624443	2019-07-02 21:33:35.3624443

Table 1.1: The time-stamps of a file under investigation.

While the names of these time-stamps might hint at what caused them, the reality of how they are calculated is vastly more complex (as described in section 7.7). Using the results of this study, we can determine that either of two things happened:

1. The original file was created or updated at 2009-07-14 05:32:31.6745400 (2a) and copied to the current file which started at 2019-07-02 21:33:35.3624443 (1a, 4a, 1b, 2b, 3b, 4b) and completed at 2019-07-02 21:33:35.3654445 (3a) this might have changed a file attribute (such as permissions) on completion.
2. A file created or updated at 2009-07-14 05:32:31.6745400 (2a) was copied at 2019-07-02 21:33:35.3654445 (3a), overwriting a file created or copied at 2019-07-02 21:33:35.3624443 (1a, 4a, 1b, 2b, 3b, 4b).

In this study we will show how to interpret the time-stamps of any file in NTFS, and how to automate this.

[1] `https://docs.microsoft.com/en-us/windows/desktop/sysinfo/file-times`

The effect of file operations on time-stamps

To reconstruct events from time-stamps, an array of time-stamps is treated as the result of a sequence of user interactions in the form of file operations. Examples of these operations are creating, copying and renaming files. Because different operations change time-stamps differently, the operations used may be inferred from the time-stamps. There are contradictory results in the literature describing the time-stamp effects of various file operations (see Chapter 3). By comparing the different studies with each other and our own findings we can validate the studies and ascertain what might have caused their differences and explain their measurements.

Some of these studies use their overview of file operations to uncover forgery. They suggest that if an array of time-stamps can not be the result of regular operations then it must be forged. To check whether an array of time-stamps may be the result of regular operations, the file operation overview should be complete. Otherwise valid time-stamps might be labeled as forged, because they match a regular operation not present in the incomplete operation overview. Until now, no complete overview of file operations exists, despite claims by the authors of those studies. For example, no study on time-stamp based event reconstruction mentions file tunneling, which can transfer a time-stamp from one file to another. It is known that ignoring file tunneling can lead to incorrect conclusions regarding time-stamps [1]. If incorrect conclusions are made by forensic investigators then court cases can suffer from false evidence. This stresses the need for an updated file operation overview, which this research will provide.

Time-stamp based event reconstruction

Until now, determining the operations that have happened for the time-stamps has been done by hand. This takes a lot of effort as for each file under investigation, the eight time-stamps will have to be compared to each other and the effects of every relevant file operation. Efforts to simplify this process can cause new problems (these problems are described in chapter 10). We propose a method and structure to automate determination of what file operations caused which time-stamps.

Time-stamps which should not be used for event reconstruction

Time-stamp based event reconstruction might be hindered or misled by time-stamps that have been forged. Time-stamp forgery can be done by adjusting the system clock to an incorrect time or using a time-stamp change tool. Arrays of time-stamps that can not be the result of regular file operations must be forged or degraded. To determine the degree to which forged time-stamps can be detected, we forge and then interpret time-stamps. Alternatively time-stamps might suffer from data degradation, causing them to appear as impossible times.

We also look into the possibility of irregular time-stamps carrying over between systems. If an irregular time-stamp may have originated on a different system, then it could have been forged by a third party irrelevant to the forensic investigation, or could have been caused by some other file system or operating system. No research has been found that considers the effect of transference of forged time-stamps between systems.

Real-life data

To verify our understanding of would-be regular and irregular time-stamps, real-life data collected by our supervisors is searched for time-stamps that fit certain regular or forging file operations. The amount of files found are then explained accordingly with the results of this research. Doing this we can ascertain the impact of bit-rot on time-stamps. We can also find out how often a quirk (See 7.4) occurs which we have not been able to uncover the cause of.

Contributions

The contributions of this research are:

1. An overview of file operations' effects on time-stamps, which can be compared to the time-stamps of a file to reconstruct events. This overview is the result of comparing and validating similar overviews of past research and uncovering different variations of the file operations mentioned in those studies.
2. An analysis of time-stamp forgery, time-stamp degradation and possible transference of forged or degraded time-stamps, which can be used to uncover if time-stamp forgery has taken place on a system.
3. A method and structure to automate determining which file operations happened for which time-stamps.

Chapter 2

Background

2.1 NTFS

The New Technology File System (NTFS) is a file system which allocates the contents of files to evenly-sized clusters. Cluster size is determined when formatting the disk. To quickly determine which clusters are allocated, NTFS uses a meta-data file called the bitmap (called '$BITMAP' on disk) which holds a 1 (allocated) or 0 (unallocated) for every cluster [2]. File names of NTFS meta-data files always start with a dollar sign ($), there is nothing preventing a user to create a file whose name starts with a dollar sign however. To restore the file system to an earlier state if necessary, NTFS keeps a log file called $LogFile, which contains before and after information of every meta-data change. $LogFile is implemented as a circular log, which means that it has a maximum size and if a new log entry is added when the log is full, then the oldest log entry is deleted[1]. NTFS contains various other meta-data files, most importantly the Master File Table (MFT, called '$MFT' on disk) which holds the meta-data of every file. When a file is deleted in NTFS, the file is marked as deleted in the MFT. Its meta-data and contents remain to eventually be overwritten. Thus, deleted files that have not yet been overwritten may be restored [3]. NTFS contains much meta-data that could possibly be used to date files, deleted or not. In this research we focus on the workings of the MFT and its contents, specifically the time-stamps.

This research focuses exclusively on NTFS. Because this is the file system used by the popular Windows series of desktop operating systems, the newest version of NTFS (NTFS 3.1) has been in use since Windows XP. Because NTFS is the file system used by Windows, it is an important file system from a digital forensics perspective as most disks under investigation will use NTFS. Different file systems have different types of time-stamps, different file operations and different behaviour by which time-stamps are changed as a result of those file operations. Because of this, the results of this research are not applicable to other file systems.

2.2 Master File Table

The MFT is a construct in NTFS that holds the meta-data of every file on the volume. It is structured as a series of evenly-sized entries. The default entry size is 1024 bytes (1 kB). The MFT contains only entries, thus its size should always be a multiple of the entry size. Each file and directory on the volume has at least one entry associated with it. The MFT is a file itself, as such the first entry contains the meta-data of the MFT.

Entries are addressed by its index, Microsoft calls this the "file number". Each entry also has a 2-byte sequence number which together with the index forms the "file reference address" [2]. The first 48 bytes of an entry form the entry header, which contains information about the entry and its contents. The entry header has a flag which specifies if it contains a file, directory, deleted file or deleted directory. If a file or directory is deleted then this flag is changed and the file is de-allocated. Deleted files can have their entry overwritten and can be overwritten on disk. The entry header of a file or directory will be followed by a list

[1] `https://github.com/jschicht/LogFileParser`

of attributes, which contain the meta-data of the file or directory. Attribute names in NTFS always start with a dollar sign.

2.2.1 $STANDARD_INFORMATION attribute

Each file or directory has one $STANDARD_INFORMATION (SI) attribute. This attribute contains four time-stamps, these are (in the order in which they are stored): creation, write, entry modification and access. These time-stamps might be called by different names: birth, modification, change and last access. None of these names accurately describe the complex way in which these time-stamps are changed by operations however (see section 7.7). An NTFS time-stamps is stored as a 64-bit value. This value represents the positive number of one-hundred nanoseconds since 1601-01-01 00:00:00 UTC[2], using the Gregorian calendar. The start of the year 1601 was picked because the Gregorian calendar operates on a 400 year cycle and 1601 is the start of the cycle during which NTFS was created[3]. Storing the time-stamps in a fixed format independent of local timezones or calendars, means that the time-stamps will not be affected if the volume is moved between timezones or calendars. Copies of the SI time-stamps are stored in parent directory attributes $INDEX_ROOT and/or $INDEX_ALLOCATION, these are updated for any interaction with the file[4]. The SI time-stamps are displayed in local timezone and calendar, rounded down to seconds when viewing the properties of a file in Windows Explorer, except for the entry modification time.

2.2.2 $FILE_NAME attribute

Each file or directory has at least one $FILE_NAME (FN) attribute. This attribute contains the name of the file or directory. FN also contains the same four types of time-stamps SI does, stored in the same order: creation, write, entry modification and access. While these time-stamps have the same names as the SI time-stamps, they are different and are changed differently by file operations. Multiple FN attributes can be used to store both long and short file names, if there are multiple attributes then their time-stamps are always equal to each other.

2.2.3 $DATA attribute

The $DATA attribute contains pointers to the clusters that hold the contents of the file. If the contents of a file fit in the entry then they are stored directly in the $DATA attribute instead. This is called a resident file, as the file will be resident in the MFT. If the $DATA attribute contains pointers to the allocated clusters then the file is non-resident.

2.2.4 $MFTMirr

Because of the importance of the MFT for the NTFS file system, the first entries of the MFT are backed up in the file "$MFTMirr" for recovery purposes. The $MFTMirr file contains copies of the first 4 entries from the MFT by default. If the cluster size is larger than 4 times the entry size, the $MFTMirr file's size is equal to cluster size. The $MFTMirr file always contains as much entries as can fit into it, so a 32kB cluster size and 1kB entry size means the $MFTMirr file holds copies of the first 32 entries of the MFT[5].

2.3 File operations

File operations are any interactions with files performed by any user or any software on the computer. This can be any reading or writing to the contents or meta-data of the file. There is no complete list of every possible file operation. Even if there was such a list, there would be no way to verify its completeness, and

[2] https://docs.microsoft.com/en-us/windows/desktop/sysinfo/file-times
[3] https://devblogs.microsoft.com/oldnewthing/20090306-00/?p=18913
[4] https://github.com/jschicht/SetMace
[5] https://flatcap.org/linux-ntfs/ntfs/files/mftmirr.html

new file operations could emerge at any time. Many file operations change the time-stamps of a file, storing times of operation in some time-stamps.

2.3.1 File operations considered in this research

While there are many file operations, in this research we only consider some. We want to discover what a user might have done to a file to cause its time-stamps. Thus any file operation a user can not knowingly perform is of no interest, neither is any file operation that does not alter or relate to time-stamps.

To reduce redundancy we do not consider file operations that consist of file operations we already consider. Application level software uses file operations provided by the operating system to implement their own file operations. For example: web browsers use sequences of file operations such as creating, renaming and updating to download files. Any web browser could use the same sequence of operations, change it in an update or use a different sequence. Furthermore, any other application software could use the same sequence of file operations for some other functionality. Considering these file operations that might have happened if some specific version of some specific software is used adds very little, as opposed to only considering the sequence of file operations that has happened.

We only consider a file operation if it satisfies the following conditions:

1. The file operation should be able to be performed by a user, either through the operating system or application software. This is because, while we are interested in how any file operation a user performs changes time-stamps. We are not interested in the implementation details of those file operations, which in the case of the closed-source operating system Windows we can only guess at.

2. The file operation should have some relation to time-stamps. Most file operations change some time-stamps and thus satisfy this condition. While "Delete" does not change any time-stamps, it is still related to time-stamps as described in the table below. If a file operation has no relation to time-stamps, we can not gain any evidence from time-stamps to whether or not this file operation happened, making the file operation irrelevant to this research.

3. In the group categorised by the previous conditions, file operations should not be a sequence of other file operations. This condition reduces redundancy by getting rid of application specific file operations that consist of file operations provided by the operating system.

We have compared all file operations found by related work and marked the file operations we will not consider in grey in table 2.1.

Name	Condition 1	Condition 2	Condition 3
Create	A user can create a file	Changes time-stamps	Not a sequence of other file operations
Copy (target)	A user can copy a file	Changes time-stamps	Not a sequence of other file operations
Copy (source)	A user can copy a file	No effect on time-stamps	
Update	A user can update a file	Changes time-stamps	Not a sequence of other file operations
Update using MS Word/Office	A user can update a file using MS Office	Changes time-stamps	Is a sequence of other file operations as shown in section 6.3
Move within volume	A user can move a file	Changes time-stamps	Not a sequence of other file operations
Move from another volume (target)	A user can move a file	Changes time-stamps differently from "Move within volume"	Not a sequence of other file operations
Move from another volume (source)	A user can move a file	No effect on time-stamps	
Overwriting copy	A user can copy a file, overwriting another file	Changes time-stamps differently from "Copy"	Not a sequence of other file operations
Overwriting move within volume	A user can move a file, overwriting another file	Changes time-stamps the same way as "Move within volume"	Is a "Move within volume", and has the same effect on time-stamps as shown in section 6.2
Overwriting move from another volume	A user can move a file, overwriting another file	Changes time-stamps differently from "Move" and "Move from another volume"	Not a sequence of other file operations
Rename	A user can rename a file	Changes time-stamps the same way as "Move within volume"	Has the same effect on time-stamps as "Move within volume" but is not a "Move within volume"
Attribute change	A user can change an attribute of a file	Changes time-stamps	Not a sequence of other file operations
Delete	A user can delete a file	No file operation can happen after deletion, meaning that deletion always happens after the latest time-stamp	Not a sequence of other file operations
Access	A user can access a file	May change time-stamps, as we show in section 7.1	Not a sequence of other file operations
Extract zip archive	A user can extract a zip archive	Changes time-stamps	Not a sequence of other file operations if Windows Explorer is used. Extracting zip archives with third party software may be a sequence of other file operations.
Upload	A user can upload a file	No effect on time-stamps	
Download	A user can download a file	Changes time-stamps	Is a sequence of operations depending on the browser used, for example "Create" and then "Update".
Move to recycle bin	A user can move a file to the recycle bin	No effect on time-stamps	
Move from recycle bin	A user can move a file from the recycle bin	Changes time-stamps	Is a "Move within volume", and has the same effect on time-stamps

Table 2.1: After the "Overwriting move within volume" where dir1/file.txt "overwrites" dir2/file.txt.

2.3.2 Time-stamp changes

This research is about the effects of file operations on time-stamps, both the four SI time-stamps and the four FN time-stamps. In this context, we treat file operations as an array of eight time-stamp changes that change an array of eight time-stamps. A time-stamp change is a collection of all characteristics of the

resulting time-stamp after the file operation, including a reference to the source time-stamp.

Chapter 3

Related Work

While NTFS has had time-stamps since its inception in 1993, research into how different operations change time-stamps did not start until 2007 when Chow et. al. [4] performed file operations and noted the change to some specific time-stamps.

Before 2007, the cause of time-stamps was judged by the names of those time-stamps and a short description, as Carrier [2] does in his analysis of NTFS. This is an over simplistic view however as it evokes a lot of questions. For example, "Creation Time: The time that the file was created.": Should the copying of a file be considered creation? Should the moving of a file from another volume be considered creation? Should extracting zip archives be considered creation? One could argue for any of these cases that it either is or is not the creation of a file. Time-stamps are still judged by their names in recent research (such as in Knutson and Carbone's analysis of time-stamps [5] from 2016), often citing Carrier [2]. We further describe the problems with judging time-stamps by their names in section 7.7.

According to Raghavan [6] there is widespread acknowledgement among digital forensics researchers that research efforts should be focused at digital forensic analysis which mainly aims to reconstruct events from time-stamps. He describes this in his study of the digital forensics state of the art. Raghavan also states that this area of research holds much promise and is the most likely to witness developments in "the years to come", (Raghavan wrote this in 2012). Buchholz and Spafford [7] state that time-stamps are crucial to event reconstruction. They suggest that time-stamps might be used to find out what file operations have taken place per file and put them into a time-line.

While there is some research into time-stamp based event reconstruction, researchers are often unaware of previous studies in this area causing them to unknowingly repeat work. Whenever results contradict earlier studies, the contradiction then remains unexplained. We can only take educated guesses as to what caused these contradictions.

3.1 File operations and their effects on time-stamps

Multiple researchers have performed file operations and read the MFT, to reconstruct events using time-stamps. They do this to get an impression of the before and after state of these file operations. These states are then compared to each other to find differences and explain these differences in the context of the operation. Cho [8], Bang and Lee [9], Ding and Zou [10], Sharma and Kaur [11] all have contradicting findings on operations' effects on time-stamps. These contradictions are not mentioned in their studies however.

3.1.1 Does Windows version affect time-stamps?

The related work is split on this question. Bang and Lee [9] and Sharma and Kaur [11] observe different effects of file operations on time-stamps depending on Windows version. Ding and Zou [10] and Cho [8] observe no difference depending on Windows version.

Bang and Lee [9] analyse file operations on 6 different versions of Windows ranging from Windows 2000 to Windows 7. Bang and Lee describe the resulting time-stamps in prior to ($<$) and equals relations ($=$)

to each other. They observe different effects on time-stamps depending on the Windows version used being later than Windows XP or not. They observe that the effect on time-stamps is the same for Windows XP and earlier versions. They observe that the effect on time-stamps is the same for Windows Vista and later versions. Bang and Lee also state that time-stamps can be changed differently for resident files and non-resident files.

Sharma and Kaur [11] analyse file operations on Windows XP, Windows Vista and Windows 7. They observe differences between Windows Vista and Windows 7 for the operations "Editing in file" and "Extraction of file from compressed zipped file". Despite citing Cho's research they do not copy his time-stamp change naming scheme, instead opting for more general terms to explain the time-stamps such as "Copy time" for copy operations. For more complex operations this terminology can become confusing, for instance they mention a "Creation time" time-stamp change for the operation "Extraction of file from compressed zipped file". This might mean the creation time-stamp of the archive is copied to the extracted file but alternatively it might refer to the time that the extracted file is created. They also ignore the processing time (Δ) differences that Cho uncovered. Why they make these choices is not explained.

3.1.2 Does file type affect time-stamps?

Ding and Zou [10] analyses 12 different file operations with a total of 16 variations. Ding and Zou group files into three categories: executable files, documents and directories. Ding and Zou claim that the effects of operations on time-stamps are dependant on file types as well as the applications used for opening/editing files. This claim is not supported by the other researchers, who only make a distinction between files and directories, or none at all. Cho [8] who has also considered different file types has not found file type to influence time-stamps.

Cho [8] identifies six different ways by which a time-stamp can change. Cho then describes the effects of 11 different file operations by their time-stamp changes, and he describes the use of a time-stamp change tool. He does this to compare regular file operations with time-stamps to detect forgery by time-stamp change tools. Cho [12] also analyses the results of his table to assess if operations can be detected as being the last performed. Cho makes no distinctions based on operating system, other than stating the known fact that Windows Vista and Windows 7 have last access updating disabled by default as opposed to Windows XP[1]. Cho does not reference the published research of Bang and Lee [9] or Ding and Zou [10].

3.1.3 Consensus on the effect of operations on time-stamps

While the different related works all have different results, there are some operations for which each of these studies observes the same effect on time-stamps. Every study suggests that when a file is created, all time-stamps are set to the time of creation. Furthermore each study agrees on the effects of one variation of each file operation, however they do not agree on the cause of the variations.

3.2 Detection of forgery by comparing time-stamps

When Cho's [8] sequence of rules based on operations effects on time-stamps is inconclusive then the $LogFile file is read to try to find the log record of an operation that forged the time-stamps. This log record might be easily recognised as the log record will show before and after time-stamps for the file and these can then be compared to the effects of file operations. This might detect forgery independent of current time-stamps, however it does depend on the forgery operation still being recorded in the $LogFile file. If the forgery happened a long time ago then likely it will not be recorded in the $LogFile file anymore.

Willassen [13] proves that system clocks that have been adjusted back can be detected by finding events that have a causality relation to each other and checking their time-stamps. If some event A must happen before an event B, then the time of event A must be earlier than that of event B. If this does not hold true, then the clock that delivered these time-stamps is assumed to be faulty. The possibility of a time-stamp change tool being used to alter the time-stamps of event A or B is not considered however.

[1] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc959914(v=technet.10)

Chapter 4

Methodology

To infer the effects of file operations on time-stamps, we perform file operations and read the change in time-stamps. The only way to definitely find all possible effects of file operations on time-stamps would be to analyse the source code of the closed-source Windows operating system. Furthermore, future versions of Windows might change how the file operations change time-stamps or introduce new file operations. As the source code is kept secret, we will instead reverse engineer the effects of file operations on time-stamps, by performing file operations and reading the change in time-stamps.

4.1 MFT or $LogFile

The effect on time-stamps may be measured by reading the contents of either the MFT or $LogFile. The MFT is where the time-stamps are stored. The time-stamps may be read before and after a file operation is performed, to infer the effect of the file operation on the time-stamps. An MFT entry contains the file name, this allows quickly searching the MFT for the file we are operating on.

$LogFile is an NTFS meta-data file which contains before (undo) and after (redo) information of all recent (limited by $LogFile size) changes in meta-data. After a file operation is performed, the changes it made to time-stamps can be read from $LogFile, to infer the effect of the file operation on the time-stamps. A problem with using the $LogFile to read time-stamp changes is that it contains records for every change in meta-data, while we are only interested in the change to time-stamps, which may be stored in multiple records. This means that we will have to search for the time-stamps. Searchability of $LogFile is poor however, as it contains only meta-data which is changed.

Reading either the MFT or $LogFile accomplishes the same goal, it provides the time-stamps before and after the file operation is performed. Alternatively both can be read, this is the method used by Cho [8], he has not noted any differences between MFT and $LogFile results. We focus only on the MFT as this is where the time-stamps are stored, making it the starting point of any event reconstruction. We do not also read $LogFile, as this redundancy has not been proven worthwhile and it would take a significant amount of time to learn the workings of $LogFile and then read it for every experiment.

4.2 Tooling

The perfect tool to find the effects of file operations would be a tool that can perform file operations just like a user would and read the time-stamps before and after a file operation. This tool would allow experiments to be run as fast as the computer can handle them. However, no such tool exists. To create such a tool, verify its correctness and then use it would take too much time, it would take far more time than performing file operations and reading the time-stamps without such a tool.

4.2.1 Reading the MFT

Time-stamps can be read directly from an MFT entry where they are stored as eight bytes each in little-endian order. These bytes hold the unsigned amount of 100 nanosecond intervals since 1601-01-01 00:00:00 UTC. While these bytes may be easily found and compared to each other, conversion is needed to read the actual date and time stored in the bytes.

An MFT entry does not contain the path of a file, it only stores the index of the MFT entry of the parent directory. Because of this, finding the MFT entry for a file becomes a problem when there are multiple files with the same name.

Because of the problems with reading raw bytes and finding the right file entry, we instead use a tool called "Mft2Csv"[1] which reads an MFT and outputs a human-readable text file containing 1 line per MFT entry. This line contains the index, sequence number, file name, file path and all eight time-stamps (and some more meta-data irrelevant to this research). Representing time-stamps as text not only makes them easier to read, it also allows searching for certain time-stamps.

4.3 Performing file operations and inferring their effects

To find and verify the effects of regular and irregular file operations, file operations are performed on NTFS 3.1 disks using Windows XP, Vista 7, 8 and 10. Multiple operating systems are tested because they might differ in time-stamp determination [9]. The following cycle is repeated:

1. Optionally, we fulfill a condition that might modify the effects of a file operation on time-stamps. This condition may relate to the operating system used, the settings of the operating system, the type of files that are operated on or the system clock. For example: the setting of the system clock to some future, far past or otherwise incorrect time, or changing the timezone, for testing the effects of an incorrect system clock.

2. The MFT is read, parsed and saved before the file operation, using Mtf2Csv.

3. The file operation is performed on multiple files. The current system time is noted after starting the operation. The files are of different types in different folders, with sizes varying from bytes to gigabytes.

4. The MFT is read and saved after the file operation, using Mtf2Csv.

5. The before and after MFT entries are compared to each other to look for changes to sequence number, location in the MFT and changes to time-stamps. The changes in time-stamps are examined to determine if they fit a type, or if they hold a new type of time-stamp change. The noted times of operation are also compared to the resulting MFT to detect any time-stamps that are earlier or later than expected.

This cycle is then repeated with both the same and other files until each time-stamp change can be definitively categorised. The cycle is then repeated again at a later time, to verify invariance of the operation under the chosen conditions. This is done at a later time to guarantee a different system state, to test if there are unknown factors which influence the effect of the operation on time-stamps.

All file operations that are mentioned in the aforementioned works related to this research will be tested. To discover irregular operations, we searched for possible problems and forgery like incorrect system times and time-stamp change tools.

4.4 Validating file operations

To verify our understanding of would-be regular and irregular time-stamps, the WildFrag database containing real-life NTFS meta-data is queried to find files that match specific time-stamp patterns. The amount of files with that specific pattern is then explained in relation to the results of this research. This will give

[1] `https://github.com/jschicht/Mft2Csv`

insight into how common some patterns are. This insight is invaluable when dealing with elusive seemingly random time-stamp changes such as time-stamp degradation. Common irregular time-stamp patterns that can not be explained mean that there are unknown file operations at play.

Chapter 5

The types of resulting time-stamps for file operations

As described in section 2.2, each file has a total of eight time-stamps. Depending on the file operations that have happened for a file, some of its time-stamps may be equal to each other. Four of these time-stamps are stored in the standard information attribute (SI) and the other four are stored in the file name attribute (FN). While each time-stamp has an implicit name within the attribute, respectively: creation, write, entry modification and access, these names do not accurately reflect what happened at those times (as described in section 7.7). For convenience, the time-stamps will be referred to by shortened names, these are:

- creation in SI:	SI.C
- write in SI:	SI.W
- entry modification in SI:	SI.E
- access in SI:	SI.A
- creation in FN:	FN.C
- write in FN:	FN.W
- entry modification in FN:	FN.E
- access in FN:	FN.A

Table 5.1: Abbreviations of time-stamp names

Each file operation results in eight time-stamps. Cho [8] identifies six different types of resulting time-stamps. We identify an additional resulting time-stamp which is caused by file tunneling. These seven are:

1. The start time of the file operation is written to the time-stamp (section 5.1).
2. The end time of the file operation is written to the time-stamp (section 5.2).
3. The time-stamp is left unchanged (section 5.3).
4. A time-stamp is copied from the SI attribute to the FN attribute (section 5.4).
5. A time-stamp from a different file entry is copied to the time-stamp (section 5.5).
6. A time-stamp stored in memory is transferred to the time-stamp because of file tunneling (section 5.6).
7. A time-stamp is set to an arbitrary value (section 5.7).

Every effect on a time-stamp of every known file operation can be described as these results.

Files stored in archive files or on other file systems may have time-stamps that have different precision than NTFS. Additionally, these time-stamps may be stored in local time without specifying the timezone.

To account for files transferred from different file systems or archive files, we introduce modifiers to resulting time-stamps for rounding (section 5.8) and time-zone difference (section 5.9).

5.1 The time at which the file operation starts: opStartTime

When a file operation is performed, the time at which the file operation is called may be written to time-stamps. We call this type of resulting time-stamp opStartTime(). A file operation that is performed has only one start time, meaning that if multiple time-stamps are set to opStartTime() by a file operation, then they will be equal after the file operation. So if for some file operation: $SI.C := opStartTime()$ and $SI.W := opStartTime()$ then $SI.C = SI.W$ (where := denotes assignment and = denotes equality). Table 5.2 shows time-stamps being set to opStartTime() for a create file operation performed at 2019-07-06 14:32:05.0577676.

Create at 2019-07-06 14:32:05.0577676			
Time-stamp	Before file operation	resulting time-stamp	After file operation
SI.C	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
SI.W	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
SI.E	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
SI.A	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
FN.C	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
FN.W	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
FN.E	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676
FN.A	N/A	***opStartTime()***	2019-07-06 14:32:05.0577676

Table 5.2: This example shows time-stamps being set to opStartTime() for a create file operation performed at 2019-07-06 14:32:05.0577676.

5.2 The time at which the file operation ends: opEndTime

When a file operation is completed, the time at which the file operation is completed may be written to time-stamps. We call this resulting time-stamp opEndTime(). Cho [8] describes this resulting time-stamp as opStartTime() + some processing time which may be zero. We have observed this resulting time-stamp to write the end time of the operation. However sometimes the start time is written instead of the end time. So the following is always true: $opStartTime() <= opEndTime()$.

5.3 An unchanged time-stamp

When a file operation is performed, some time-stamps may be left unchanged. Table 5.3 shows time-stamps left unchanged for an attribute change file operation performed at 2019-07-06 11:22:23.3571618.

Attribute change at 2019-07-06 11:22:23.3571618			
Time-stamp	Before file operation	resulting time-stamp	After file operation
SI.C	2009-07-14 05:32:32.0000000	***SI.C***	2009-07-14 05:32:32.0000000
SI.W	2019-06-16 15:23:17.6506413	***SI.W***	2019-06-16 15:23:17.6506413
SI.E	2019-06-16 15:23:17.9616808	opStartTime()	2019-07-06 11:22:23.3571618
SI.A	2019-06-16 15:23:17.6576422	***SI.A***	2019-06-16 15:23:17.6576422
FN.C	2019-06-14 12:55:30.3611443	***FN.C***	2019-06-14 12:55:30.3611443
FN.W	2009-07-14 05:32:31.6745400	***FN.W***	2009-07-14 05:32:31.6745400
FN.E	2019-06-14 12:55:30.3621444	***FN.E***	2019-06-14 12:55:30.3621444
FN.A	2019-06-14 12:55:30.3611443	***FN.A***	2019-06-14 12:55:30.3611443

Table 5.3: This example shows time-stamps left unchanged for an attribute change file operation performed at 2019-07-06 11:22:23.3571618.

5.4 A time-stamp transferred from the SI attribute

Some file operations such as "Rename" transfer time-stamps from SI to FN. Table 5.4 shows time-stamps being transferred from the SI attribute for a rename file operation performed at 2019-07-07 20:01:28.1003004.

Rename at 2019-07-07 20:01:28.1003004			
Time-stamp	Before file operation	resulting time-stamp	After file operation
SI.C	2019-07-04 18:44:49.3263725	SI.C	2019-07-04 18:44:49.3263725
SI.W	2019-07-04 18:44:49.6723923	SI.W	2019-07-04 18:44:49.6723923
SI.E	2019-07-04 18:44:49.6723923	opStartTime()	2019-07-07 20:01:28.1003004
SI.A	2019-07-04 18:44:49.3873760	SI.A	2019-07-04 18:44:49.3873760
FN.C	2019-07-04 18:44:40.0078395	***SI.C***	2019-07-04 18:44:49.3263725
FN.W	2019-06-23 13:18:51.8347653	***SI.W***	2019-07-04 18:44:49.6723923
FN.E	2019-07-04 18:44:49.3883761	***SI.E***	2019-07-04 18:44:49.6723923
FN.A	2019-07-04 18:44:49.3873760	***SI.A***	2019-07-04 18:44:49.3873760

Table 5.4: This example shows time-stamps being transferred from the SI attribute for a rename file operation performed at 2019-07-07 20:01:28.1003004.

5.5 A time-stamp transferred from another file entry

When a file operation is performed, some time-stamps might be transferred from another file entry. We will call this other file entry SRC. Table 5.5 shows SI.W transferred from another file entry for a copy file operation performed from 2019-07-07 09:01:47.8710875 to 2019-07-07 09:01:47.8880884.

Copy from 2019-07-07 09:01:47.8710875 to 2019-07-07 09:01:47.8880884			
Time-stamp	Before file operation (source file SRC)	resulting time-stamp	After file operation (target file)
SI.C	2019-06-23 13:17:28.7660140	opStartTime()	2019-07-07 09:01:47.8710875
SI.W	2019-06-23 13:18:53.5948659	***SRC.SI.W***	2019-06-23 13:18:53.5948659
SI.E	2019-06-23 13:18:53.5948659	opEndTime()	2019-07-07 09:01:47.8880884
SI.A	2019-06-29 14:37:11.3834131	opStartTime()	2019-07-07 09:01:47.8710875
FN.C	2019-06-23 13:18:51.8307650	opStartTime()	2019-07-07 09:01:47.8710875
FN.W	2019-06-23 13:18:51.8347653	opStartTime()	2019-07-07 09:01:47.8710875
FN.E	2019-06-23 13:18:53.3288507	opStartTime()	2019-07-07 09:01:47.8710875
FN.A	2019-06-23 13:18:53.3288507	opStartTime()	2019-07-07 09:01:47.8710875

Table 5.5: This example shows SI.W transferred from another file entry for a copy file operation performed from 2019-07-07 09:01:47.8710875 to 2019-07-07 09:01:47.8880884.

5.6 A time-stamp transferred by file tunneling

When a file operation is performed, file tunneling can transfer SI.C and FN.C time-stamps from memory. These time-stamps belonged to a file that existed with the same file name and path. We will call this memory location TNL. Why and when file tunneling happens is described further in section 7.2. Table 5.6 shows SI.C and FN.C transferred by file tunneling for a create file operation performed at 2019-07-24 20:50:31.3366254.

Create at 2019-07-24 20:50:31.3366254 with file tunneling			
Time-stamp	Before file operation (in memory TNL)	resulting time-stamp	After file operation (target file)
SI.C	2017-08-03 10:02:31.8012975	***TNL.SI.C***	2017-08-03 10:02:31.8012975
SI.W	N/A	opStartTime()	2019-07-24 20:50:31.3366254
SI.E	N/A	opStartTime()	2019-07-24 20:50:31.3366254
SI.A	N/A	opStartTime()	2019-07-24 20:50:31.3366254
FN.C	2017-08-03 10:02:31.8012975	***TNL.FN.C***	2017-08-03 10:02:31.8012975
FN.W	N/A	opStartTime()	2019-07-24 20:50:31.3366254
FN.E	N/A	opStartTime()	2019-07-24 20:50:31.3366254
FN.A	N/A	opStartTime()	2019-07-24 20:50:31.3366254

Table 5.6: This example shows SI.C and FN.C transferred by file tunneling for a create file operation performed at 2019-07-24 20:50:31.3366254.

5.7 A time-stamp set to an arbitrary value

A user can use a time-stamp change tool to set time-stamps to user specified values. We call this resulting time-stamp any(). This resulting time-stamp is due to forgery or bit-rot.

5.8 Result modifier: rounding

While NTFS stores time-stamps with a precision of 100 nanoseconds, other file systems or archives may store files with a lower precision. A file operation may transfer these time-stamps with lower precision to NTFS, which can be detected by the time precision of the resulting time-stamp. We introduce a function round(resulting time-stamp, precision, type) which specifies the rounding a resulting time-stamp has and whether it is rounded up, down or to the nearest amount. The precision is an amount of 100 nanoseconds. For example: the FAT file system stores creation time (FATs equivalent time-stamp to SI.C) with a precision

of 10 milliseconds. When a FAT file is moved to NTFS, the rounding for SI.C is round(SRC.SI.C, 100000, up). The effect of transferring files from FAT is described in further detail in section 7.3.

5.9 Result modifier: timezone difference

While NTFS stores time-stamps in UTC, other file systems or archives may store time-stamps in local time without specifying a timezone. FAT is an example of a file system that stores time-stamps without specifying a timezone. A file operation may transfer time-stamps from FAT to NTFS, which a Windows system will read in its own timezone and then store in UTC. If the timezones of the systems differ, then the time-stamp will be off by the difference. When a FAT file is moved to NTFS, the resulting time-stamp for SI.C is round(SRC.SI.C, 100000, up) + tzd. Where tzd is the timezone difference, this is an amount of hours that can be negative, positive or zero. Table 5.7 shows time-stamps of a FAT file operated on in New York (UTC-5) being moved to an NTFS volume on a Windows system in the Netherlands (UTC+1), causing the resulting time-stamps to be off by -6 hours.

Move from FAT volume from 2019-07-07 18:15:00.0481954 to 2019-07-07 18:15:00.0871976 where the timezone of the target system is UTC+1			
Time-stamp	Before file operation (source file SRC)	resulting time-stamp	After file operation (target file)
SI.C	2019-06-23 12:12:12.76 (set in timezone UTC-5)	***round(SRC.SI.C, 100000, up) + tzd***	2019-06-23 11:12:12.7600000
SI.W	2019-06-23 13:12:52 (set in timezone UTC-5)	***round(SRC.SI.W, 20000000, up) + tzd***	2019-06-23 12:12:52.0000000
SI.E	N/A	opEndTime()	2019-07-07 18:15:00.0871976
SI.A	2019-06-29 (set in timezone UTC-5)	opStartTime()	2019-07-07 18:15:00.0481954
FN.C	N/A	opStartTime()	2019-07-07 18:15:00.0481954
FN.W	N/A	opStartTime()	2019-07-07 18:15:00.0481954
FN.E	N/A	opStartTime()	2019-07-07 18:15:00.0481954
FN.A	N/A	opStartTime()	219-07-07 18:15:00.0481954

Table 5.7: This example shows time-stamps of a FAT file operated on in New York (UTC-5) being moved to an NTFS volume on a Windows system in the Netherlands (UTC+1), causing the resulting time-stamps to be off by -6 hours.

Chapter 6

Time-stamps as a Result of File Operations

To discover what file operations might have happened to a file to cause its time-stamps, we consider all file operations mentioned in related work. We have found different results for the file operations: Extract Zip Archive, Overwriting Move Within Volume and Update MS Word/Office. The effects of file operations that all related work agrees on and that we can reliably reproduce are not described in detail but are included in the resulting file operation overview in section 6.3.

A file operation may have different effects on time-stamps depending on some modifiers. This chapter is about the base effect of file operations on time-stamps, chapter 7 describes the different modifiers that file operations can have and their effects. For convenience, the overviews of base operations and modifiers are bundled together in appendix B.

6.1 Extracting zip archives using Windows Explorer

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2019-06-14 12:55:30.3611443	2019-06-14 12:55:30.3611443
W	2009-07-14 05:32:31.6745400	2019-06-14 12:55:30.3611443
E	2019-06-14 12:55:30.3621444	2019-06-14 12:55:30.3611443
A	2019-06-14 12:55:30.3611443	2019-06-14 12:55:30.3611443

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2009-07-14 05:32:32.0000000	2019-06-14 12:57:32.0431041
W	2009-07-14 05:32:32.0000000	2019-06-14 12:57:32.0431041
E	2019-06-14 12:57:32.5111309	2019-06-14 12:57:32.0431041
A	2009-07-14 05:32:32.0000000	2019-06-14 12:57:32.0431041

Figure 6.1: Change in time-stamps when extracting a zip file using Windows Explorer in Windows 7. The original time-stamps are show on the left, the time-stamps after zipping and extracting are shown on the right. The time-stamp transferred from the original file is marked in grey.

When extracting zip archive files with Windows Explorer, SI.C, SI.W and SI.A are set to the SI.W (modification time for other file systems) of the archived file rounded up to the nearest even number of seconds. This is because by default zip files store only SI.W with 2 second precision. This operation is illustrated by Figure 6.1, where the effect of a zip extraction at 2019-06-14 12:57:32.0431041 are shown. The SI.W in the zip file is not stored in UTC and thus can suffer from timezone differences, leading to time-stamps that are possibly off by any amount of hours. Because the time-stamps are rounded on 2 seconds exactly while NTFS has a precision of 100 nanoseconds, an extracted time-stamp can be detected by its 2 second precision. However, every other time-stamp will have a 1/20,000,000 chance of a false positive. The FN time-stamps are all set to opStartTime() while SI.E is set to opEndTime().

Rule	$SI# creation time	$SI# Last Modification time	$SI# Entry modification time	$SI# Last Access time
	$FN* creation time	$FN* last Modification time	$FN* Entry modification time	$FN* last Access time
9. Extraction of file from compressed zipped file	Extraction time	Creation time	Extraction time	Extraction time
	Extraction time	Extraction time	Extraction time	Extraction time

Figure 6.2: Time-stamp change observed by Sharma and Kaur on Windows 7. As taken from "Time Rules for NTFS File System for Digital Investigation" [11].

Rule	$SI# creation time	$SI# Last Modification time	$SI# Entry modification time	$SI# Last Access time
	$FN* creation time	$FN* last Modification time	$FN* Entry modification time	$FN* last Access time
9.Extraction of file from compressed zipped file	No change	No change	Extraction time	No change
	Extraction time	Extraction time	Extraction time	Extraction time

Figure 6.3: Time-stamp change observed by Sharma and Kaur on Windows Vista and Windows XP. As taken from "Time Rules for NTFS File System for Digital Investigation" [11].

While Sharma and Kaur [11] observed a difference between extracting zip files on Windows 7 (see Figure 6.2) and extracting zip files on XP or Vista (see Figure 6.3), we have not observed any difference. We used Windows Explorer for extracting zip files on Windows Vista, Windows XP and Windows 7, with equal results. As Sharma and Kaur have not mentioned the application used for zip extracting it is possible that they might have used a third party program on Windows 7, as we have only observed extraction using Windows Explorer to have a unique effect. When the zip is extracted with WinRAR (not a part of Windows) it is treated as a regular copy operation, it still has the same problems inherent to zip archives however (2 second precision and timezones).

6.2 Overwriting move within volume

When a file is moved within the same volume to overwrite another file, it does not actually overwrite the target file or file entry. Instead the target file is marked as deleted and the source file is then moved within the volume, without file tunneling. After an "Overwriting move within volume" the target file and file entry are only de-allocated, while any other overwriting operation would overwrite the target file and file entry.

Tables 6.1 and 6.2 serve as an example, they show the before and after state of file dir1/file.txt at MFT index 45 "overwriting" dir2/file.txt at MFT index 63.

Performing "overwriting move within volume"			
MFT index	Deleted	Path	File operation performed
45	Not deleted	dir1/file.txt	Move within volume
63	Not deleted	dir2/file.txt	Delete

Table 6.1: Before the "Overwriting move within volume" where dir1/file.txt "overwrites" dir2/file.txt.

After "overwriting move within volume"			
MFT index	Deleted	Path	File operation performed
45	Not deleted	dir2/file.txt	Moved within volume
63	Marked as deleted	dir2/file.txt	Deleted

Table 6.2: After the "Overwriting move within volume" where dir1/file.txt "overwrites" dir2/file.txt.

6.3 Update MS Word/Office

We have found that when updating a file with Microsoft Word, that file is actually deleted and a new file is made. This is because the "'safe save method" is used, this method of updating files does not alter the original file but creates a new one and writes the updated data to the new file. The creation time-stamps are unchanged, the other time-stamps are set to near operation time and always SI.A = FN.A < SI.W = FN.W < FN.E < SI.E.

This is what could have led to these time-stamps:

1. source file has its name changed to "W̃RL0001.tmp"
2. new file is created with file tunneling from source file
3. update
4. attribute change
5. rename

This would lead to a new copy of the file with the same name where always SI.A = FN.A < SI.W = FN.W < FN.E < SI.E. Because updating a file using MS Office is made up of different operations, it will be excluded from the file operation overview. updating a file using MS Office does not have a unique effect on time-stamps and thus can not be reliably detected. Other software could be using the same updating method.

6.4 Overview of time-stamp changes by operations

The following overview (table 6.3) contains the resulting time-stamps for the unmodified file operations. These are the effects on time-stamps of files (not directories) if last access update is disabled, files are not transferred from other file systems, no file tunneling takes place and the SI.E quirk (section 7.4) does not happen. Some file operations and time-stamps can be affected by these modifiers. The different modifiers are described in chapter 7. For convenience, the overviews of the base file operations and all modifiers are grouped together in appendix B.

Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Create	opStartTime() opStartTime()	opStartTime() opStartTime()	opStartTime() opStartTime()	opStartTime() opStartTime()
Copy	opStartTime() opStartTime()	SRC.SI.W opStartTime()	opEndTime() opStartTime()	opStartTime() opStartTime()
Update	SI.C FN.C	opEndTime() FN.W	opStartTime() FN.E	SI.A FN.A
Move Within volume	SI.C SI.C	SI.W SI.W	opEndTime() SI.E	SI.A SI.A
Move From another volume	SRC.SI.C opStartTime()	SRC.SI.W opStartTime()	opEndTime() opStartTime()	opStartTime() opStartTime()
Overwriting copy	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Overwriting move From another volume	SRC.SI.C FN.C	SRC.SI.W FN.W	opStartTime() FN.E	SI.A FN.A
Rename	SI.C SI.C	SI.W SI.W	opStartTime() SI.E	SI.A SI.A
Attribute change	SI.C FN.C	SI.W FN.W	opStartTime() FN.E	SI.A FN.A
Delete	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Access	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Extract zip file	round(SRC.SI.W, 20000000, up) + tzd opStartTime()	round(SRC.SI.W, 20000000, up) + tzd opStartTime()	opEndTime() opStartTime()	round(SRC.SI.W, 20000000, up) + tzd opStartTime()

Table 6.3: The unmodified effects of file operations on time-stamps.

Chapter 7

Modifiers to the Effects of File Operations on Time-stamps

To be able to find the operations that have happened to cause some time-stamps. We need to consider every possible effect the operation might have on time-stamps.

The effect of a file operation on time-stamps may be influenced by modifiers. A modifier overwrites some resulting time-stamps for some operations. The known modifiers are:

1. Last access updating (section 7.1) can be disabled or enabled in Windows by editing a registry key, its default values differ per Windows version. This modifier only affects (SI.A). Last access updating is a widely known modifier [8][9][10]. However, no research analyses the effect of last access updating on time-stamps in detail for every file operation.

2. File tunneling (section 7.2) saves information of a file that is renamed, relocated or deleted. Among this information are the SI.C and FN.C time-stamps which may be written to a new file which has the same path and name as the old one. File tunneling can be replicated reliably on all versions of Windows [1]. However, no research analyses the effect of file tunneling on time-stamps in detail for every file operation.

3. Transferring files from other file systems (section 7.3) can cause timezone differences and can be detected by rounded time-stamps. This is because other file systems use different time-stamp precision and might not store or enforce a timezone. We consider all file systems supported by Windows at time of writing, these are: FAT and exFAT. Other file systems can only be read by third party software, considering all these file systems and third party software is outside of the scope of this study.

4. We have found a quirk (section 7.4) where sometimes copying or moving from another volume leaves SI.E unchanged. Unfortunately we have not been able to unearth the cause for the quirk.

The modifiers are not mutually exclusive, except perhaps for the quirk which has proven difficult to reproduce. This means that if a file is copied from FAT while last access updating is enabled and file tunneling happens, then the resulting time-stamps are affected by all these three modifiers.

There are contradictory results in past research relating to whether the effect of file operations might be modified by the type of file operated on (section 7.5) or the version of Windows used (section 7.6). We investigate these claims and compare the results of past research with our own.

7.1 Last access updating

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2019-05-21 12:40:50.1938658	2019-05-21 12:40:50.1938658
W	2009-07-14 05:32:32.0000000	2009-07-14 05:32:32.0000000
E	2019-06-10 17:51:21.2234489	2019-06-08 22:32:59.5945453
A	2019-06-01 15:38:06.8173095	2019-06-01 15:38:06.8173095

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2019-05-21 12:40:50.1938658	2019-05-21 12:40:50.1938658
W	2009-07-14 05:32:32.0000000	2009-07-14 05:32:32.0000000
E	2019-06-10 17:51:21.2234489	2019-06-08 22:32:59.5945453
A	2019-06-15 18:40:16.4897393	2019-06-01 15:38:06.8173095

Figure 7.1: Change in time-stamps when accessing a file by reading its properties, with NtfsDisableLastAccessUpdate set to false. The original time-stamps are show on the left, the time-stamps after access are shown on the right. The changed time-stamps are marked in grey.

Whenever a file is accessed, which is a broad term used to describe any kind of reading information from the file, the value of the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\NtfsDisableLastAccessUpdate registry key will determine if SI.A is updated to time of operation [4]. In Windows Vista, Windows 7, Windows 8 and early versions of Windows 10 this is set to 1 (true) by default, which means that the access time will remain unchanged when accessing files. Figure 7.1 shows the change in time-stamps for a file accessed at 2019-06-15 18:40:16.4897393, with the registry key set to false.

In Windows 10 with update "Redstone 4" and later, the system determines the value of the registry key during booting by default. If the system volume (where Windows is installed) is 128GB or smaller in size, then it is set to 0x80000002 during system booting which enables last access updates. If the system volume is larger than 128GB it is set to 0x80000003, which disables last access updates. A user can turn off the system management by setting it to 0x80000000 (last access update enabled) or 0x80000001 (last access update disabled) [1].

If the registry key is set to anything other than 1, 0x80000002 or 0x80000003 then the registry key has been changed by a user. If the registry key is changed by the user then this might be a weak attempt at forgery. However if it is set to any of the default values, the registry key might have been changed and reset. The system disk might also have changed over time, leading to changes in the registry key. Thus we can not assume that the value has not been changed any number of times. This is why we can not consider either the access operation with or without update to be indicative of forgery attempts.

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2019-06-14 19:20:34.9108399	2019-06-14 19:20:34.9108399
W	2009-07-14 05:32:31.6745400	2019-06-14 19:20:34.9108399
E	2019-06-14 19:20:34.9133402	2019-06-14 19:20:34.9108399
A	2019-06-15 14:31:49.4457852	2019-06-14 19:20:34.9108399

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2019-06-14 19:20:34.9108399	2019-06-14 19:20:34.9108399
W	2009-07-14 05:32:31.6745400	2019-06-14 19:20:34.9108399
E	2019-06-15 18:40:07.7711322	2019-06-14 19:20:34.9108399
A	2019-06-15 18:40:04.2021790	2019-06-14 19:20:34.9108399

Figure 7.2: Delayed change in time-stamps when accessing a file by opening it, with NtfsDisableLastAccessUpdate set to false. The original time-stamps are show on the left, the time-stamps after access are shown on the right. The changed time-stamps are marked in grey.

The last access update can be delayed up to an hour [2], when the access time-stamp is then updated it is set to the original operation time. A delayed update is treated as an attribute change operation, meaning that SI.E will hold the time after delay. Figure 7.2 shows the change in time-stamps for a file accessed at 2019-06-15 18:40:04.2021790, with the registry key set to false and the access update delayed till 2019-06-15 18:40:07.7711322. The last access update might in some cases not happen at all, even if the registry key is set to false.

When an operation is performed the file might be accessed before or after the operation. A file will be accessed before: moving within volume, changing an attribute or deleting. A file will be accessed after: updating or extracting from zip. Some operations do not have an associated separate access operation, but have different resulting time-stamps if last access update is enabled. For the operations copy, update, move,

[1] https://dfir.ru/2018/12/08/the-last-access-updates-are-almost-back/

[2] https://docs.microsoft.com/en-us/windows/desktop/sysinfo/file-times

overwriting copy and overwriting move, the access time is set to the latest time of operation if last access update is enabled.

Table 7.1 shows the resulting time-stamps and file operations affected by the modifier. Resulting time-stamps marked "..." are the same as in the base operation overview 6.3.

Last access updating enabled				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	...	...	opEndTime()
	...	...	...	...
Update	...	...	...	opEndTime()
	...	...	...	...
Move	...	...	...	opEndTime()
From another volume	...	...	...	...
Overwriting copy	...	...	...	opStartTime()
	...	...	...	...
Overwriting move	...	...	...	opStartTime()
From another volume	...	...	...	...
Access	...	...	...	opStartTime()
	...	...	...	...

Table 7.1: The effects of having last access update enabled on time-stamps for file operations. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

7.2 File tunneling

When a file has its short name changed, is copied, is moved or is deleted: the full path, short name (prior to name change) and SI.C are kept in memory for some time (15 seconds by default) [3]. If within this time another file is created or modified to have the same short name and path. Then this file has both its SI.C and FN.C set to the SI.C in memory.

File tunneling is done to support the "safe save method"[3]. This is a method of updating a file, which does not alter the original file. In this method, a new file is created and the updated data is written to this new file, and only then is the old file is deleted. Renaming is done to either the new or old file so they can exist at the same time and the new file gets the name of the old file. Because of file tunneling, updating a file with the safe save method keeps the creation time of the original file.

File tunneling can happen for the following operations: "create", "copy", non-overwriting "move in the same volume" and "file name change". For non-overwriting "move in the same volume" and "file name change", the FN.C is not tunneled however. It has been proven that ignoring file tunneling can lead to incorrect conclusions regarding time-stamps [1]. File tunneling does not happen for directories.

The maximum amount of files tunneled at the same time can be changed by creating the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\MaximumTunnelEntries, setting it to 0 will disable file tunneling. The time frame in which tunneling can happen (15 seconds by default) can be changed by creating the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\MaximumTunnelEntryAgeInSeconds.

Tunneling might be proven by finding the file that was tunneled from. The only other regular operations that can result in multiple files having the exact same creation time are extract zip file and the FAT related operations (due to rounding inaccuracies). So if multiple files have the same creation time-stamp not rounded on 10 milliseconds, tunneling must have taken place.

[3] https://df-stream.com/2012/02/file-system-tunneling-in-windows/

Table 7.2 shows the resulting time-stamps and file operations affected by the modifier.

File tunneling				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Create	TNL.SI.C	...	...	...
	TNL.FN.C	...	...	...
Copy	TNL.SI.C	...	...	...
	TNL.FN.C	...	...	...
Move	TNL.SI.C	...	...	...
In the same volume	...	...	...	...
Rename	TNL.SI.C	...	...	...
	...	...	...	...

Table 7.2: The effects of file tunneling on time-stamps for file operations. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

7.3 Transferring files from other file systems

FAT

FAT (File Allocation Table) is a file system which has the following time-stamps: Creation (SI.C equivalent), Modification (SI.W equivalent), Access (SI.A equivalent). FAT has varying precision for the creation, modification and access time-stamps. The creation time is stored with 10 millisecond precision. The modification time is stored with 2 second precision. The access time is stored with 1 day precision, it can not be transferred to NTFS however. A creation or modification time-stamp can thus be detected as originating from a FAT file system. However because any operation could lead to a time-stamp with this exact precision, there exists the chance of a false positive. Each file will have a 1/100,000 false positive chance for creation time and 1/20,000,000 for modification time.

FAT time-stamps do not store any timezone information, nor is any timezone enforced. Because of this timezone differences might arise when FAT files are read in a different timezone or transferred to a different timezone.

Table 7.3 shows the resulting time-stamps and file operations for a file transfer from FAT.

Transfer from FAT				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	round(SRC.SI.W, 20000000, up) + tzd	...	...
	...	...	...	...
Move	round(SRC.SI.C, 100000, up) + tzd	round(SRC.SI.W, 20000000, up) + tzd	...	...
From another volume	...	...	...	...
Overwriting copy	...	round(SRC.SI.W, 20000000, up) + tzd	...	...
	...	...	...	...
Overwriting move	round(SRC.SI.C, 100000, up) + tzd	round(SRC.SI.W, 20000000, up) + tzd	...	...
From another volume	...	...	...	...

Table 7.3: The effects of transferring from FAT on time-stamps for file operations. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

exFat

exFat (Extended File Allocation Table) is a variant of FAT file systems, which solves some of the problems inherent to FAT. exFAT has the same time-stamps as FAT: Create (SI.C), Modification (SI.W) and Access (SI.A). Unlike FAT, exFAT stores time-stamps in UTC, avoiding unknown timezone differences. Furthermore, exFAT has a modification time (SI.W) precision of 10 milliseconds, as opposed to FAT's 2 second precision. The access time is stored with 2 second precision in exFAT, it can not be transferred to NTFS however.

Table 7.4 shows the resulting time-stamps and file operations for a file transfer from exFAT.

Transfer from exFAT				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	round(SRC.SI.W, 100000, up)	...	...
	...	...	...	...
Move	round(SRC.SI.C, 100000, up)	round(SRC.SI.W, 100000, up)	...	...
From another volume	...	...	...	...
Overwriting copy	...	round(SRC.SI.W, 100000, up)	...	...
	...	...	...	...
Overwriting move	round(SRC.SI.C, 100000, up)	round(SRC.SI.W, 100000, up)	...	...
From another volume	...	...	...	...

Table 7.4: The effects of transferring from exFAT on time-stamps for file operations. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

7.4 Unexpected behaviour of SI.E

There is a quirk in Windows where SI.E does not get updated for a period of time. This behaviour was encountered on a Windows 10 system for the operations: copy and move from another volume. The origin of the quirk is unknown at this point, the system in question did not have the quirk before or after. All other time-stamps were updated as expected.

Table 7.5 shows the resulting time-stamps and file operations affected by the quirk.

Quirk				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	...	SRC.SI.E	...
	...	...	...	...
Move	...	...	SRC.SI.E	...
From another volume	...	...	...	...

Table 7.5: The effects of the quirk on time-stamps for file operations. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

7.5 Operations per file type

It has been suggested that file type influences the time-stamp change pattern.

7.5.1 Executable files

Ding and Zou [10] observe that executable files have SI.E changed differently from other files. They note that the operations "Copy" and "Inter-volume move" update the SI.E time to time of operation for other files, but not for executables. However when we copy and move executables and other files across volumes the SI.E time is updated for either type of file. Ding and Zou might have stumbled upon the same quirk that we did and thought it to be related to the file type. This likely means that the quirk happens more often for some systems as it has proven elusive to us.

Ding and Zou also note a difference for the operation "Touch" where .exe files have their SI.E set to time of operation, while other files have their SI.E unchanged. The operation "Touch" is not described, but we have not found running or looking at the properties of .exe files to have an effect on the SI.E time-stamp. It is possible that touching the file caused an attribute to change which sets SI.E to operation time. This might have been caused by a short delay in the last access update.

7.5.2 Directories

For directories we have found the same time-stamp effects for operations as Ding and Zou [10] have. File operations have the same effect on directories, except for the update operation. Furthermore directories can not overwrite each other, they can only combine. When two directories are combined, files are copied or moved from one directory to the other. This is an update operation for the directory. Directories are not extracted from zip archives either. Instead, the directories are newly created and updated when a zip archive is extracted.

Table 7.6 shows the resulting time-stamps for file operations preformed on directories.

Is directory				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Update	... ...	... ...	... ...	opEndTime() ...
Overwriting copy	N/A N/A	N/A N/A	N/A N/A	N/A N/A
Overwriting move From another volume	N/A N/A	N/A N/A	N/A N/A	N/A N/A
Extract zip file	N/A N/A	N/A N/A	N/A N/A	N/A N/A

Table 7.6: The effects of file operations on the time-stamps of directories. Resulting time-stamps marked "..." are the same as in the base file operation overview 6.3.

7.6 Operations per operating system

It has been suggested that time-stamp calculation is done differently per operating system. Bang and Lee [9] have noted differences between Windows XP and later versions of Windows. We have confirmed the invariance of their results. Most of these differences are related to the NtfsDisableLastAccessUpdate registry key however, as it is set to 0 by default in Windows XP as opposed to Windows Vista and Windows 7.

Sharma and Kaur [11] have noted differences between Windows XP, Windows Vista and Windows 7. They observed differences between time-stamp updating in Windows Vista and Windows 7 for "Editing in file" and "Extraction of file from compressed zipped file", however we have not seen any difference when performing these operations on Windows Vista and Windows 7. They suggest that "Editing in file" does not update the SI.W time in Windows Vista, we have not been able to reproduce this however.

Both Bang and Lee and Sharma and Kaur have observed that when copying a file or moving a file between volumes on Windows XP, SI.E is set to t^E_{src}. We have confirmed this effect on Windows XP. This can be explained as the quirk happening more often or always on Windows XP.

To conclude: we have not observed any difference in the effects of file operations across Windows versions newer than Windows XP. Furthermore, the differences between Windows XP and later Windows versions might be the result of the default setting of NtfsDisableLastAccessUpdate and a higher frequency of the quirk in Windows XP.

7.7 Accuracy of time-stamp names

Now that all file operations and modifiers can be considered, we can ascertain the accuracy of the names and descriptions given to the time-stamps by past research. We will not consider the FN time-stamps as past research already recognises that the names and descriptions of the time-stamps are not applicable to the FN time-stamps as FN time-stamps are calculated differently [2][5]. Carrier [2] defines the time-stamps as follows:

- Creation Time (SI.C): "The time that the file was created."
- Modified Time (SI.W): "The time that the content of the $DATA or $INDEX attributes was last modified."
- MFT modified Time (SI.E): "The time that the metadata of the file was last modified."
- Accessed Time (SI.A): "The time that the content of the file was last accessed."

Creation Time (SI.C)

To verify the following: "This is the time the file was created.", we must first define "created". Oxford English Dictionary defines "creation" as "The action or process of bringing something into existence.". In this case either the file or the file entry must be brought into existence at the time of SI.C.

SI.C is not the time that a file entry was brought into existence, as moving files from other volumes leaves SI.C unchanged.

SI.C is not the time that a file was brought into existence either. When a file is copied then SI.C of the new file is updated as it should because a new file is brought into existence. However, when a file is extracted from a zip archive then SI.C of the extracted file becomes SRC.SI.W, which is unrelated to creation. Furthermore any file tunneling causes SI.C to become TNL.SI.C which is related to the creation of an arguably different file.

Modification Time (SI.W)

The description: "The time that the content of the $DATA or $INDEX attributes was last modified." is true for NTFS, only updating the contents of the file will update SI.W. However, if a file is transferred from another file system then it would not have had a $DATA or $INDEX attribute to modify.

MFT modified Time (SI.E)

The description: "The time that the metadata of the file was last modified.", is false whenever the quirk (section 7.4) happens.

Accessed Time (SI.A)

SI.A is described as: "The time that the content of the file was last accessed.". This is only true if the NtfsDisableLastAccess registry key is false however, and even then last access updating might not always happen.

Conclusion

There is some truth to the names and descriptions of the time-stamps. However, the names and descriptions provide a simplistic view which allows room for interpretation and error. In real forensic investigation these simplified names and descriptions should be avoided as they fall short of the detail gained by considering individual file operations.

Chapter 8

Time-stamp Forgery and Degraded Time-stamps

Time-stamp based event reconstruction might be hindered or misled by time-stamps that have been forged or affected by bit-rot. Time-stamps that are forged or degraded may match no file operations at all. Alternatively, these time-stamps will match file operations that may not have happened with the wrong times.

Time-stamp forgery can be done by either changing time-stamps directly to some user-specified value (section 8.1) or by adjusting the system clock to an incorrect time before performing operations.

Time-stamp changing is a powerful anti-forensics tool that if used correctly can fool forensic investigators into assuming operations picked by the forger happened at times defined by the forger. We will describe the three ways to change time-stamps, and their accompanying tools.

Adjusting the system clock is also a powerful tool to hinder forensics. As long as a clock is only adjusted forward, the time-stamps will still accurately show what file operations took place only at the wrong times.

Bit-rot can also affect time-stamps, hindering digital forensic investigation or causing investigators to draw wrong conclusion. Knowing the possible impact of forgery and bit-rot on the time-stamps of a file will allow us to determine detectability. If forgery and bit-rot can be detected, then wrong conclusions can be avoided.

Possible transference of forged or degraded time-stamps from a different volume is also considered. If a time-stamp is forged or degraded then it will be interesting to know where the forging or degradation happened. Especially in the case of forgery where a suspect could get blamed for forgery someone else might have committed.

8.1 Changing time-stamps to user-specified values

Changing the time-stamps to user-specified times is the only way to set time-stamps to times unrelated to any time of operation. The resulting time-stamps might help to possibly detect the time-stamp forgery if they are set to some impossible or suspicious value. However, because they are not related to any operation time, time-stamps forged by a time-stamp change tool can not provide any other information.

8.1.1 The SetFileTime system call

The Windows system call "SetFileTime" can set the SI.C, SI.W and SI.A time-stamps to specified values[1]. This system call will set SI.E to opStartTime().

Most time-stamp change tools use this system call to set time-stamps and are thus limited to changing SI.C, SI.W and SI.A. They also have another limitation not imposed by the system call, all these time-stamp change tools use second precision. This means that resulting time-stamps will be rounded on seconds.

[1] `https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime`

Rounding on seconds for SI.C and SI.W also happens for transferring from FAT, exFAT or zip archive. However the only modified or unmodified file operation that rounds SI.A on seconds is extract zip archive, which rounds SI.A on an even number of seconds. Thus time-stamps rounded on an uneven number of seconds should be seen as suspicious, especially SI.A. The reason that these tools all use second precision is perhaps that the more precise time-stamp information is hidden when looking at files in Windows Explorer.

Table 8.1 shows the resulting time-stamps for time-stamp change tools using SetFileTime, and the potential of SetFileTime.

SetFileTime				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Time-stamp change tool	round(any(), 10000000, nearest) FN.C	round(any(), 10000000, nearest) FN.W	opStartTime() FN.E	round(any(), 10000000, nearest) FN.A
Potential	any() FN.C	any() FN.W	opStartTime() FN.E	any() FN.A

Table 8.1: The effect of using SetFileTime to forge time-stamps.

8.1.2 the undocumented NtSetInformationFile system call

The officially undocumented Windows system call "NtSetInformationFile" can set all SI time-stamps to user-specified values[2]. The time-stamp change tool Timestomp is the only known tool that uses this system call[3]. Timestomp suffers from the same limitation as previous time-stamp change tools however, it can only set time-stamps with one second precision. These time-stamps are even more suspicious as there is no file operation that rounds SI.E.

Table 8.2 shows the resulting time-stamps for time-stamp change tools using NtSetInformationFile, and the potential of NtSetInformationFile.

NtSetInformationFile				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Timestomp	round(any(), 10000000, nearest) FN.C	round(any(), 10000000, nearest) FN.W	round(any(), 10000000, nearest) FN.E	round(any(), 10000000, nearest) FN.A
Potential	any() FN.C	any() FN.W	any() FN.E	any() FN.A

Table 8.2: The effect of using NtSetInformationFile to forge time-stamps.

NtSetInformationFile can not change the FN time-stamps. However, all time-stamps can be set to user specified values by the following steps[4]:

1. Use NtSetInformationFile to set all SI time-stamps to desired FN values.
2. Move the file within the volume or change the file name, to copy the SI time-stamps to the FN time-stamps.

[2] https://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FNT%20Objects%2FFile%2FNtSetInformationFile.html

[3] https://www.forensicswiki.org/wiki/Timestomp

[4] https://www.forensicswiki.org/wiki/Timestomp

3. Use NtSetInformationFile to set all SI time-stamps to desired values.

Detecting this 3-step time-stamp forgery might require more information than only time-stamps, as all the time-stamps can be forged to appear legitimate (apart from possible second precision when using any tools available at time of writing).

8.1.3 Editing the raw time-stamps data in the MFT to change time-stamps

While Windows protects NTFS meta-data files, using a different operating system the MFT of a disk can be modified directly. Alternatively, an experimental tool inspired by Timestomp called "SetMace"[5] can change all time-stamps (SI and FN) to user specified values, with maximum NTFS precision (100 nanoseconds). This tool works by writing directly to physical disk, thus circumventing operation system and file system security. The copies of the SI time-stamps stored in parent directory attributes $INDEX_ROOT and/or $INDEX_ALLOCATION, are also changed by SetMACE. Windows has protections in place against writing to physical disk and as such SetMACE has to circumvent these protections, at time of writing it uses a kernel mode driver for this.

Table 8.3 shows the resulting time-stamps for editing the raw time-stamp data in the MFT.

Editing the raw data of the MFT				
Operation	Resulting time-stamps			
	SI.C	SI.W	SI.E	SI.A
	FN.C	FN.W	FN.E	FN.A
Editing raw time-stamps	any()	any()	any()	any()
	any()	any()	any()	any()

Table 8.3: The effect of editing the raw time-stamps.

Because the raw MFT data is edited circumventing any logging by Windows or NTFS, no record of it will exist anywhere. However, if $LogFile still contains some record of the old time-stamps then it is clear that an unlogged impossible time-stamp change happened to lead to the current time-stamps.

8.1.4 Detection

Because most time-stamp change tools work with 1 second precision, time-stamps that are rounded on seconds should be considered suspicious. Especially if the same time-stamps are not rounded on 2 seconds as well, meaning they can not originate from a zip file or FAT modification time.

Time-stamps that do not match regular non-forging file operations could be the result of forgery. These time-stamps could be matched against the possible forging operations. However, editing raw time-stamps will match any time-stamps as it is always a possibility.

If there are still time-stamps before the time-stamp change present then Cho's [8] method can be used which uses the $LogFile to detect forgery by time-stamp change tool. This method requires the time-stamp change tool changes to still be logged in $LogFile, which is bounded and overwrites old records with new ones.

If time-stamps forged by a time-stamp change tool match a series of non-forging file operations, have time-stamps that are neither impossible nor rounded on seconds, with the evidence of forgery overwritten in $LogFile then these forged time-stamps can not be detected.

8.2 Changing the system clock

When a regular file operation is performed, some time-stamps will be set to opStartTime() or opEndTime(). These are the time values the clock has when the operation is performed, if the clock is set to the wrong time then these time-stamps will be set to the wrong time as well.

[5] `https://github.com/jschicht/SetMace`

Detection

If an incorrect clock is ideal (never moving back) then the time-stamps will seem formed by regular file operations. If the clock is moved back then files can have time-stamps where a time of operation is lower than an unchanged time which might be detected as time-stamps not formed by regular file operations. Furthermore, if $LogFile holds time-stamps from before and after the moving back of the clock, this can easily be detected by the jump back in time. If a clock is moved back or forward to impossible times this can easily be detected as time-stamp forgery, because the time-stamps will have impossible values.

8.3 Time-stamp degradation

	$STANDARD_INFORMATION	$FILE_NAME ($FN)
C	2015-10-23 22:32:42.4215809	2015-10-23 22:32:42.4215809
W	2015-10-23 22:32:42.4215809	7857-04-01 04:03:30.2301953
E	2015-10-23 22:32:42.4215809	2015-10-23 22:32:42.4215809
A	2015-10-23 22:32:42.4215809	2015-10-23 22:32:42.4215809

Figure 8.1: Time-stamps of an $MFT file created at 2015-10-23 22:32:42.4215809 where the FN.W has degraded. The degraded time-stamp is marked in grey.

Bit-rot on disk might cause time-stamps to degrade. If only the time-stamp is affected then the structure of the MFT entry is still correct and the signature of the entry will not indicate any problems. We have encountered degraded time-stamps. Figure 8.1 shows the time-stamps of an $MFT file created at 2015-10-23 22:32:42.4215809 where the FN.W has degraded. While the time-stamps might not seem related to one another, 2015-10-23 22:32:42.4215809 is actually written on disk as 0x0175 28BB E20D D101, while 7857-04-01 04:03:30.2301953 is written on disk as 0x0175 28BB E20D 661B (NTFS stores time-stamps in little endian byte order). Only the last 2 bytes of the FN.W time-stamp have degraded in this case. Interestingly, the MFT mirror $MFTMirr has the exact same time-stamps, it seems that one of these files mirrored the degraded time-stamp from the other.

Degraded time-stamps can possibly be detected by looking at the $LogFile if the time-stamps have changed recently. If $LogFile still contains some record of the undegraded time-stamp then it is clear that an unlogged impossible time-stamp change happened to lead to the current time-stamps.

The real danger of degraded time-stamps are the least significant bytes. While degradation in the most significant bytes is easily detected due to the time-stamp having some impossible future or far past value, degradation in the least significant byes is more subtle. As mentioned earlier, NTFS stores time-stamps in 8 bytes containing 100 nano second intervals since 1601 01-01. The least significant 4 bytes contain half of the time-stamp but only store $2\hat{}(4*8) - 1 = 4294967295$ 100 nanosecond intervals. This amounts to a mere 429.4967295 seconds, which means that when time-stamp degradation happens, half the time it will change time-stamps by less than 430 seconds. This small change in a time-stamp will likely not be detected and the time-stamps might be matched with the wrong file operation.

8.4 Origins of irregular time-stamps

As time-stamps are associated with files, the time-stamps can be carried over between volumes or systems. This can happen for any operation that transfers files across volumes and copies time-stamps from other file entries (SRC).

- SI.C can be transferred by moving a file from another volume (overwriting or non-overwriting) or extracting a zip file.
- SI.W can be transferred by copying, moving from another volume (both operations overwriting or non-overwriting) or extracting a zip file.
- SI.E can be transferred by copying or moving from another volume if the quirk occurs.

- SI.A can be transferred by extracting a zip file.
- The FN time-stamps can all be copied over from SI and thus time-stamps transferred from other systems could wind up in the FN attribute by renaming the file or moving it within volume.

The most difficult time-stamps to transfer are thus SI.A and FN.A. If SI.A or FN.A does not have 2 second precision then it is not extracted from a zip file which means that according to all known file operations it could not be transferred from another system. This means that if SI.A or FN.A has some impossible value that is not rounded on 2 seconds then that value must have originated on the current volume.

Using the time-stamps of the $MFT file to detect forgery or degradation

When an NTFS volume is formatted, the $MFT file is created and in accordance with the "Create" file operation all time-stamps are set to time of operation. However, the $MFT file is special in that it will never have its time-stamps updated, no matter how many times it is modified. This means that time-stamps that do not originate on some other volume should always be later than those associated with the $MFT file.

8.5 Conclusion

The MFT is limited in the information it contains about the time-stamps, this also limits its ability to detect forgery or degradation. $LogFile is perhaps a better tool to detect forgery or degradation as it contains past time-stamps, however, $LogFile contains only recent changes.

The time-stamps in the MFT are far more useful to detect if forgery or degradation have most likely not taken place. If the time-stamps match file operations, are not impossible times and are not rounded on seconds (unless the matched file operations round the time-stamps on seconds) then there is no indication that forgery or degradation have taken place.

Chapter 9

Investigating Time-stamps in the Wildfrag Database

Real-life data collected by our supervisors is searched for time-stamps that match with certain regular or forging file operations. Doing this we can ascertain the frequency of certain file operations and time-stamp degradation. Knowing how frequent time-stamps degrade provides insight into the impact of time-stamp degradation. By extension, this provides insight into the reliability of time-stamps themselves. We can also find out how often the quirk occurs, of which we have not been able to uncover the cause.

9.1 WildFrag database

To aid digital forensic research, Vincent van der Meer has composed a database with file meta-data of 220 student volunteer laptops running the Windows operating system. The Wildfrag database composed by Vincent van der Meer contains file meta-data of 100,014,916 files across 729 NTFS volumes on 220 student volunteer laptops. As the volunteer students are presumably innocent of time-stamp forgery, their time-stamps can be used to gain insight into the commonality of future and far past time-stamps on the average volume.

While WildFrag was intended for research into file fragmentation, the SI time-stamps are included in the database. The start time of the disk reading run is also stored in the database which will allow comparing with SI time-stamps to find time-stamps that were in the future at read time. Because of the tooling used for reading the laptops (The Sleuth Kit[1] and Fiwalk[2]), the time-stamps have been converted to UNIX Epoch times. This means that the time-stamps are rounded on seconds and have a range of 1901-12-13 20:45:52 to 2038-01-19 03:14:07, the time-stamps outside this range are nulls in the database. It should be noted that at time of writing there was a mix-up with signed and unsigned integers causing time-stamps between 1901-12-13 20:45:52 and 1970 to appear as time-stamps after 2038-01-19 03:14:07. The nulls and mix-up will cause little problems however, as both future and far past all have the same causes and are equally irregular.

By investigating the SI time-stamps from the WildFrag database, we gain insight into how commonly specific time-stamp patterns can be found. The amount of files with some specific pattern is explained accordingly with the results of this research.

9.2 Future and far past time-stamps

The database contains 39,886 NTFS files with time-stamps that are either in the future counting from the start time of the disk analysing run or from 1970-01-01 00:00:00 or before. This is approximately 0.04% of all NTFS files in the database. These files are spread out across 339 volumes out of 729. Only 18 of these

[1] `https://www.sleuthkit.org/`

[2] `https://www.forensicswiki.org/wiki/Fiwalk`

339 volumes have a size of less than 100 GB, while 367 of the 390 remaining volumes have a size of less than 100GB. From this we can conclude that presence of future or far past time-stamps on a disk is mainly tied to the amount of files and not to the user. Thus because having future and far past time-stamps on a volume is common and usually not tied to the user, their mere presence on a disk does not indicate any foul play.

9.2.1 Volume 59

Volume 59 contains the largest share of files with future or far past time-stamps: 13,099. This might be the result of an erroneous clock, as 13,067 of these file only have time-stamps that are within a day of the start time of the disk analysing run. Because volume 59 has both Windows 10 and Linux installed, these might have separate clocks. Interestingly 11,065 of these 13,067 files only have SI.E in the future or far past. It is likely that the Linux operating system is not writing the NTFS time-stamps correctly. Testing file operations on Ubuntu 18.04.2 shows that the time-stamps written to disk by the Linux NTFS driver NTFS-3G are not accurate, we have observed the time-stamps being off by a seemingly random amount of time between two and three hours from the actual time of operation. Furthermore some file operations have different effects on time-stamps when using Linux. Because this volume is an outlier not representative of the average volume and its time-stamps have now been examined, this volume will not be considered in the further analysis of the WildFrag database.

9.2.2 Amount of future or far past time-stamps per type

We would expect SI.C and SI.W to be in the future or far past more often than SI.E and SI.A, because of how many file operations can transfer these between systems. However, the database has approximately 7000 of each time-stamp, which seems to indicate that internal (not transferred from another volume) time-stamp degradation happens far more often than transference of future or far past time-stamps. As time-stamp degradation is equally likely to happen for any time-stamp, this would explain the little difference in amount of future or far past time-stamps per type.

Time-stamp	Files with only this time-stamp in the future or far past	Files with multiple time-stamps in future or far past	Total files	Total volumes
SI.C	6321	484	6805	313
SI.W	6608	1660	8268	311
SI.E	5885	466	6351	308
SI.A	6252	1365	7617	310

Table 9.1: The amount of future and far past time-stamps per type of time-stamp.

Table 9.1 shows the amount of future and far past time-stamps per time-stamp type. It also shows the difference between the amount of files that have only one future or far past time-stamp, and those that have multiple. There are many more files with only one future or far past time-stamp than files with multiple, this is likely due to time-stamp degradation happening more often for only one time-stamp than for two.

These future and far past time-stamps might have been caused by bit-rot, however they might not have been. As these are all time-stamps that have been subject to change, they may be the result of wrong calculation by Windows or the CPU.

9.3 The quirk: SI.C > SI.E

All regular file operations result in an SI.C earlier than SI.E, except for the copy operation with the quirk that does not update the SI.E time-stamp. Alternatively, some these time-stamps might be caused by timezone differences when extracting zip files or transferring from FAT volumes. The WildFrag database

contains 3,173,018 files across volumes with SI.C later than SI.E (not counting future or far past time-stamps or volume 59). This is likely mostly due to the quirk in Windows where SI.E does not get updated when copying. 2,692,002 of these files match "Copy with quirk" as the last operation. These files all have equal SI.C and SI.A. times and earlier SI.M and SI.W times. When we also consider the possibility of a copy with quirk followed by an access with last access update enabled (SI.C $<$ SI.A), this matches 443,488 files out of the remaining 481,016.

We have encountered this quirk while reading the MFT of a Windows 10 system, but from the three Windows 7 systems in the database it seems that this quirk is also present on Windows 7. On these three systems, approximately 4% of files have time-stamps that match copying with quirk.

Chapter 10

Automating Comparison of Time-stamps with Operations

To make easier use of the file operation overviews, past studies have derived rules from them by hand. These rules have conditions based on time-stamp patterns, if the condition is fulfilled then the time-stamps must be the result of some operation. Comparing an array of time-stamps to every operation in the overview should give the same result as the rule does. The downside with using these rules is that when operations need to be added or removed from the operation overview (for example, due to a Windows update), that every rule will need to be re-evaluated. Furthermore, finding rules is a complex process which can easily lead to human error, due to the large amount of possible time-stamp changes that need to be considered.

To solve both these problems, we propose a structure and method for comparing operations to time-stamp arrays. This method and structure is then implemented in a tool which can match time-stamps to file operations.

10.1 Examples of rules derived from file operation overviews

Ding and Zou [10] derive rules from the table of file operations that they made. These derived rules provide detailed information on what operations have happened. Unfortunately, likely due to human error, not all derived rules match their table of file operations. They state these derived rules for all file types: If SI.C $>$ SI.W, then the contents and the summary property of the file have not been modified in the current volume; If SI.C $<$ SI.W, then SI.W is the last modification time of content or summary property of the file in the current volume.

These rules do not match the "Inter-volume replace" operation in their table. Ding and Zou describe Inter-volume replace as keeping SI.C unchanged and setting SI.W to SRC.SI.W. Because SI.C may be either later or earlier than SRC.SI.W, SI.C might be either later or earlier than SI.W following an Inter-volume replace. This means that any Inter-volume replace can match either derived rule, while the derived rules have mutually exclusive results. This means that the derived rules can not be correct based on the file operation overview of Ding and Zou.

Cho [8] uses his table to devise a set of 7 rules to be used in sequence to determine if a set of time-stamps have been forged. Because these rules are meant to be used in sequence they do not only rely on the file overview being correct and complete, but also rely on the rules earlier in the sequence. Now, updates to Windows and developments in time-stamp forgery have made some of these rules outdated however. "Rule 2: if at least one of the FN field has no U, the case is not timestamp forgery" has become outdated because of developments in time-stamp forgery where the FN time-stamps can be forged. Rules 3 to 7 relate to some subset of operations that did not pass the condition of rule 2. Now that rule 2 has become outdated, this subset of operations and its associated 5 rules have also become outdated.

These examples show that formulating rules from an operation overview can cause problems as adding or changing one operation can cause multiple rules to no longer be correct. They also show that deriving

these rules is a complex process which can easily lead to human error, due to the large amount of resulting time-stamps that need to be compared to each other.

10.2 Structure

To automate comparison of operations with time-stamps, the operations must be structured in a way that allows easy comparison. This and past research have provided overviews where file operations are structured as an array of time-stamp results which allows comparing of file operations with the related time-stamps. To automate this, we must define and structure the differences between the time-stamp result types.

10.2.1 Time-stamp result types

Time-stamp result types (such as opStartTime()) need to be characterised to be distinguished from each other so they can be correctly compared and matched. This characterisation is done by assigning the following attributes to each:

- integer "lateness": If the time associated with this time-stamp result type should be later or earlier than other time-stamp results. For instance: opStartTime() (lateness 1) is always later than a SRC or unchanged time-stamp (lateness 0) [8]. Note that lateness is ignored if the time-stamps are equal. If the lateness can not be compared then it should be set to -1, this is the case for time-stamp results with a possible timezone difference.

- integer "possibleEquivalence": Other time-stamp results with the same value can have the same time. The time-stamp results that have rounding can all have the same times because they might be different times rounded to the same time. opStartTime() and opEndTime() might also be equivalent.

- boolean "alwaysSelfEquivalent": If other time-stamp results with this type must have equal times. For example, this is true for opStartTime() as there can only ever be one operation start time for an operation.

- positive integer "rounding": The rounding the time-stamp should have in 100 nanosecond intervals. A value of 1 means rounded on 100 nanoseconds (no rounding).

- integer "equivalenceWithSameType": Other time-stamp results with the same value have the same time, if the related time-stamps are of the same type (creation, modification, entry modification or access). This is true only for a time-stamp transferred from SI to FN and an unchanged time-stamp as this unchanged time-stamp will have been copied to FN. As it is a copy, the time-stamps must be equal.

Now the time-stamp result types can be compared with time-stamps to find the latest operation that has taken place. However we also want to know when the operation has taken place and what operations happened before it. To achieve this, attributes need to be added which will let us determine which time-stamp results hold part of the time of operation. We will also need to add attributes which will let us mark already matched time-stamps so that those can be ignored when considering what operations happened before it. Additionally, to find multiple operations we need to be able to restore time-stamps to their state before the operation, if possible. These are as follows:

- boolean "operationResult": If the time-stamp is related to the time of the operation. Only opStartTime() and opEndTime() have this value set to true. These time-stamps will determine at what time the operation happened.

- boolean "fromFile": Time-stamp changes with this parameter set to the same value, have time-stamps originating from the same file. Different values mean the time-stamps stem from different files. For example: this is 1 for a time-stamp from SRC but 0 for an unchanged time-stamp so when an operation like "Overwriting move from another volume" happens, the remaining unmatched time-stamps need to be divided among 2 files.

- enum "copyStyle": This can have one of three values: NOT_COPIED if this resulting time-stamp is not copied from another type of time-stamp.

  COPIED_FROM_TIMESTAMP if the time-stamp is copied to this time-stamp from some specific time-stamp. For instance: time-stamps being copied from SRC.SI.W when extracting zip files.

  COPIED_FROM_SAME_TYPE if the time-stamp is copied from the other attribute to the same time-stamp type. This is true for a time-stamp copied from SI to the same time-stamp type in FN.

- optional integer "copiedFrom": If copyStyle is set to COPIED_FROM_TIMESTAMP this denotes the time-stamp that was copied from. This is a 0-7 value that follows the same time-stamp order as the rest of this research: SI.C SI.W SI.E SI.A FN.C FN.W FN.E FN.A.

10.3 Method

10.3.1 Comparing an array of time-stamps to an operation

When comparing an array of time-stamps with an array of time-stamp changes (an operation), each time-stamp change pair must be compared with their corresponding time-stamps. If we compare the following array of time-stamps:

- SI.C: 2019-03-17 20:39:40.4969433
- SI.W: 2009-07-14 05:32:31.6745400
- SI.E: 2019-03-17 20:39:40.4969433
- SI.A: 2019-03-17 20:39:40.4969433
- FN.C: 2019-03-17 20:39:40.4969433
- FN.W: 2019-03-17 20:39:40.4969433
- FN.E: 2019-03-17 20:39:40.4969433
- FN.A: 2019-03-17 20:39:40.4969433

to the file operation copy, whose time-stamp effects are shown in table 10.1.

Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	opStartTime() opStartTime()	SRC.SI.W opStartTime()	opEndTime() opStartTime()	opStartTime() opStartTime()

Table 10.1: The effects on time-stamps of the file operation "Copy".

It must first be verified that SI.C could have been set to opStartTime(), for this it must be compared with every other time-stamp.

1. Comparing it with SI.W which should have been set to SRC.SI.W, shows that SI.C is later than SI.W, just like opStartTime() should be later than SRC.SI.W.
2. Comparing it with SI.E shows that SI.C and SI.E are equal, opStartTime() and opEndTime() have possible equivalence, so this matches.
3. Comparing SI.C with all other time-stamps shows that they are all equal and all match opStartTime().

This confirms that SI.C could have been set to opStartTime() in the context of this operation and these time-stamps. Now the other time-stamps are compared with each other, if all time-stamps are compared and no problems have been found then this time-stamp array matches the operation.

If any of the FN time-stamps is not 2019-03-17 20:39:40.4969433 then it will not match SI.C, as opStartTime() is always self equivalent. This will cause the comparison to stop with the conclusion that this time-stamp array can not be the result of the copy operation.

In pseudo-code this is as follows:

```
amount of time-stamps = 8
For (ii = 0; ii < amount of time-stamps - 1; ii++):
  For (jj = ii + 1; jj < amount of time-stamps; jj++):
    If time-stamps ii and jj do not match the operation's resulting time-stamps ii and jj
        Return this operation does not match the time-stamps
Return this operation matches the time-stamps
```

10.3.2 Finding a possible sequence of operations for an array of time-stamps

To find the operation that happened before a matched operation, we first mark the operation result time-stamps (for which the operationResult attribute is true) as matched. Because we do not know what time-stamp came before a matched one, it can be any time-stamp. Looking back at our previous example: the copy operation marked every time-stamp as matched except SI.W. At that time (2009-07-14 05:32:31.6745400) either a create or an update operation happened, as these are the only operations that set SI.W to an operation result time. The other time-stamps are no longer considered as they have already been matched.

Splitting the array of time-stamps

Some operations such as the overwriting move and copy, have time-stamps from two different files. When any of these operations is matched the time-stamps need to be split up to two arrays as they were before the operation.

Copied time-stamps

Copied time-stamps are more complicated as they can have an implicit time of operation. For example the following time-stamps (previous example with name changed):

- SI.C: 2019-03-17 20:39:40.4969433
- SI.W: 2009-07-14 05:32:31.6745400
- SI.E: 2019-03-18 13:21:24.7343231 matched as “attribute change”
- SI.A: 2019-03-17 20:39:40.4969433
- FN.C: 2019-03-17 20:39:40.4969433
- FN.W: 2009-07-14 05:32:31.6745400
- FN.E: 2019-03-17 20:39:40.4969433
- FN.A: 2019-03-17 20:39:40.4969433

The SI.E time-stamp has been matched as an “attribute change” operation which is possible. However, when looking at the FN time-stamps we know that a rename or move within volume must have happened, as FN.W is not equal to FN.E and no other file operations can cause that. We know that the operation (rename or move within volume) must have happened after 2019-03-17 20:39:40.4969433, as that time-stamp was moved by the operation. We also know that it happened before 2019-03-18 13:21:24.7343231, as this time-stamp was changed after the operation (otherwise it would have been copied to FN.E). Thus for copied time-stamps with no operation time left to match, we can match these to the following range: later than

the latest time-stamp that was copied to, but earlier than the earliest time-stamp value that has not been copied. If an operation with copied time-stamps is matched then those time-stamps should be marked and copied back. For this example this would look as follows:

- SI.C: 2019-03-17 20:39:40.4969433 copied back
- SI.W: 2009-07-14 05:32:31.6745400 copied back
- SI.E: 2019-03-17 20:39:40.4969433 copied back
- SI.A: 2019-03-17 20:39:40.4969433 copied back
- FN.C: 2019-03-17 20:39:40.4969433 marked
- FN.W: 2009-07-14 05:32:31.6745400 marked
- FN.E: 2019-03-17 20:39:40.4969433 marked
- FN.A: 2019-03-17 20:39:40.4969433 marked

10.3.3 Finding all possible sequences of operations for an array of time-stamps

As the previous example showed, multiple sequences can be matched to an array of time-stamps. We should strive to find all possible sequences. The pseudo-code to find all possible sequences for an array of time-stamps is as follows:

```
amount of time-stamps = 8

Algorithm matchTimestampsToSequences:
  While not all time-stamps in all sequences are matched:
    For each sequence:
      matchTimestampsToSequence

Algorithm matchTimestampsToSequence:
  List matchedOperations
  While not all time-stamps are matched:
    For each operation:
      If matchTimestampsToOperation is not null then:
        If operation marks or copies different time-stamps than matchedOperations then:
          fork sequence with time and operation
        else:
          store time and operation in matchedOperations
    Mark matched time-stamps
    (split and fork) or copy back if necessary

Algorithm matchTimestampsToOperation:
  For (ii = 0; ii < amount of time-stamps - 1; ii++):
    For (jj = ii + 1; jj < amount of time-stamps; jj++):
      If (time-stamps ii and jj do not match the operation's time-stamp results ii and jj)
      and neither ii nor jj is marked then:
        Return this operation does not match the time-stamps (null)
  Return matched time-stamps
```

10.4 Time-stamp Analyser

The Time-stamp Analyser reads an MFT and compares the meta-data of each entry to the list of regular operations. If no regular file operation can be matched with the time-stamp then it is assumed to be the result of forgery or decay and those are matched instead. For each time-stamp a list of possible operations is given. This can be done because unless a time-stamp change tool is used, every time-stamp is formed by a time of operation. The tool is open source and the list of file operations is made to be easily changeable if new regular or irregular operations are discovered.

10.4.1 Use

The Time-stamp Analyser is written in Java 8, this means JRE 8 or newer is required to run it. The Time-stamp Analyser takes an MFT file as input and produces a text file as output. The Time-stamp Analyser is a command line tool with the following parameters:

1. input file
2. output file
3. (optional) MFT entry size in bytes: default is 1024.
4. (optional) filter: either "deleted" to analyse only time-stamps of deleted files, "irregular" to find files with irregular time-stamps or "all" for everything (default).
5. (optional) list of indexes or file names to be analysed separated by "|". By default every file in the MFT is analysed.

An MFT file can be extracted from disk using RawCopy [1].

10.4.2 Output

The time-stamp analyser outputs a text file which describes what possible sequences of operations have happened. A sequence is structured as follows:

(MFT entry index) (file name) ((time: possible operations at that time) (If the operations took place on a different volume)) (<-) ((earlier time: possible operations at that time) (If the operations took place on a different volume))

If an operation combines time-stamps from two files (such as overwriting copy), the sequence for the file from a different entry is marked by not having an MFT entry index. The latest file operation is mentioned first because the possible file operations converge to the time-stamps.

10.4.3 Use case

In the introduction of this study the following time-stamps are shown:

		a. $STANDARD_INFORMATION	b. $FILE_NAME
1	Creation	2019-07-02 21:33:35.3624443	2019-07-02 21:33:35.3624443
2	Write	2009-07-14 05:32:31.6745400	2019-07-02 21:33:35.3624443
3	Entry Modification	2019-07-02 21:33:35.3654445	2019-07-02 21:33:35.3624443
4	Access	2019-07-02 21:33:35.3624443	2019-07-02 21:33:35.3624443

Table 10.2: The time-stamps of penguins.jpg.

These time-stamps belong to the file penguins.jpg, in the introduction we claim to interpret from these time-stamps that either of two sequences of file operations have happened:

[1] `https://github.com/jschicht/RawCopy`

1. The original file was created or updated at 2009-07-14 05:32:31.6745400 (2a) and copied to the current file which started at 2019-07-02 21:33:35.3624443 (1a, 4a, 1b, 2b, 3b, 4b) and completed at 2019-07-02 21:33:35.3654445 (3a) this might have changed a file attribute (such as permissions) on completion.

2. A file created or updated at 2009-07-14 05:32:31.6745400 (2a) was copied at 2019-07-02 21:33:35.3654445 (3a), overwriting a file created or copied at 2019-07-02 21:33:35.3624443 (1a, 4a, 1b, 2b, 3b, 4b).

We run the time-stamp analyser with the following parameters:

```
C:\research\MFT_with_penguins C:\research\out.txt 1024 all penguins.jpg
```

This results in the file out.txt containing the following text, which is equal in meaning to the two sequences of file operations written above:

113 .\Penguins.jpg (From 2019-JULY-2 21:33:35.3624443 UTC to 2019-JULY-2 21:33:35.3654445 UTC: Copy) <- (At 2009-JULY-14 5:32:31.6745400 UTC: Create | Create with file tunneling | Update | Update with last access update enabled) possibly on other volume

113 .\Penguins.jpg (At 2019-JULY-2 21:33:35.3654445 UTC: Attribute change) <- (At 2019-JULY-2 21:33:35.3624443 UTC: Copy | Copy with last access update enabled | Copy with quirk) <- (At 2009-JULY-14 5:32:31.6745400 UTC: Create | Create with file tunneling | Update | Update with last access update enabled) possibly on other volume

.\Penguins.jpg (At 2019-JULY-2 21:33:35.3654445 UTC: Overwriting copy) <- (At 2009-JULY-14 5:32:31.6745400 UTC: Create | Create with file tunneling | Update | Update with last access update enabled) possibly on other volume

113 .\Penguins.jpg (At 2019-JULY-2 21:33:35.3654445 UTC: Overwriting copy) <- (At 2019-JULY-2 21:33:35.3624443 UTC: Create)

113 .\Penguins.jpg (At 2019-JULY-2 21:33:35.3654445 UTC: Overwriting copy) <- (At 2019-JULY-2 21:33:35.3624443 UTC: Copy from FAT volume | Copy from FAT volume with last access update enabled | Copy from exFAT volume | Copy from exFAT volume with last access update enabled)

113 .\Penguins.jpg (At 2019-JULY-2 21:33:35.3654445 UTC: Overwriting copy) <- (At 2019-JULY-2 21:33:35.3624443 UTC: Copy | Copy with last access update enabled | Copy with quirk)

These are all the possible sequences of file operations for these time-stamps, which may seem like a lot of possibilities. All the possible sequences have two things in common however, the file was copied at 2019-JULY-2 21:33:35 and created or updated at 2009-JULY-14 5:32:31.6745400.

Because of time constraints on this study, the choice was made to not implement a visualisation of the time-stamp analyser output and settle for a basic text output instead. A visualised output would have looked as Figure 10.1.

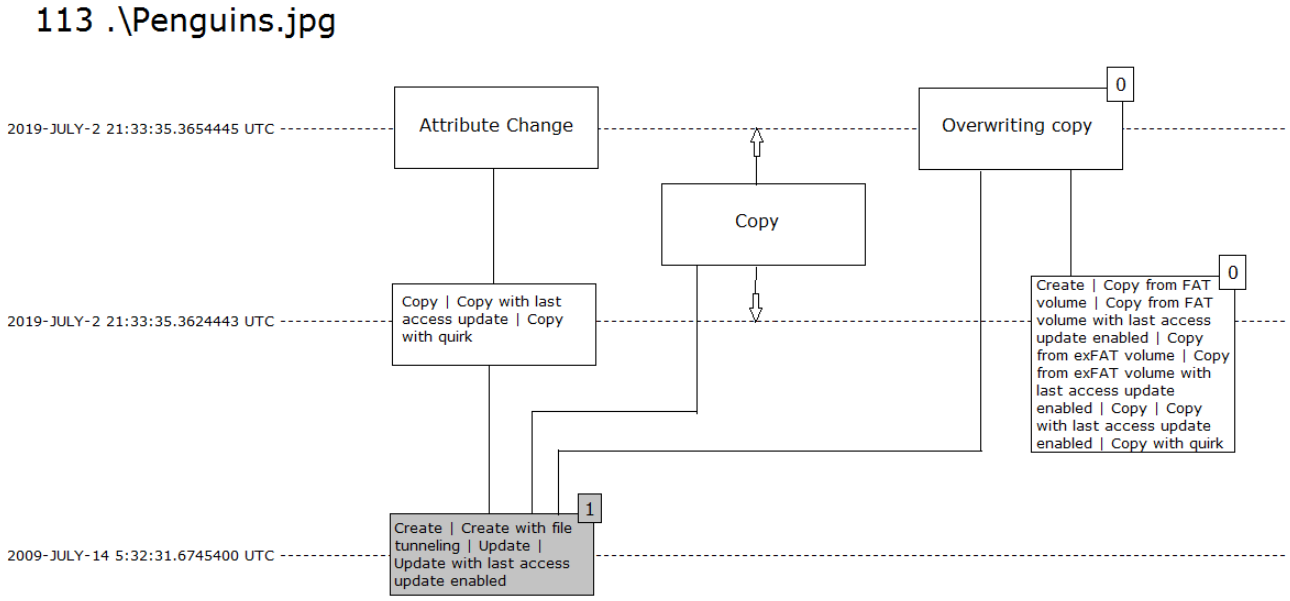

Figure 10.1: Visualisation of possible file operations for the time-stamps of penguins.jpg

The time-stamps are positioned along side the vertical axis. File operations are either aligned with time-stamps (precise time), between two time-stamps (range) or between two time-stamps with arrows pointing to the time-stamps (duration). The file operations are connected, each path is a possible sequence of file operations. The file operations marked in grey may have happened on another volume. The number in the top-right corner of file operations indicate what files the time-stamps are from before any file operations that combine time-stamps from different files, in this case "Overwriting copy".

10.4.4 Testing

Development of the tool was driven by continuous JUnit tests.

Alpha testing has been performed with various MFT files as input. These MFT files contain file entries on which all (modified) file operations have been performed at least once, including forging operations. Correct matching of time-stamps with file operations has been verified both for files where only a few file operations have happened and files where all eight time-stamps are different from each other.

The author's personal real-life MFTs of over 1GB (over 1 million file entries) were input to statistically test the correct solving of the MFT and ability to interpret many time-stamps without running into obvious problems such as errors.

Chapter 11

Discussion

We have uncovered some workings of the MFT that, while not related to the rest of this study, might be of interest to future digital forensics research. These workings relate to the allocation of entries, and the sequence number.

We have found that when a new file is created, it overwrites the deleted file with the lowest index (except for entries 0 through 34 which are reserved for NTFS meta-data files). If a new file is created while the MFT has no deleted files, then 256 empty, nameless, deleted files are first appended to the MFT.

Each MFT entry has a sequence number, this is a value used by NTFS/Windows to adress files. The sequence number does not work as it is described in literature. According to Carrier [2], the sequence number increments whenever a file is allocated to the entry. We have observed this to be false however. We have not been able to reproduce a sequence number incrementing as a result of allocation. Instead, the sequence number increments only when the file entry is marked as deleted.

11.1 MFT entry allocation

From file deletions and creations the following conclusions about the MFT can be drawn:

```
If a file is deleted:
  The file is marked as deleted and de-allocated.
If a file is created:
  If the MFT is full:
    256 rows are appended after the last index.
    These rows contain deleted files with nulls for meta-data.
  The new file overwrites the deleted file with the lowest index.
```

index	file	entry creation time	deletion time
35	file1	12:00	
36	[DELETED]file2	13:00	16:30
37	file3	14:00	
38	[DELETED]file4	15:00	16:30
39	file5	16:00	

index	file	entry creation time	deletion time
35	file1	12:00	
36	file6	17:00	
37	file3	14:00	
38	[DELETED]file4	15:00	16:30
39	file5	16:00	

Figure 11.1: Basic example of MFT entry allocation. The left table shows part of the MFT before file6 is created. The right table shows the same part of the MFT after. The deleted file entry to be overwritten next is marked in grey.

Figure 11.1 serves as an example, showing the creation of a new file “file6". The newly created file overwrites the deleted file with the lowest index higher than 34. Entries 0 through 34 contain NTFS meta-

data files and additional space for NTFS meta-data files. As long as there is a deleted file at index 36 any newly created file will have to be created at an index smaller or equal to 36, because it must overwrite the deleted file with the lowest index. In the example “file2" and “file4" were deleted at 16:30. Any file created after 16:30 like “file6" can not have an index higher than that of the deleted files.

11.2 The effects of file operations on the sequence number

Testing all known file operations shows that the sequence number starts at 1 and only increments when a file is marked as deleted in the Master File Table. This includes deleting a file by moving it out of the volume, but does not include overwriting a file as this never marks the entry as deleted. The only exception are the NTFS-related file entries located at index 1 through 15, these have a sequence number equal to their index number. These files are protected and can not be deleted, their sequence number is fixed. The sequence number is stored in 2 bytes, it overflows to 0. Thus the sequence number can be defined as follows: sequence number = (x + 1) % 65536 where x = the amount of deletions. This behaviour of the sequence number has been found to be consistent across different Windows versions.

Carrier [2] observed the sequence number to be incremented when a file is allocated. However Carrier made this observation by deleting files and then allocating new files, possibly wrongfully linking the allocation to the incremention.

11.2.1 Possible use of the sequence number: uncover the original ordering of file entries

A file entry that has never been deleted will never have its sequence number incremented. All these files will have a sequence number of 1. Because these file entries have never been marked as deleted, any file entry with a higher index will have been created at a later date. File entries that have been marked as deleted a multiple of 65,536 times will also have a sequence number of 1. These files that have been deleted a multiple of 65,536 times in the same file entry are unlikely to be encountered as files are not expected to be deleted that often, seeing as only 15 out of 100,014,916 NTFS files in the WildFrag database have sequence numbers higher than 50,000.

Chapter 12

Future work

While we have uncovered and explained the effect on time-stamps of more (modified) file operations than past research, not everything has an explanation yet. Furthermore, while we have focused on reproducing file operations from time-stamps in the MFT, higher accuracy could possibly be gained from comparing matched file operations with other meta-data. $LogFile comes to mind for this, as it contains records of changes to time-stamps.

Phenomena left with no explanation

We did not find out under what conditions the quirk happens where SI.E does not get updated. Neither was uncovered under what conditions the last access update will be delayed or aborted. The conditions under which these variations of operations can happen might be forensically interesting, as knowing under what conditions time-stamps are formed will give insight into what happened on the system at that time.

$LogFile

This research and the resulting software focus only on the MFT. It could be expanded upon by also considering $LogFile which contains before and after meta-data of file operations [2]. The usefulness of $LogFile for detecting time-stamp forgery has already shown by Cho [8]. We believe that it could also be used to gain insights into what events happened, combined with time-stamps. For example, if 'rename' is one of the matched operations for an array of time-stamps. Then $LogFile could be searched for a change of file name, to possibly prove that the operation happened and if it did, recover the earlier file name.

Chapter 13

Conclusions

To allow forensic investigators to discover what operations have happened for an array of time-stamps, we have provided a complete overview of file operations and their effects on time-stamps. To construct this overview, past research into this subject has been explored, compared and tested. Some modified file operations missing from past research have also been added. As these file operations are too many and too complex to reliably compare by hand we have delivered a structure and method by which comparing file operations to time-stamps can be automated. This has as an added advantage that file operations can be easily added or removed from some list without it affecting other file operations. Unlike rules derived from time-stamp overviews, which we have shown will often become incorrect if new operations are added or existing ones are changed. We have also implemented this method and structure in an open source tool.

We have made an analysis of irregular time-stamps and possible transference of those time-stamps. Which can be used if an array of time-stamps is suspicious or does not match any regular operations, to uncover if time-stamp forgery has taken place on the system, or elsewhere. From the WildFrag database we have learned that degraded time-stamps are somewhat common with 0.04% of files having time-stamps degraded into the future or far past. This means that most disks will have at least one future or far past time-stamp, so their presence does not prove foul play.

Because of this research, future digital forensics investigation will have greater understanding of time-stamps. Time-stamps may now be used to accurately find the sequences of file operations that may have happened at those times. If no file operations can be found for some time-stamps, then the time-stamps can be compared with forging operations to find evidence of forging and the bytes of the time-stamps can be compared with each other to find evidence of degradation.

Bibliography

[1] E. Casey, "Digital stratigraphy: Contextual analysis of file system traces in forensic science," *Journal of Forensic Sciences*, vol. 63, no. 5, pp. 1383–1391, 2018.

[2] B. Carrier, *File system forensic analysis*. Addison-Wesley Professional, 2005.

[3] S. H. Mahant and B. Meshram, "Ntfs deleted files recovery: Forensics view," *IRACST-International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN*, pp. 2249–9555, 2012.

[4] K.-P. Chow, F. Y. Law, M. Y. Kwan, and P. K. Lai, "The rules of time on ntfs file system," in *Systematic Approaches to Digital Forensic Engineering, 2007. SADFE 2007. Second International Workshop on*. IEEE, 2007, pp. 71–85.

[5] T. Knutson and R. Carbone, "Filesystem timestamps: What makes them tick," *GIAC GCFA Gold Certification*, vol. 11, 2016.

[6] S. Raghavan, "Digital forensic research: Current state of the art," *CSI Transactions on ICT*, vol. 1, 03 2012.

[7] F. P. Buchholz and E. H. Spafford, "On the role of file system metadata in digital forensics," *Digital Investigation*, vol. 1, pp. 298–309, 2004.

[8] G.-S. Cho, "A computer forensic method for detecting timestamp forgery in ntfs," *Computers & Security*, vol. 34, pp. 36–46, 2013.

[9] B. Y. Jewan Bang and S. Lee, "Analysis of changes in file time attributes with file manipulation," *Digital Investigation*, vol. 7, no. 3, pp. 135 – 144, 2011.

[10] X. Ding and H. Zou, "Time based data forensic and cross-reference analysis," in *Proceedings of the 2011 ACM symposium on applied computing*. ACM, 2011, pp. 185–190.

[11] T. Sharma and M. Kaur, "Time rules for ntfs file system for digital investigation," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 4, pp. 1146–1151, 2015.

[12] G. S. Cho, "An intuitive computer forensic method by timestamp changing patterns," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. IEEE, 2014, pp. 542–548.

[13] S. Willassen, "Hypothesis-based investigation of digital timestamps," in *IFIP International Conference on Digital Forensics*. Springer, 2008, pp. 75–86.

Appendix A

Reflection on Process

Yvonne Vollebregt, Robbert Noordzij and I were originally going to write a thesis together. We were all attracted to the project by Hugo Jonker and Vincent van der Meer, which was to find a way to determine a time-line for a disk based on fragmentation information, specifically to date deleted files. Personally I was attracted to this project because it had a clear, complicated and important problem that I wanted to solve: deleted files needed to be dated for evidence in court. When we were doing our literature study to get further acquainted with the workings of file systems, something interesting caught my eye. From the literature it seemed that when a file is deleted it is only marked as deleted. When I asked Hugo and Vincent about this they told me to try it out and see what I would find. When I tried it out, I saw with much enthusiasm the deleted file's meta-data still present in the MFT. My interest for file meta-data only grew from here and when we were deciding what sub-researches we wanted to do I immediately opted for a sub-research about meta-data. I am grateful to Vincent and Hugo for the freedom to take the direction I wanted to take. Yvonne wanted to do her sub-research about the data Vincent had collected in the WildFrag database, while Robbert wanted to do his sub-research about simulating fragmentation. As time progressed and we were all writing our own sub-research for the thesis plan, it became more and more apparent that our sub-researches were very different. Eventually Hugo told us that it would be better to each produce our own thesis.

A.1 Communication

While we each produce our own thesis, we did not split up as group. We have continued to have group meetings, both with and without Vincent and Hugo. We also stayed in contact via Skype, discussing parts of our research and helping each other out if possible. Each two weeks on Monday we would have a Skype call with Hugo and Vincent, usually on the Wednesday before these Skype calls we would submit some part of our thesis(es) or plan for reading. In the beginning Hugo and Vincent would only quickly read the titles and provide feedback on those, but as time progressed their feedback became more detailed, their feedback was always incredibly useful however. We have also provided feedback to each-other, reading each-others parts and providing feedback varying from marking English grammatical errors to asking detailed questions about the contents.

A.2 Planning

When the thesis plan was submitted, I had not realised the complexity of file operations yet. Performing the research I learned of file tunneling, quirks in Windows and the complex working of the NtfsDisableLastAccessUpdate registry key. The original thesis plan was focused on deleted files and forged time-stamps. The focus on deleted files faded as time-stamps are updated the same way for any file, so the research results could apply to any file. Similarly time-stamp forgery became less interesting as regular file operations, as it became clear that time-stamp forgery detection has a larger false positive problem than it does a false negative problem. Because the thesis changed like it did (for the better), the original planning was simply

not accurate anymore. I proceeded to focus on the things that were most important for the research at that time, this also meant more literature study searching for anything that could alter time-stamp calculation.

A mistake that I did make with regards to planning, is postponing writing the research in a way that considered the reader. I did write down a lot of my results but never in a way that considered the reader, until the last few weeks. This caused me to have to spend more time than I would have wanted near the end of the research. Luckily Hugo did help me when I started the final writing by providing useful feedback to get me on the right track.

A.3 Writing

I have never liked writing, and as such I have postponed it while fooling myself into thinking I had not, by writing down barely anything but results early on. I have struggled for a long time with writing during the research, but in the end I feel confident that I have put together a good thesis due to lots of hard work and rewriting.

Appendix B

Overviews of File Operations, Modifiers and Their Effect on Time-stamps

B.1 Base File Operations

Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Create	opStartTime() opStartTime()	opStartTime() opStartTime()	opStartTime() opStartTime()	opStartTime() opStartTime()
Copy	opStartTime() opStartTime()	SRC.SI.W opStartTime()	opEndTime() opStartTime()	opStartTime() opStartTime()
Update	SI.C FN.C	opEndTime() FN.W	opStartTime() FN.E	SI.A FN.A
Move Within volume	SI.C SI.C	SI.W SI.W	opEndTime() SI.E	SI.A SI.A
Move From another volume	SRC.SI.C opStartTime()	SRC.SI.W opStartTime()	opEndTime() opStartTime()	opStartTime() opStartTime()
Overwriting copy	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Overwriting move From another volume	SRC.SI.C FN.C	SRC.SI.W FN.W	opStartTime() FN.E	SI.A FN.A
Rename	SI.C SI.C	SI.W SI.W	opStartTime() SI.E	SI.A SI.A
Attribute change	SI.C FN.C	SI.W FN.W	opStartTime() FN.E	SI.A FN.A
Delete	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Access	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Extract zip file	round(SRC.SI.W, 20000000, up) + tzd opStartTime()	round(SRC.SI.W, 20000000, up) + tzd opStartTime()	opEndTime() opStartTime()	round(SRC.SI.W, 20000000, up) + tzd opStartTime()

B.2 Modifiers

Last access updating enabled				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	...	...	opEndTime()
	...	...	...	...
Update	...	...	...	opEndTime()
	...	...	...	...
Move	...	...	...	opEndTime()
From another volume	...	...	...	...
Overwriting copy	...	...	...	opStartTime()
	...	...	...	...
Overwriting move	...	...	...	opStartTime()
From another volume	...	...	...	...
Access	...	...	...	opStartTime()
	...	...	...	...

File tunneling				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Create	TNL.SI.C	...	...	...
	TNL.FN.C	...	...	...
Copy	TNL.SI.C	...	...	...
	TNL.FN.C	...	...	...
Move	TNL.SI.C	...	...	...
In the same volume	...	...	...	...
Rename	TNL.SI.C	...	...	...
	...	...	...	...

Transfer from FAT				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	round(SRC.SI.W, 20000000, up) + tzd	...	...
	...	...	...	...
Move	round(SRC.SI.C, 100000, up) + tzd	round(SRC.SI.W, 20000000, up) + tzd	...	...
From another volume	...	...	...	...
Overwriting copy	...	round(SRC.SI.W, 20000000, up) + tzd	...	...
	...	...	...	...
Overwriting move	round(SRC.SI.C, 100000, up) + tzd	round(SRC.SI.W, 20000000, up) + tzd	...	...
From another volume	...	...	...	...

Transfer from exFAT				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	round(SRC.SI.W, 100000, up)	...	...
	...	...	...	...
Move	round(SRC.SI.C, 100000, up)	round(SRC.SI.W, 100000, up)	...	...
From another volume	...	...	...	...
Overwriting copy	...	round(SRC.SI.W, 100000, up)	...	...
	...	...	...	...
Overwriting move	round(SRC.SI.C, 100000, up)	round(SRC.SI.W, 100000, up)	...	...
From another volume	...	...	...	...

Quirk				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Copy	...	...	SRC.SI.E	...
	...	...	...	...
Move	...	...	SRC.SI.E	...
From another volume	...	...	...	...

Is directory				
Operation	Resulting time-stamps			
	SI.C FN.C	SI.W FN.W	SI.E FN.E	SI.A FN.A
Update	...	...	...	opEndTime()
	...	...	...	...
Overwriting copy	N/A	N/A	N/A	N/A
	N/A	N/A	N/A	N/A
Overwriting move	N/A	N/A	N/A	N/A
From another volume	N/A	N/A	N/A	N/A
Extract zip file	N/A	N/A	N/A	N/A
	N/A	N/A	N/A	N/A

Appendix C

Diagrams

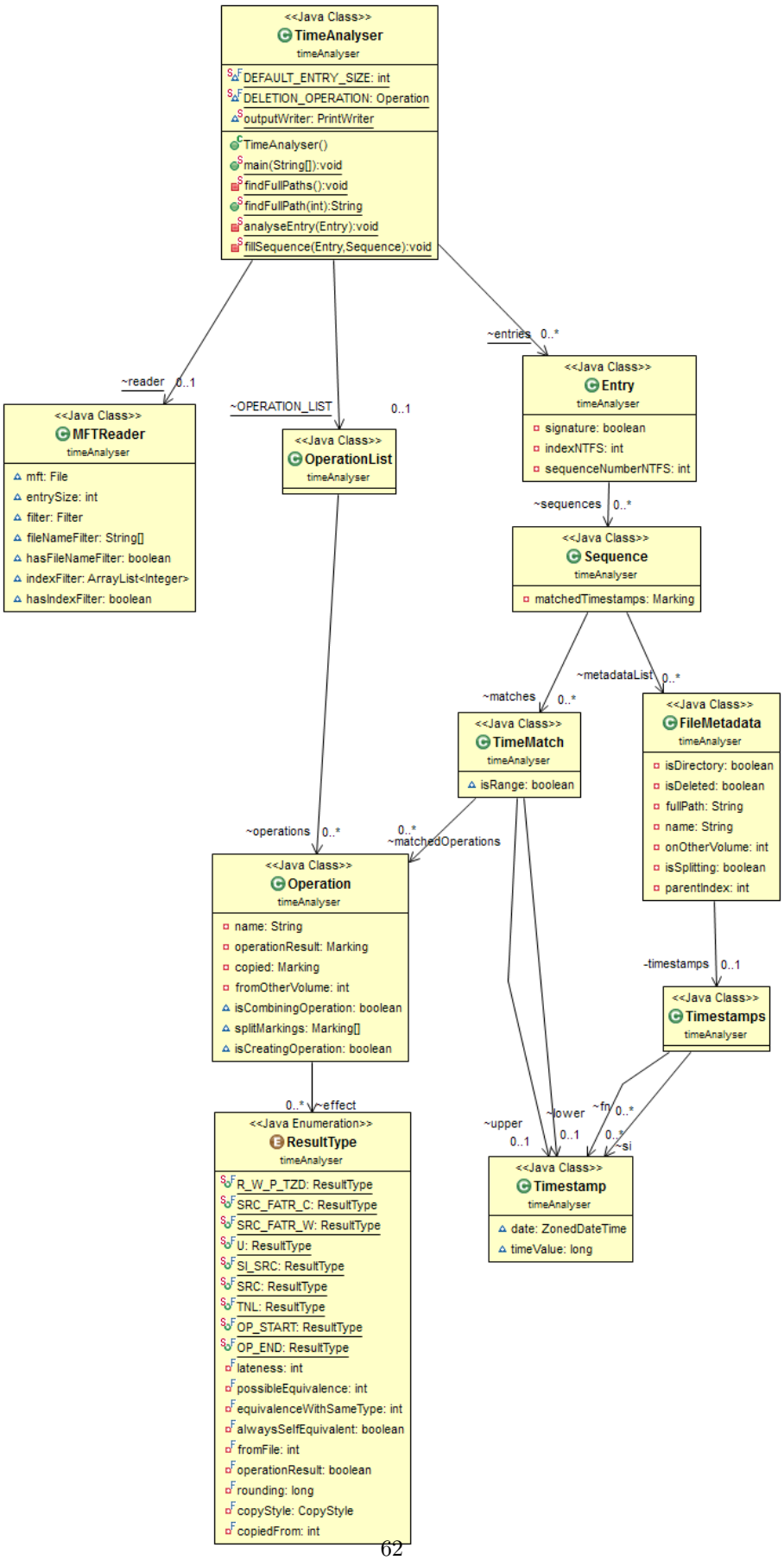

Figure C.1: Class diagram, shows methods only for the main class TimeAnalyser.