



An Investigation into Timestamps in NTFS

LK049 - Bachelor of Science in Cyber
Security and IT Forensics

Final Year Project
Final Report

Nathaniel Patrick Teskey
20247672

Dr. Jacqueline Walker

02/April/2024

Abstract

This project investigates the topic of timestamps within the New Technology File System on Windows. Timeline forgery is now more prevalent than ever. A simple Google search will reveal lots of free, user friendly, timestamp altering (timestomping) tools [1] that make it harder for digital forensic investigators to reconstruct a timeline of events. Timestamps are credible evidence that can be relied upon; however investigators must prove that the timestamp is legitimate.

I will examine the effect of various file operations on different file types and their effect on the timestamps stored at the core of NTFS. I also carried out experiments on timestomping tools such as NewFileTime and SetMACE.

I took an interest in this project as it requires a deep understanding of the file system within Windows. Although I did not know how important or intricate timestamps were before I started my research, I always had an interest in how they worked. The knowledge that I have gained from this project will undoubtedly stick with me throughout my career given this project has such a “real-world” application.

Microsoft has published limited information on the NTFS filesystem [2], let alone the timestamps. This provides an interesting challenge, allowing me to carry out my own experiments into NTFS timestamps, gaining knowledge continuously.

Declaration

This report is presented in part fulfilment of the requirements for the Bachelor of Science in Cyber Security and IT Forensics **Final Year Project**.

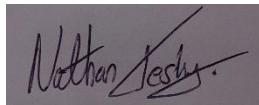
It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.

Where there has been made use of work of other people it has been fully acknowledged and referenced.

Name

Nathaniel Patrick Teskey

Signature



Date

02/April/2024

ACKNOWLEDGEMENTS

I want to extend my sincere thanks to my supervisor Dr. Jacqueline Walker for always giving me valuable advice and constructive criticism, whether that be in a lecture or tutorial, or a meeting when I was looking for some guidance on this project. I am incredibly thankful for the expertise and feedback Dr. Walker provided me with, which really helped me to digest the topic, as it was something I had never explored before.

I would also like to thank my friends and family, especially my parents, Mary and Jimmy, along with my brothers Neil and Ned, for the love, support patience and encouragement that they have always provided. I could not have completed this project without their support.

Table of Contents

ABSTRACT.....	I
DECLARATION.....	II
ACKNOWLEDGEMENTS	III
TABLE OF CONTENTS	IV
LIST OF FIGURES.....	VI
LIST OF TABLES	IX
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND	4
2.1 INTRODUCTION TO NTFS.....	4
2.2 THE MASTER FILE TABLE IN NTFS	5
2.3 TIMESTAMPS IN NTFS	9
2.4 LEGAL AND ETHICAL CONSIDERATIONS.....	12
CHAPTER 3 RELATED WORK	15
CHAPTER 4 METHODOLOGY.....	16
4.1 TESTING ENVIRONMENT	16
4.2 TIMESTAMP BEHAVIOUR EXPERIMENTS	17
4.3 TIMESTAMP FORGERY EXPERIMENTS	18
4.4 TIMESTAMP RETRIEVAL	20
CHAPTER 5 EXPERIMENT PROCEDURE	21
5.1 TIMESTAMP BEHAVIOUR EXPERIMENTS	21
5.1.1 FILE CREATED	21
5.1.2 FILE ACCESSED.....	26
5.1.3 FILE CONTENTS MODIFIED.....	29
5.1.4 FILE RENAMED	30
5.1.5 FILE COPIED.....	30
5.1.6 FILE MOVE.....	31
5.1.7 FILE DELETED.....	31
5.1.8 FOLDER CREATED	33
5.1.9 FOLDER CONTENTS CREATED	34
5.2.0 BASIC TIMESTAMP FORGERY – NEWFILETIME.EXE	35
5.2.1 NEWFILETIME AND FILE RENAME OPERATION	36

5.2.2 ADVANCED TIMESTOMPING WITH SETMAC	37
5.2.3 SETMAC ON WINDOWS XP	41
CHAPTER 6 DISCUSSION OF RESULTS	45
6.1 FILE CREATED (CONTROL)	46
6.2 FILE ACCESSED	49
6.3 FILE CONTENTS MODIFIED	49
6.4 FILE RENAMED	50
6.5 FILE COPIED (ORIGINAL FILES)	50
6.6 FILE COPIED (DESTINATION FILES)	51
6.7 FILE MOVED	51
6.8 FILE DELETED	52
6.9 FOLDER TIMESTAMPS	53
6.9.1 BASIC METHOD: NEWFILETIME	54
6.9.2 NEWFILETIME AND RENAME COMBINATION	55
6.9.3 ADVANCED METHOD: SETMAC	55
CHAPTER 7 LIMITATIONS	57
CHAPTER 8 CONCLUSIONS AND FUTURE WORK	59
REFERENCES	60
APPENDICES	- 1 -
APPENDIX A: UPDATED PROJECT GANTT CHART	- 2 -
APPENDIX B: FINAL PRESENTATION SLIDES	- 3 -
APPENDIX C: PROJECT POSTER	- 9 -

List of Figures

Figure 1: Windows Properties view of file timestamps.	1
Figure 2: Structure of an MFT entry (file record) with a small header, 3 attributes and some unused space at the end. [13]	5
Figure 3: Fragmented MFT, The boot sector is used to find \$MFT and therefor determines the layout of the MFT [13]	6
Figure 4: MFT entry address and sequence number combining to form the file reference address [13]	7
Figure 5 TSK's "istat" output of MFT entry, SIA and FNA.....	8
Figure 6: TSK's "fls" command showing parsed output of portion of MFT.....	9
Figure 7 :Viewing \$SIA timestamps in Windows PowerShell.....	10
Figure 8: Modifying or "timestomping" \$SIA timestamps in Windows PowerShell and displaying modified timestamps.....	10
Figure 9 TSK's "istat" output of MFT entry attributes SIA and FNA.....	11
Figure 10: SIA file times format	12
Figure 11: Windows Disk Management Util view of test device	16
Figure 12: NewFileTimes intuitive GUI.....	18
Figure 13 Image showing SetMace input command.....	19
Figure 14 Image showing SetMace output after input command run	19
Figure 15 Initial test files created, viewed from Windows Explorer.....	21
Figure 16: fdisk showing test device is mounted on SIFT Workstation	21
Figure 17: Image showing MD5 hash of test device being created for file created experiment.	22
Figure 18: Image of "dd" command creating raw image of test device for file created experiments. 22	
Figure 19 MD5 hash generated from raw image of test drive	22
Figure 20 Comparison of MD5 hashes before and after evidence extraction.....	22
Figure 21 Files prepared for analysis in SIFT Workstation.....	23
Figure 22Output of TSK's "fls" command displaying my test files and the MFT entry numbers.	23
Figure 23 SIA and FNA timestamps for the creation of the .txt file.....	24
Figure 24 Created .docx files timestamps.....	24
Figure 25: Created .pdf files timestamps.....	25
Figure 26 Created .pptx files timestamps.....	25
Figure 27 Created .png files timestamps.....	25
Figure 28 MD5 hashes matching for the file accessed experiment.....	26
Figure 29 Text file after file accessed experiment.....	26
Figure 30 Word file after file accessed experiment.....	27
Figure 31 PDF file after accessed experiment.....	27
Figure 32 PowerPoint file after file accessed experiment.....	28
Figure 33 PNG (Image) file after file accessed experiment.....	28
Figure 34 Text file content modified example. All files had this text added to them.	29
Figure 35 Proof of MD5 hashes matching for file content modified experiment.	29
Figure 36 Files displayed in Windows Explorer after being renamed.	30
Figure 37File copy operation preformed.....	30
Figure 38 All 5 original test files moved into newly created folder.....	31
Figure 39 File to be deleted created Windows Explorer view.....	31

Figure 40 File to be deleted content added before deletion.....	31
Figure 41 File deleted and no longer present in Windows Explorer view.	32
Figure 42 MD5 hashes calculated and matching in SIFT Workstation.....	32
Figure 43 File to be deleted viewed at MFT entry number 53. Marked with asterisks – no longer accessible and “-” signifying marked for deletion.	32
Figure 44 Deleted file viewed in Autopsy, marked with the delete flag.....	33
Figure 45 Empty folder created viewed in Windows Explorer.	33
Figure 46 File created inside the folder named “This_is_a_folder”	34
Figure 47 Contents of the file that was created inside the folder.	34
Figure 48 Basic timestomping test file created.	35
Figure 49 Original timestamps viewed in Windows properties view.	35
Figure 50 Test file opened in NewFileTime with modified timestamps about to be set.....	35
Figure 51 Test file opened in NewFileTime with modified timestamps about to be set.....	36
Figure 52 Basic timestomped and then renamed file view in SIFT Workstation using TSK.....	36
Figure 53 File created and contents modified but not timestomped with SetMace, Notepad view. ...	37
Figure 54 setMace_test.txt test file visible in Windows Explorer on my test device.	37
Figure 55 Side by side comparison of TSK’s “istat” output of original unmodified timestamps and Windows properties view of the same original and unmodified timestamps.	38
Figure 56 SetMace.exe command in Windows PowerShell Administrator mode.	38
Figure 57 SetMace.exe output, appears to be a success. However, it has failed.....	39
Figure 58 setMace_test.txt viewed in Windows Explorer after SetMace.exe attempt. Note the missing timestamp value here.	39
Figure 59 SetMace_test.txt viewed in SIFT Workstation using TSK’s “istat” command. Notice the strange timestamp values.....	40
Figure 60 Test device mounted to Windows XP VM and setMace_test file correctly presented.	41
Figure 61 SetMace.exe input and output on Windows XP VM running in Administrator CMD, featuring DeviceIOController error.....	42
Figure 62 Windows XP Properties View showing failed SetMace attempt due to DeviceIOController error caused by running Windows XP in a VM and having the test device as a USB mounted through VirtualBox.....	43
Figure 63 SANS time rules for Windows 7 SIA [24].....	45
Figure 64 SANS time rules for Windows 7 FNA [24]	45
Figure 65 Text File Created results SIA and FNA.....	46
Figure 66 Word file created results SIA and FNA.....	47
Figure 67 PDF file created results SIA and FNA.....	47
Figure 68 PPTC file created results SIA and FNA.....	48
Figure 69 Image file created results SIA and FNA	48
Figure 70 TSK’s istat command confirming that no timestamps were changed after deletion	52
Figure 71 TSK’s “fls” command showing “-” and “*” signifying the file has been deleted.....	52
Figure 72 Autopsy output showing file marked as deleted.	52
Figure 73 Parsed MFT entry shows file status as inactive.	52
Figure 74 Autopsy showing contents of file after deletion, file recovery possible.	53
Figure 75 TSK’s “istat” command displaying initial file times before the folders content was modified.	53
Figure 76 TSK’s “istat” command displaying file times after the folders content was modified.	53

Figure 77 TSK's "istat" command displaying forged SIA file times.	54
Figure 78 TSK's output for the "istat" command showing forged FNA file times after file renamed. .	55
Figure 79 TSK's "istat" command output showing the failed SetMace attempt on Windows 11.	55
Figure 80 "dd" command to take raw image and output	- 1 -
Figure 81 Updated Project Gantt Chart	- 2 -

List of Tables

Table 1 Timestamp behaviour when file was accessed.	49
Table 2 Timestamp behaviour when file had contents modified.	49
Table 3 Timestamp behaviour when file was renamed.	50
Table 4 Timestamp behaviour on original file when file it was copied.	50
Table 5 Timestamp behaviour on destination file when file it was copied.	51
Table 6 Timestamp behaviour when file was moved.	51

Chapter 1 Introduction

Since 1993 the New Technology File System (NTFS) has been the foundation of the Windows operating system (OS) [3]. Timestamps are a key feature within this OS as they provide key information regarding files, such as modification or creation. NTFS's unique structure means that the operating system considers everything as a file. Each file has its own set of unique timestamps. These timestamps are updated when operations are carried out on the file such as file creation, file move or modification of the contents [1]. You may be familiar with timestamps as some timestamps are displayed to the user in the Windows "Properties" view of a file.

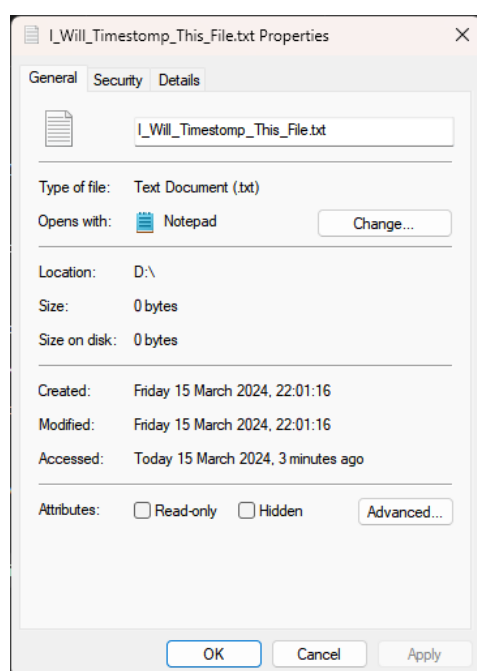


Figure 1: Windows Properties view of file timestamps.

Timestamps serve as crucial metadata that provide an insight into the time a file was created, modified or accessed. This data is however temporary as the timestamps are set each time a new operation is carried out on the file [4]. This means forensic investigators must carefully extract data from the hard drive following good forensic practices to ensure it is not modified during extract as this would make the evidence inadmissible. Analysing these timestamps requires the use of various forensic tools to get deep into the heart of the filesystem and access the locations where these values are stored [5].

The main aim of this project is to determine the set of rules used by Microsoft to define how these timestamps are updated. By comparing various file types and operations on these files I will display my findings based on the experiments I have carried out. I will also compare and contrast the results of my experiments with rules published by computer forensics organisations in to shed some light on how a current version of Windows 11 differs from the version of Windows that was being used in the published rules [6].

Timestamps are trusted pieces of information that are valid in court [7]. However, bad actors can sometimes use anti-forensic methods to hide the trail of their activities making it harder for forensic investigators to prove the truth. A quick Google search returns many tools that can be used to obscure and edit the timestamps within the NTFS filesystem. These tools have varying degrees of usability, for example some are provided with a graphical user interface while others rely on the command line to run. Advanced tools may be difficult for the end user to understand and use, but if the magnitude of the evidence outweighs the time it takes to figure out how to use one of these tools then it can be expected that the end user will figure it out and use it. Later on in this report I will present my findings on basic user-friendly timestamp modification tools and more advanced timestamp forgery tools. The process of artificially modifying timestamps is known in the world of digital forensics as “timestomping” [5]. The name probably derives from the process of stomping out the original valid timestamp and replacing it with the modified time.

To determine valid timestamp behaviour in comparison to “timestomping” it is important that forensic investigators must understand the legitimate behaviour of timestamps in Windows [2]. Documentation on the NTFS filesystem is sparsely documented with not much published online, let alone published by Microsoft themselves [2]. In my opinion this is mainly down to security. If Microsoft published detailed guides on the inner workings of NTFS then it could compromise the security of the entire operating system, similar to how the National Security Agency in the USA have not published a detailed map of the White House or hierarchy of the Secret Service. IT Forensics organisations such as SANS Institute have published limited data regarding timestamp behaviour in NTFS and by comparing their results with my experiments on Windows 11, I will be able to come to conclusions determining if Microsoft have changed the rules for logging timestamps.

Over the course of this report, I will display timestamp changes in NTFS, focusing on version 22631 of Microsoft Windows 11 Home. I carried out experiments on seven different file operations native to Windows which are file created, file accessed, file contents changed, file renamed, file copied, file moved and file deleted. Experiments for each operation were undertaken on 5 different filetypes which are text files, Word files, PDF files, PowerPoint files and image files, I then ran a separate experiment to determine folder timestamp behaviour. The timestamps are stored within attributes inside NTFS’ master file table (MFT). In terms of timestamp alteration I focused on two applications, a basic approach and a more advanced approach. The basic software is called NewFileTime [8] which operates through a GUI and SetMace [9] which is used through a command line interface such as Windows PowerShell. The main metadata attributes considered are \$FILE_NAME (FNA) and \$STANDARD_INFORMATION (SIA).

Firstly I will provide a background on NTFS and the structures within it such as the MFT and the key Timestamp attributes. I will then elaborate on related work in this field, before presenting the methodology regarding my experiments. I will also explain my procedure in depth along with the outcomes and results. I will explain the key limitations of my investigation before providing a conclusion and a plan for future work.

Chapter 2 Background

2.1 Introduction to NTFS

NTFS was first introduced over 30 years ago, as the filesystem at the core of Windows NT 3.1 in 1993 [10]. It took the reins after the file allocation table (FAT) was being phased out by Microsoft. This was a much-needed improvement as FAT was created in 1977 and so much had changed between 1977 and the advent of NTFS [3]. FAT is more suited to devices with smaller storage capacity such as removable storage devices, cameras and smart TVs among other portable devices. NTFS of course can handle larger storage devices as it has a maximum partition of around 2TB [2]. This is essential in modern computers, as they have more storage than was ever imaginable on personal computers in the 1970s. Unlike FAT, NTFS also has the ability to allow backups of the system while it is running – this is known as taking a shadow copy [3]. Interestingly, the NTFS format can also be used on Linux machines.

Some of the mystery around NTFS stems from the fact that Microsoft has only published very limited information regarding its on-disk layout. They have published some high-level details of the filesystem however low-level details have not been published [2]. This is essential to maintain a high level of security within the filesystem. To this day NTFS is the standard filesystem used in Windows operating systems, although it has received various updates over the years, which added features and enhanced security [11].

In 1995 Windows updated NTFS to support compressed files, named streams and they also added access control lists. In October 2001, another key update came, this allowed for the expansion of the Master File Tables entries with redundant MFT file record numbers. This was essential to allow for the recovery of damaged MFT files. Based on my research so far this seems to be the last time NTFS has received a large update [7].

However, with the release of Windows 11 in October 2021, it seems that Microsoft has added support for a new filesystem called the Resilient File System (ReFS) [12]. This new filesystem is currently used in Windows Servers, however unlike NTFS, ReFS is much better at data availability and scalability. This may be a hint that Microsoft plans to ditch NTFS in the future, but who knows. NTFS supports a maximum file size of 256TB [2]; however, ReFS supports

up to 35PetaBytes (1PB is 1024TB) [12]. This is probably one of the reasons why they are currently using ReFS on Windows servers, as they can have much larger storage sizes in comparison to personal computers.

In my opinion NTFS is here to stay for at least another couple of years as ReFS is still in the development stage. It does not yet have support for compression, encryption, disk quotas or removable media [12] – NTFS on the other hand is well equipped to deal with these challenges [2].

2.2 The Master File Table in NTFS

At the heart of NTFS there is a file called the master file table (MFT). This file contains at least one entry for every file within the NTFS volume on the disk. Microsoft calls each of these entries a “file record”. These file records or MFT entries are 1KB (1024 bytes) in size, however only the first 42 bytes have a defined purpose. The remaining bytes are used to store attributes. These attributes all have specific essential purposes, such as storing the files name, storing the files content and so on [13].

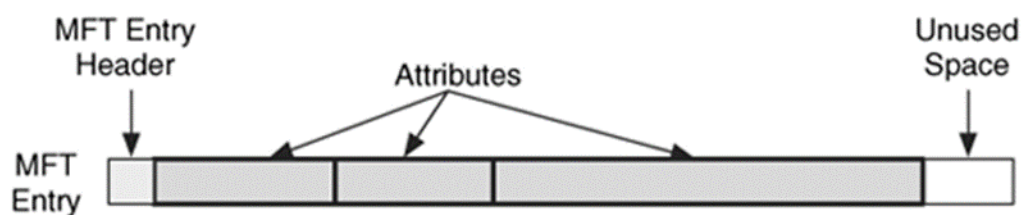


Figure 2: Structure of an MFT entry (file record) with a small header, 3 attributes and some unused space at the end. [13]

The starting location of the MFT is defined in the boot sector. The boot sector is always located in the first sector of the filesystem [13]. Like everything else in NTFS the MFT itself is a file. What is quiet interesting is that the first entry inside the MFT is named \$MFT which is a record of itself on the disk, as seen below the boot sector is used to find this first MFT entry. Below, we can see that the MFT is fragmented and goes from clusters 32 to 34 and then from 56 to 58. Similar to FAT, NTFS uses clusters too [2]. Clusters are groups of consecutive sectors.

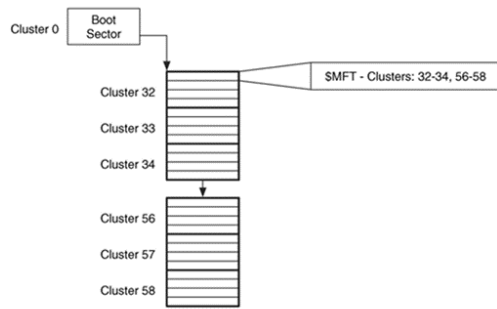


Figure 3: Fragmented MFT, The boot sector is used to find \$MFT and therefor determines the layout of the MFT [13]

NTFS reserves space close to the start of the volume upon formatting [13]. This is done to improve access times when hard disk drives (HDD) were the main form of storage on computers, as it reduced the amount of time disk head spend moving around platter when locating files. This is slightly redundant now with the advent of solid state drives (SSD), however as a general rule of thumb it is nice to be able to find the MFT at the beginning of the volume. Space is reserved in order to keep the MFT as contiguous as possible, which means to keep it all in one piece if possible [13]. There is also a location close to the middle of the volume that is reserved for the MFT, this space is used if the MFT outgrows its space near the start of the volume. This space is known as the MFT zone [10].

If the MFT grows outside this reserved space at the start or middle of the volume, then it is known as a fragmented MFT. Fragmentation greatly slows down data read and write speeds when using a HDD as the disk head would have to go to mulitple positions to find the pointer to the file in the MFT. Thankfully NTFS comes well equipped to prevent MFT fragmentation such as by attempting to reserve enough space that the MFT can stay contiguous. However, fragmentation can still occur due to not having enough space on the disk or preforming many unneccessary file operations which will cause MFT entry numbers and sequence numbers to grow.

The MFT can be defragmented using tools on Windows such as the builtin defrag.exe tool [14]. Defragmenting an MFT can be very beneficial when using a HDD [15], but it is discouraged when using an SSD as the impact of the fragmented MFT on an SSD is much less and it can cause the tiny electronic components inside an SSD to wear out due to wear levelling. Improper practice can result in files being corrupted or data loss when defragmentation occurs. Running defragmentation on an SSD results in trimming being carried out instead of actual defragmentation.

The MFT entry number corresponds to the files inode, which is the files unique identifier number. Each MFT entry is defined by a sequential 48-bit value where the first entry has an address of 0 [13]. Microsoft calls this address the “file number” [2]. NTFS uses file reference addresses to refer the MFT entry’s location. All of the MFT entries have a 16-bit sequence number that is incremented when the entry is allocated [13]. An example being, say that MFT entry number 33 has a sequence number of 1, if the file located at MFT entry number 33 is deleted the MFT entry is then allocated to a new file. However, the new file will have a sequence number of 2, but will still have the MFT entry number set as 33, as it is the second

file to exist at this address. This MFT entry and sequence number are combined, holding the sequence number in the upper 16 bits which then forms the 64-bit file reference address [13].

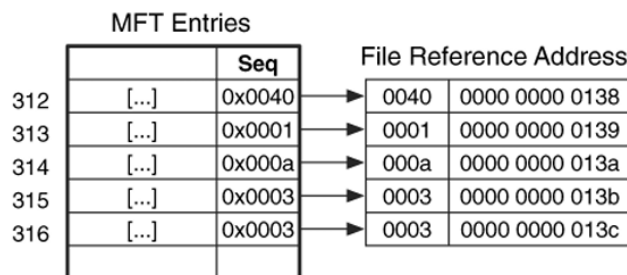


Figure 4: MFT entry address and sequence number combining to form the file reference address [13]

When files are created on the NTFS volume the MFT grows in size as more “file records” are added. Therefore, the MFT is dynamic as it can adjust its size [16]. If a file is deleted from the system, its MFT entry is marked for deletion and can be reused by another file. When the deleted files MFT entry has been deleted another file can take its place at that MFT entry number. However, when a new file fills the MFT entry of the now deleted file, the new file has its sequence number increased. Each time an MFT entry is reused this sequence number is incremented. This stops the NTFS filesystem from checking the MFT and then trying to navigate to the deleted file, which would slow down the OS [16]. Instead the combination of the MFT entry number and sequence number are unique for each file [13].

NTFS uses the file reference address to refer to MFT entries as the sequence number allows it to better determine the location and status of files when the file system is corrupted [13]. This could be due to the system crashing when certain data structures for a file are in the allocation phase, the sequence number can tell NTFS whether a data structure contains an MFT entry address because the previous file used it or because it is part of the newly created file [16]. This is interesting as we can also use it to recover deleted content by checking if there is an unallocated data structure with a file reference number inside it. We can then check if the MFT entry has been moved, since the data structure we are looking at used it [13]. I used forensic tools to view the MFT using a set of tools called “The Sleuth Kit” (TSK) which runs in the command line. It has a GUI version called “Autopsy” [13], however I find TSK much faster to use instead of navigating through Autopsy’s GUI.

Using a tool within TSK called “istat” in the image below, we can see the MFT entry number or inode 39 is accompanied by the sequence number 1. This shows that this file is located at entry 39 of the MFT with a sequence number of 1, therefore it is the first file to have been assigned this location in the MFT. The sequence number and MFT entry number can also be

helpful to identify and restore a deleted file. Also visible are the timestamp attributes within \$FILE_NAME and \$STANDARD_INFORMATION, which ties the MFT into my investigation into timestamps in NTFS.

```
5 total -# ntfs FileViewedTool.Ln 39
HFT Entry Header Values:
Entry: 39      Sequence: 1
SlotFile Sequence Number: 1053912
Allocated File
Links: 1

STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner Id: 0
Security Id: 264 (5-1-5-21-1119575518-1359959274-3140511340-1001)
Created: 2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 17:58:18.424177800 (UTC)
HFT Modified: 2024-03-21 16:48:36.434468800 (UTC)
Accessed: 2024-03-21 16:48:31.229264300 (UTC)

SFILF_NAME Attribute Values:
Flags: Archive
Name: Text1renamed.txt
Parent HFT Modified: 52      Sequence: 1
Allocated Size: 56      Actual Size: 58
Created: 2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 17:58:18.424177800 (UTC)
HFT Modified: 2024-03-15 20:32:40.205556100 (UTC)
Accessed: 2024-03-21 16:48:31.229264300 (UTC)

Attributes:
Type: STANDARD_INFORMATION (16-0) Name: N/A      size: 72
Type: SFILF_NAME (48-5) Name: N/A      Resident size: 194
Your: N/A (20-1) Name: N/A      Resident size: 58
```

Figure 5 TSK's "istat" output of MFT entry, SIA and FNA

After the MFT entry at position 0, the \$MFT which is a record of itself on the disk [13]. The next 15 entries in the MFT are reserved for other NTFS system files such as \$MFTMirror, \$Volume, \$Bitmap and the \$LogFile [13]. These files are known as the file system metadata files. These files are hidden from the Windows Explorer view but they are very important for NTFS to operate. \$MFTMirror is a partial backup of the MFT. It can be used in circumstances when the MFT is corrupted and can be useful to restore the MFT. It normally just contains information about the first few file records of the MFT which are the special files that the filesystem needs to boot and operate [13]. It is generally located in the middle of the disk to act as a protected back up incase the MFT located at the start of the disk is damaged. \$Volume contains information about the volume that NTFS is operating on such as the volume label and volume version. This helps NTFS to recognise its volume by storing its identity. \$Bitmap manages the space on the disk by keeping track of free and used clusters. This allows for efficient allocation of files on the disk and it is constantly updated as disk space is allocated. The \$LogFile records operations before NTFS applies them [13]. In terms of digital forensics, the \$LogFile can be crucial in tracing a timestamped file as the operation of time stamps being changed is recorded in the log, I will speak more about this later on. The \$LogFile mainly serves as a recovery option if the system crashes due to a bad or corrupted file operation, NTFS can check the \$LogFile for the last change and revert back to its previous state [11].

```

sansforensics@slftworkstation: ~/Desktop/Evidence/FileAccessedExperiments
$ fls -f ntfs FileAccessedUSB.ing
r/r 4-128-1: $AttrDef
r/r 8-128-2: $BadClus
r/r 8-128-1: $BadClus:$Bad
r/r 6-128-4: $Bitmap
r/r 7-128-1: $Boot
d/d 11-144-4: $Extend
r/r 2-128-1: $LogFile
r/r 0-128-6: $MFT
r/r 1-128-1: $MFTMirr
r/r 9-128-8: $Secure:$SDS
r/r 9-144-11: $Secure:$SDH
r/r 9-144-5: $Secure:$SII
r/r 10-128-1: $UpCase
r/r 10-128-4: $UpCase:$Info
r/r 3-128-3: $Volume
r/r 42-128-3: PaintPNG.png
r/r 44-128-4: PDFFile.pdf
r/r 41-128-1: PowerPoint.pptx
d/d 36-144-1: System Volume Information
r/r 39-128-1: TextFile.txt
r/r 40-128-1: WordFile.docx
-/r * 43-128-1: ~$PowerPoint.pptx
V/V 256: $OrphanFiles

```

Figure 6: TSK's "fls" command showing parsed output of portion of MFT

The image above shows the TSK's "fls" command being run to display a parsed output of a portion of the MFT, displaying NTFS system metadata files (\$MFT, \$MFTMirror, \$Bitmap, \$LogFile etc.) along with the files inode or MFT entry number [5]. The first entry at MFT entry 0 is the \$MFT, this is at position zero which is the first entry on the volume and where the MFT keeps a record of itself. "d" signifies a directory while "r" signifies a file. Looking at the ~\$PowerPoint.pptx file second from the bottom, we can see that it displays "-/r". The presence of the "-" tells us that the file is unallocated and therefore marked for deletion [13]. However, data may still be recoverable. This is a duplicate of the file located at inode or MFT entry number 41. This behaviour is interesting as I observed it every time that I saved the inode 41 PowerPoint.pptx file.

2.3 Timestamps in NTFS

Microsoft's documentation refers to the timestamps inside the MFT entry as "file times". File times or timestamps are 64-bit values that display the total number of 100 nanosecond intervals that have passed since 12:00AM on the 1st of January 1601 [6]. These timestamps are in the Coordinated Universal Time (UTC) format. The UTC format is unique, as it is not impacted by changes in time zone or daylight-saving time, which makes it excellent for recording timestamps. NTFS records the file times when applications create, access and write to files. File times or timestamps can be updated at different times and as a result of different file operations. The only guarantee about a file timestamp is that the file time is correctly represented when the handle that makes the change is closed [13]. In some cases the NTFS filesystem can delay the update of the last access time for a file by up to 1 hour after the last access time [13].

Within Windows, PowerShell can be used to read some timestamps (\$SIA), similar to the Windows Properties view of the timestamps. I did not this PowerShell method in my

experiments as it accesses the file in order to pull the time, it then updates the accessed timestamp. This is not good forensic practice as the files access file time is being modified during analysis. An example is shown in the image below.

```
PS C:\WINDOWS\system32> Get-ChildItem "D:\TextFilerenamed - Copy.txt" -Force | Select-Object FullName, CreationTime, LastAccessTime, LastWriteTime, Mode, Length

FullName      : D:\TextFilerenamed - Copy.txt
CreationTime   : 15/03/2024 20:43:14
LastAccessTime : 29/03/2024 19:16:31
LastWriteTime  : 15/03/2024 17:58:18
Mode           : -a----
Length        : 50
```

Figure 7: Viewing \$SIA timestamps in Windows PowerShell

Some timestamps can also be updated manually through Windows PowerShell, when run as administrator, however this method was not used in my experiments. This is a basic method of timestamping as it only changes the \$SIA timestamps. This method of timestamp modification allows changes of file time up to one second resolution, modification of millisecond or nanosecond intervals are not permitted. This method is very similar to how some basic GUI “timestamping” tools modify timestamps – namely the NewFileTime application. An example is shown in the image below.

```
PS C:\WINDOWS\system32> Get-ChildItem -force "D:\TextFilerenamed - Copy.txt" * | ForEach-Object{$_ .CreationTime = ("01 February 2014 01:00:00")}; Get-ChildItem -force "D:\TextFilerenamed - Copy.txt" * | ForEach-Object{$_ .LastWriteTime = ("02 March 2014 02:00:00")}; Get-ChildItem -force "D:\TextFilerenamed - Copy.txt" * | ForEach-Object{$_ .LastAccessTime = ("03 April 2014 03:00:00")}
PS C:\WINDOWS\system32> Get-ChildItem "D:\TextFilerenamed - Copy.txt" -Force | Select-Object FullName, CreationTime, LastAccessTime, LastWriteTime, Mode, Length

FullName      : D:\TextFilerenamed - Copy.txt
CreationTime   : 01/02/2014 01:00:00
LastAccessTime : 03/04/2014 03:00:00
LastWriteTime  : 02/03/2014 02:00:00
Mode           : -a----
Length        : 50
```

Figure 8: Modifying or “timestamping” \$SIA timestamps in Windows PowerShell and displaying modified timestamps.

Expanding on each MFT entry number using TSK, even more hidden information about the file is present. In the screenshot below I used the “istat” command to view the timestamp attributes within the files MFT entry. Here we are looking at a file called PowerPoint.pptx. The \$STANDARD_INFORMATION attribute (SIA) values and \$FILE_NAME attribute (FNA) values are visible here. Keep in mind that SIA and FNA file times are not always identical, actually they most often have different file times stored within them. The majority of these timestamps are not visible in the Windows Properties view. The SIA and FNA each contain 4 timestamps which are created, file modified, MFT modified and accessed. However, the FNA can also have an extra four identical timestamps if the file has a longer

file name [13]. We can compare the timestamps within the SIA and FNA to determine the rules Microsoft uses when deciding to update the timestamps.

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 ( )
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 17:12:12.515970700 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PowerPoint.pptx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:20.429818200 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)
```

Figure 9 TSK's "istat" output of MFT entry attributes SIA and FNA

You might assume that the created timestamp is updated when the file was initially created, file modified is updated when the file contents have been changed, MFT modified is updated when the MFT entry for the file is changed for example due to renaming the file or moving it, and accessed is updated each time the file is viewed – however even though these are logical predictions, it couldn't be further from the truth. Over the course of my experiments I have been deeply surprised by the timestamp behaviour in relation to different file types and file operations. It is commonly discussed that the SIA timestamps can be updated by the user through basic “timestomping” methods or file operations, however FNA timestamps can only be updated by the Windows kernel [16]. I will explain my findings on this later on in this report.

```
$STANDARD_INFORMATION Attribute Values:  
Flags: Archive  
Owner ID: 0  
Security ID: 264  ()  
Created:      2024-03-15 15:15:20.429818200 (UTC)  
File Modified: 2024-03-15 15:15:20.429818200 (UTC)  
MFT Modified:  2024-03-15 17:12:12.515970700 (UTC)  
Accessed:      2024-03-15 15:15:20.429818200 (UTC)
```

Figure 10: SIA file times format

As seen in the image above, timestamps in NTFS have a resolution of 100 nanoseconds. Timestamps are represented in the “YYYY-MM-DD HH:mm:ss.MMMmmmN” format [13]. This allows for very accurate timelines to be created. Even though the actual resolution of timestamps is 100 nanoseconds, in real world applications this is not the case. The accuracy can be affected by variations in system hardware or the configuration of the system. One of the main reason for the variance in real world timestamps is due to the system clocks precision which can have a skew of up to 16 milliseconds [17]. This can have an impact on the resolution of the timestamps – therefore, for this investigation differences of up to 16 milliseconds in timestamps have been ignored and these timestamps will be categorised as having the same update time. NTFS sets the 100-nanosecond resolution based on rounding it to the nearest system clock tick [7], which inevitably will come with a small amount of errors in the 100-nanosecond range, although these errors are negligible. In some cases, mainly within organisations timestamp update behaviour can be controlled by the system administrator. This may increase the performance of the computer slightly as if configured, Windows will not update the specified timestamps such as \$SIA accessed to reduce the number of write operations to the disk. This was not the case in my experiments as my laptop is not part of an organisation and uses standard NTFS timestamp rules.

2.4 Legal and Ethical Considerations

It is important to take note of the legal and ethical considerations when working with timestamps. Authenticated timestamps and file timelines are admissible in court as evidence [18]. Privacy laws must be adhered to when dealing with timestamps, as they can reveal sensitive data regarding a person’s activities, such as when a certain document was created or last accessed. In Europe, this legislation is known as GDPR and it is in charge of the rules surrounding the collection, storage and processing of personal data [18].

Digital forensic investigators must also abide by rules surrounding the consent of accessing user’s personal data, such as timestamps. In legal proceedings a warrant may be granted in order for the evidence to be analysed. In a corporate setting, there may be policies in place which allow the company to access sensitive data without consent from the user or employee.

In most cases the process of manipulating file times is a serious breach of the organisations acceptable use policy.

Accessing timestamps without following the correct procedures of consent may result in legal proceedings against the digital forensic investigator and may also disqualify the evidence from being considered [18].

Manipulation of timestamps or timeline forgery, especially in an attempt to obscure the actions taken on a file may obstruct the justice procedure. It is up to investigators to challenge the evidence presented and to find discrepancies within it, indicative of timestamp manipulation. The digital forensic investigator can then piece together an accurate timeline of events, showing the actual file times, and in turn prove that the timestamps were manipulated.

Chapter 3 Related Work

Author Brian Carrier has made a huge impact on the field of digital forensics. He has a deep interest in NTFS which is evident in his book “File System Forensic Analysis”. This book was exceedingly beneficial in helping me to get a good general understand on the NTFS file system. Carrier is also the creator of a key tool used in my experiments. This tool is known as “The Sleuth Kit”. It is well suited to dealing with the NTFS file system. To this day, Carrier is still contributing to the digital forensics’ world through his research [13].

David Palmback and Frank Breitingner have also played a key role in presenting their findings related to the manipulation of NTFS timestamps on Windows. The pair have investigated the intricacies of the \$STANDARD_INFORMATION and \$FILE_NAME attributes. In their report “Artifacts for Detecting Timestamp Manipulation in NTFS on Windows and Their Reliability” [7], they outlined the impact of four new artifacts for tracking timestamps. Their studies presented a deep analysis of the \$UsnJrnl metadata file, link files, prefetch files and also Windows event logs [7]. This allows digital forensics investigators to further expand and contribute on the knowledge base surrounding NTFS timestamps.

Joakim Schicht is a digital forensic analyst and researcher from Norway who has also contributed to the field with his program called SetMace [9]. This is an advanced timestamp forgery tool that he has developed. Unfortunately, it seems that SetMace is now deprecated, however for many years it was a useful too which allowed others to learn about and modify the timestamps in NTFS. On Schicht’s GitHub there are many projects with a key focus on NTFS. They range from LogFile and UsnJrnl parsers to applications for decoding security descriptors in the \$Secure metadata file [9].

Eric Zimmerman is also recognised for his contributions to the field of NTFS timestamps [19]. A former member of the Federal Bureau of Investigations, Zimmerman has developed a set of tools known as EZ Tools. These contain useful components such as NTFS MFT parsers and also a tool for extracting information from the MFT directly from Windows Command Prompt [19].

Chapter 4 Methodology

4.1 Testing Environment

Tests were conducted on a HP Pavilion 14 - CE3501na laptop. The laptop was running Windows 11 Home version 10.0.22631. This indicates that the laptop is based on Windows NT version 10.0. The operating itself is installed on the C: drive. This drive was not involved in my experiments. I conducted my experiments on a Verbatim Store n' Go 32 GB USB drive.

The USB drive was named “TESTUSB” and was mounted as drive D:. The initial size of the USB drive was 32GB, I created a testing partition of 64 MB. This made the USB drive display as two separate storage devices. The second partition was mounted as drive E:. This drive was named “DONOTUSE” and not used and therefore, not involved in testing. This drive E: consisted of the remainder of the USB drives ~ 28GB of space and was formatted in FAT32.

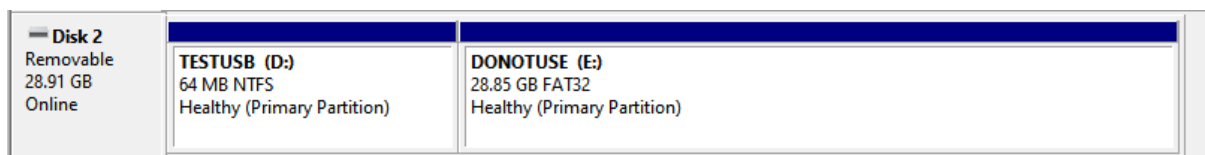


Figure 11: Windows Disk Management Util view of test device

I created the “TESTUSB” partition through Windows’ Disk Management utility. The partition was formatted in NTFS with a size of 64MB. After formatting its size was 63.9MB. The reason for formatting was to reduce the size of the drive to the minimum amount of space required to carry out my tests, as taking a raw image and generating the hashes of the data took far too long on the full 32GB. In addition to this my laptop did not have the capacity to store multiple raw images of larger size.

4.2 Timestamp Behaviour Experiments

File Types

I conducted my experiments on 5 file types commonly found in Windows. These files were a text file named TextFile.txt, a Microsoft Word file named WordFile.txt, a Microsoft PowerPoint file named PowerPoint.pptx, a PDF file named PDFFile.pdf and a PNG image file named PaintPNG.png.

The programs required to create these files were Microsoft Notepad, Microsoft Word, Microsoft PowerPoint, Microsoft Paint, Google Chrome and Geek Software's PDF24 Creator. Microsoft Notepad was used to create and edit .txt file. Microsoft Word was used to create and edit .docx file. Microsoft PowerPoint was used to create and edit the .pptx file. Microsoft Word was also used to create the .pdf file while PD24 Creator was used to edit the .pdf file and Google Chrome was used to view the PDF file. Microsoft Paint was used to create and edit the .png file.

File Operations

I carried out 7 commonly used operations on these 5 test files. The operations were file creation, file access, file contents changed, file renamed, file copied and file moved.

Files were created, accessed and had their content modified through the applications mentioned previously. The file rename operation was carried out through Windows Explorer. The copy operation was performed by creating a new folder on TESTUSB and copying the files to this new folder. The file move operation was done through Windows Explorer using the drag and drop feature to move files into the folder. File deletion was preformed through Windows Explorer, files did not go to the recycling bin after deletion.

Folders

I also carried out an investigation into folder behaviour in NTFS. Everything is considered a file, even directories so I was interested to examine how folder timestamps behave [13]. I created a new folder for this experiment and then created a file inside the folder. I studied the timestamps of the folder to see how they updated if the file inside the folder was changed.

4.3 Timestamp Forgery Experiments

When performing experiments on timestamp forgery or timestomping, I took two routes. Which were a basic method of timestamp forgery and an advanced method. The basic method consisted of using the NewFileTime application while the advanced method used the SetMace program.

NewFileTime is a program that has the capabilities to modify SIA timestamps. I carried out experiments using this application, regarding basic timestomping methods. I downloaded NewFileTime version 7.16 from Google. When searching for timestamp modification software, this is one of the most popular approaches, hence my reasoning for choosing it. It is a very lightweight application with an intuitive GUI which is very easy to navigate even for non-tech savvy users. It even features a file drag and drop feature along with picking the date from a calendar. It has the ability to timestomp the modified, created and accessed timestamps in the SIA to a granularity of one second intervals. It is considered a basic timestomping method as it does not have the ability to update the FNA file times, nor can it change the millisecond or nanosecond data.

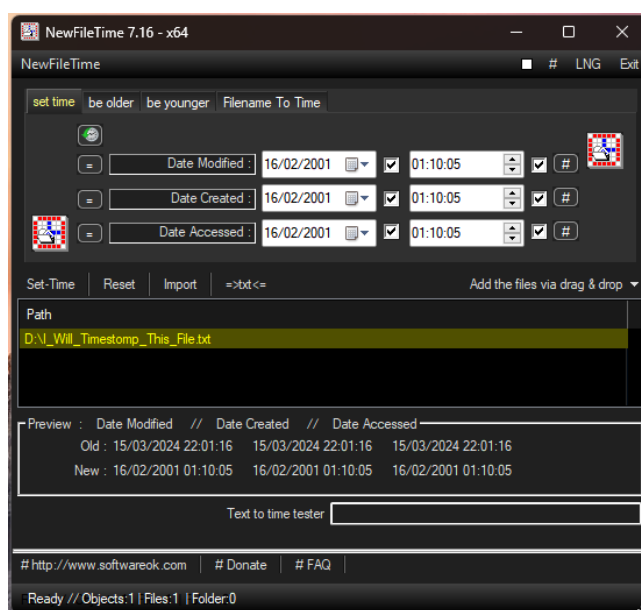


Figure 12: NewFileTimes intuitive GUI

I also carried out investigations using a more advanced timestomping method, namely SetMace by Joakim Schicht. This is a command line tool that can be run in either Windows Command Prompt (CMD) or PowerShell. It caters only for the NTFS file system within Windows [9]. I downloaded version 1.0.0.16 from GitHub. It relies on taking advantage of the filesystem's internal workings, allowing it to write directly to the physical disk. Administrator privileges are required for this method of timestomping. It has the ability to modify timestamps down to nanosecond granularity. The version of SetMace that I used was

released in 2014 and has not received an update since then, this is the current version [9]. Knowledge of the command line is required to use SetMace as it features many parameters which affect the operations the program undertakes; users must also have knowledge of correct file path specification in the command line. In order for SetMace to work correctly on systems running Windows NT version 6.x (Windows Vista, Windows 7, Windows 8) and up another program called KPP Destroyer [20] is required in conjunction with SetMace, although compatibility with Windows 11 is not confirmed in the documentation, I still tested it. Windows 11 runs version 10.x of Windows NT. KPP Destroyer allows SetMace to work on Windows 64-bit systems as it circumvents one of Microsoft's security measures called PatchGuard, which protects the kernel in memory and prevents the OS from loading unsigned drivers or drivers that are signed with a test certificate [9]. In my results I will explain my experiments using SetMace and its outcomes on a current version of Windows 11.

```
PS C:\Users\nptes\OneDrive - University of Limerick\Desktop\Cybersecurity Y4 S2\FYP REPORT ACTUAL STUFF\SetMace-master\SetMace-master> ./SetMace.exe "D:\setMACE_test.txt" -z "2001:16:02:01:10:05:123:4567" -x
```

Figure 13 Image showing SetMace input command

```
PS C:\Users\nptes\OneDrive - University of Limerick\Desktop\Cyber
3:4567" -x
Starting SetMace by Joakim Schicht
Version 1.0.0.16

Target filename: setMACE_test.txt
Target fileref: 53
Target MFT record offset: 0x01562400
Parent filename: .
Parent fileref: 5
Parent MFT record offset: 0x01556400

Start patching timestamps

Trying volume offset 0x01562400
Error: NtLoadDriver: 0xC0000428
Error: Could not load driver
Attempting write to physical disk without driver
Success dismounting D:
Success writing timestamps

Patching resident INDX records of parent ($INDEX_ROOT)

Trying volume offset 0x01556400
Warning: The index in $INDEX_ROOT is not resident any more.

Patching non-resident INDX records of parent ($INDEX_ALLOCATION)

Trying volume offset 0x00024000
Error: NtLoadDriver: 0xC0000428
Error: Could not load driver
Attempting write to physical disk without driver
Success dismounting D:
Success writing timestamps

File system cache cleared in RAM

Job took 2 seconds
PS C:\Users\nptes\OneDrive - University of Limerick\Desktop\Cyber
```

Figure 14 Image showing SetMace output after input command run

4.4 Timestamp Retrieval

Timestamps of the test files were taken from the values stored within the MFT at \$SIA and \$FNA. This procedure had to be repeated in the same sequence when retrieving the results of the timestamps for each file. The analysis took place on a VirtualBox virtual machine (VM) running SIFT Workstation. This is an Ubuntu machine packed pre-packaged with a collection of open-source tools facilitating digital investigations [6].

After a test has been carried out on the test device, the following procedure is adhered to:

1. SIFT Workstation is launched from VirtualBox.
2. TESTUSB is mounted to SIFT Workstation.
3. MD5 hash of mounted TESTUSB is calculated and saved to file.
4. Raw image is taken of TESTUSB using the “dd” command.
5. MD5 hash of raw image is calculated and saved to file.
6. Hashes are compared ensuring the evidence has not been altered during extraction.
7. TSK’s “fls” command is run to view MFT entry number.
8. TSK’s “istat” command is run to view MFT entry of test file.
9. Check for changes in SIA and FNA file time values.
10. Record and screenshot results.

Within the SIA and FNA there are 4 time values. These are known in digital forensics as MACB values [3].

- “M” stands for modified and is represented by the File Modified timestamps.
- “A” stands for accessed and is represented by the File Accessed timestamps.
- “C” stands for changed and is represented by the MFT Modified timestamps.
- “B” stands for born and is represented by the Created timestamps.

Timestamp behaviour results are then displayed along with an explanation.

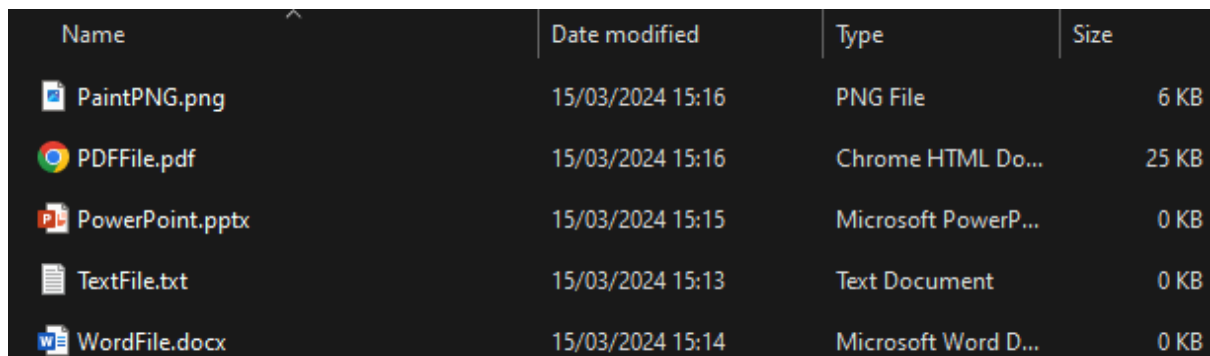
Chapter 5 Experiment Procedure

5.1 Timestamp Behaviour Experiments

These experiments were carried out to gain an insight into timestamp behaviour in NTFS. The file types and operations in this section are all commonly used and permitted on Windows. These results serve as an understanding into normal timestamp behaviour.

5.1.1 File Created

This was the first experiment that I ran, it also served to create the 5 test files. Files were created by their relevant applications in the test environment. As noted in the image below the five files are present on the USB mounted at D: on the testing environment. At this point the files have never been accessed or modified.








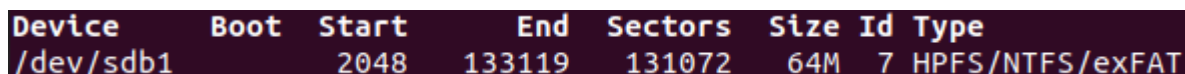
Name	Date modified	Type	Size
 PaintPNG.png	15/03/2024 15:16	PNG File	6 KB
 PDFFile.pdf	15/03/2024 15:16	Chrome HTML Do...	25 KB
 PowerPoint.pptx	15/03/2024 15:15	Microsoft PowerP...	0 KB
 TextFile.txt	15/03/2024 15:13	Text Document	0 KB
 WordFile.docx	15/03/2024 15:14	Microsoft Word D...	0 KB

Figure 15 Initial test files created, viewed from Windows Explorer

I then mounted the USB to my SIFT Workstation VM. Running the “fdisk” command displayed the test device mounted at “/dev/sdb1”. The size is consistent with TESTUSB’s size at 64MB.



Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdb1		2048	133119	131072	64M	7	HPFS/NTFS/exFAT

Figure 16: fdisk showing test device is mounted on SIFT Workstation

I then calculated the MD5 hash of the drive mounted at “/dev/sdb1”. This was saved to a file called md5USB.txt. The hash calculated quickly due to the small size of the test device.

```
sansforensics@siftworkstation: /media/sansforensics
$ sudo md5sum /dev/sdb1 > ~/Desktop/Evidence/FileCreatedExperiments/md5USB.txt
```

Figure 17: Image showing MD5 hash of test device being created for file created experiment.

Next, I used the “dd” command to create a raw image of my evidence from the test device. This was done to ensure that the evidence present on the test device was not modified or altered by me during my investigation. The small size of the test device was beneficial in allowing me to create raw images quickly. I saved this raw image as FileCreatedUSB.img.

```
sansforensics@siftworkstation: /media/sansforensics
$ sudo dd if=/dev/sdb1 of=~/Desktop/Evidence/FileCreatedExperiments/FileCreatedUSB.img
131072+0 records in
131072+0 records out
67108864 bytes (67 MB, 64 MiB) copied, 0.925367 s, 72.5 MB/s
```

Figure 18: Image of “dd” command creating raw image of test device for file created experiments.

Then I calculated the MD5 hash of the raw image, in preparation to compare both MD5 hashes. I saved this in a text file called md5Image.txt.

```
sansforensics@siftworkstation: /media/sansforensics
$ md5sum ~/Desktop/Evidence/FileCreatedExperiments/FileCreatedUSB.img > ~/Desktop/Evidence/FileCreatedExperiments/md5Image.txt
```

MD5 hash being generated for the raw image of the test device.

Figure 19 MD5 hash generated from raw image of test drive

I then used Linux’s “cat” command to display the text within the two MD5 hash files. The two hashes were identical; therefore the evidence was not modified during extraction.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileCreatedExperiments
$ cat md5USB.txt
9bac9f88311bbb8e4a0e263165463b3a  /dev/sdb1
sansforensics@siftworkstation: ~/Desktop/Evidence/FileCreatedExperiments
$ cat md5Image.txt
9bac9f88311bbb8e4a0e263165463b3a  /home/sansforensics/Desktop/Evidence/FileCreatedExperiments/FileCreatedUSB.img
```

Figure 20 Comparison of MD5 hashes before and after evidence extraction.

As this was my first experiment, I created some unnecessary files in error. The files considered in this experiment were FileCreatedUSB.img, md5Image.txt and md5USB.txt. Please disregard the other files in the image below.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileCreatedExperiments
$ ls
FileCreatedMftAnalyzed.csv  FileCreatedMft.raw  FileCreatedUSB.img  md5Image.txt  md5USB.txt
```

Figure 21 Files prepared for analysis in SIFT Workstation

Next I used TSK’s “fls” command to display a parsed output of key information in the MFT [5]. Following correct digital forensics procedures, this was being run on the raw image that I extracted previously. The key information that is considered from this part of the investigation is the MFT entry number that points to the MFT entry of my test files.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileCreatedExperiments
$ fls -f ntfs FileCreatedUSB.img
r/r 4-128-1:  $AttrDef
r/r 8-128-2:  $BadClus
r/r 8-128-1:  $BadClus:$Bad
r/r 6-128-4:  $Bitmap
r/r 7-128-1:  $Boot
d/d 11-144-4: $Extend
r/r 2-128-1:  $LogFile
r/r 0-128-6:  $MFT
r/r 1-128-1:  $MFTMirr
r/r 9-128-8:  $Secure:$SDS
r/r 9-144-11: $Secure:$SDH
r/r 9-144-5:  $Secure:$SII
r/r 10-128-1: $UpCase
r/r 10-128-4: $UpCase:$Info
r/r 3-128-3:  $Volume
r/r 42-128-3: PaintPNG.png
r/r 44-128-4: PDFFile.pdf
r/r 41-128-1: PowerPoint.pptx
d/d 36-144-1: System Volume Information
r/r 39-128-1: TextFile.txt
r/r 40-128-1: WordFile.docx
-/r * 43-128-4: msoF559.tmp
V/V 256:      $OrphanFiles
```

Figure 22 Output of TSK’s “fls” command displaying my test files and the MFT entry numbers.

The .txt file is present at MFT entry 39. The .docx file is present at MFT entry 40. The .pdf file is present at MFT entry 44. The .pptx file is present at MFT entry 41 and the .png file is present at MFT entry 42.

Running TSK's "istat" command below, I was able to view the key timestamp attributes inside the MFT entry for the specified test file. Expanding on MFT entry 39 displayed the SIA and FNA timestamps of the file.

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 ( )
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:14:31.165472300 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: TextFile.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:13:56.390481700 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)
```

Figure 23 SIA and FNA timestamps for the creation of the .txt file.

Looking at the image above, it is evident that the timestamps are all the same except SIA MFT Modified. This could be due to the way NTFS updates its timestamps and the system could have been busy at the time causing a delayed write to the MFT [6]. This is normal behaviour of NTFS.

Displaying the SIA and FNA timestamps on file creation for 4 remaining test files:

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 ( )
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 15:14:54.146772200 (UTC)
Accessed:      2024-03-15 15:14:42.710238100 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: WordFile.docx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 15:14:42.710238100 (UTC)
Accessed:      2024-03-15 15:14:42.710238100 (UTC)
```

Figure 24 Created .docx files timestamps.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.122964600 (UTC)
Accessed:      2024-03-15 15:20:53.154764800 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PDFFile.pdf
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 28672      Actual Size: 25600
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.091672300 (UTC)
Accessed:      2024-03-15 15:16:57.091672300 (UTC)

```

Figure 25: Created .pdf files timestamps.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:31.158764000 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PowerPoint.pptx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0      Actual Size: 0
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:20.429818200 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

```

Figure 26 Created .pptx files timestamps.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.736927400 (UTC)
MFT Modified:  2024-03-15 15:16:17.736927400 (UTC)
Accessed:      2024-03-15 15:16:17.736927400 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PaintPNG.png
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0      Actual Size: 0
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.705622300 (UTC)
MFT Modified:  2024-03-15 15:16:17.705622300 (UTC)
Accessed:      2024-03-15 15:16:17.705622300 (UTC)

```

Figure 27 Created .png files timestamps.

5.1.2 File Accessed

This experiment ran through the exact same steps as the previous experiment, following the same analysis procedure. I ran this experiment on the same files that I created in the previous experiment. In order to perform the file access operation, I implemented Windows Explorer to open my test device. I then double-clicked each of the files to open them in their specified application and then closed them again, using the red “X” in the top right corner, ensuring no modification, saving or renaming occurred.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileAccessedExperiments
$ cat MD5USBAccessed.txt
fc3093ddad7721574992c21389331d62  /dev/sdb1
sansforensics@siftworkstation: ~/Desktop/Evidence/FileAccessedExperiments
$ cat MD5ImageAccessed.txt
fc3093ddad7721574992c21389331d62  FileAccessedUSB.img
```

Figure 28 MD5 hashes matching for the file accessed experiment.

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:14:31.165472300 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: TextFile.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:13:56.390481700 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)
```

Figure 29 Text file after file accessed experiment.

As shown above, no change was recorded from accessing the text file in Notepad as the timestamps are the same since file creation.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 17:12:30.367136300 (UTC)
Accessed:      2024-03-15 17:12:30.184359600 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: WordFile.docx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 15:14:42.710238100 (UTC)
Accessed:      2024-03-15 15:14:42.710238100 (UTC)

```

Figure 30 Word file after file accessed experiment.

As shown above, none of the FNA timestamps have changed however the SIA timestamp values MFT Modified and Accessed were updated after the file access occurred.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.122964600 (UTC)
Accessed:      2024-03-15 17:12:02.373050100 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PDFFile.pdf
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 28672     Actual Size: 25600
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.091672300 (UTC)
Accessed:      2024-03-15 15:16:57.091672300 (UTC)

```

Figure 31 PDF file after accessed experiment.

The only changed noted here is that SIA Accessed was updated when the file was accessed.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 17:12:12.515970700 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PowerPoint.pptx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:20.429818200 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

```

Figure 32 PowerPoint file after file accessed experiment.

FNA displayed no changes after the file had been accessed however, SIA MFT Modified was updated.

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.736927400 (UTC)
MFT Modified:  2024-03-15 15:16:17.736927400 (UTC)
Accessed:      2024-03-15 17:11:49.635230800 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PaintPNG.png
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.705622300 (UTC)
MFT Modified:  2024-03-15 15:16:17.705622300 (UTC)
Accessed:      2024-03-15 15:16:17.705622300 (UTC)

```

Figure 33 PNG (Image) file after file accessed experiment.

FNA displayed no change to its timestamps, however the SIA Accessed was updated.

As you can see from the experiments so far, it seems that each file type has a different procedure of updating the timestamps in the SIA and FNA.

5.1.3 File Contents Modified

All files had their contents modified through their relevant applications. All file content modifications consisted of adding text to the file. The text added is displayed in the image below. This was the same content for all files. It is important to note that modifying a PDF file is not normally possible or good practice, however I still ran an experiment on it using PDF24 Creator.

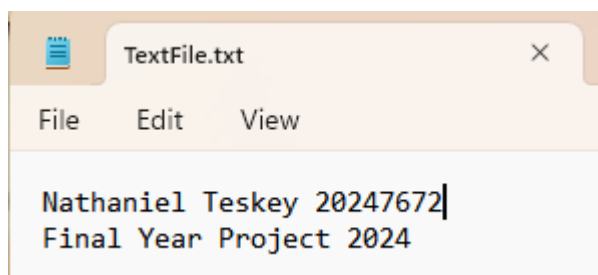


Figure 34 Text file content modified example. All files had this text added to them.

The evidence was extracted in SIFT Workstation following the same procedure as the file accessed and file created experiments shown previously.

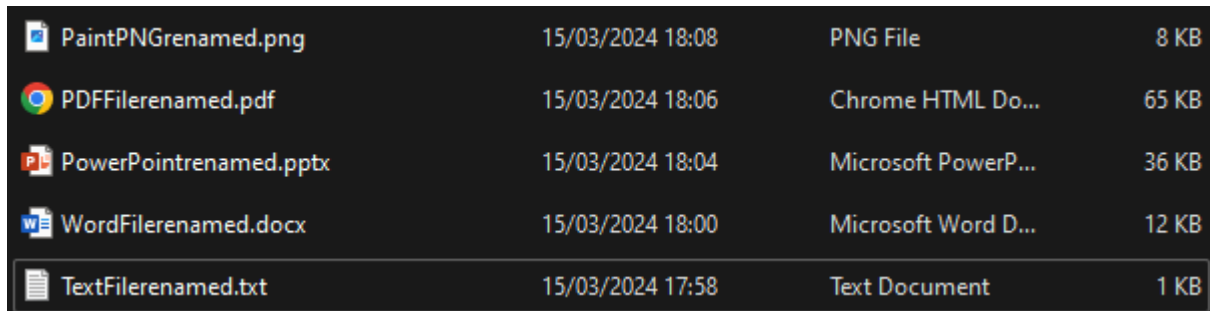
```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileContentsChangedExperiments
$ cat MD5USBContentChanged.txt
c6443fb4d7610219b0fbd40992cd56ef /dev/sdb1
sansforensics@siftworkstation: ~/Desktop/Evidence/FileContentsChangedExperiments
$ cat MD5ImageContentChanged.txt
c6443fb4d7610219b0fbd40992cd56ef FileContentChangedUSB.img
```

Figure 35 Proof of MD5 hashes matching for file content modified experiment.

I will explain the results in the results and discussion section later in the report for this experiment and future experiments. Proof of content modified and related images will be attached in the appendices section for this experiment and future experiments.

5.1.4 File Renamed

All files were renamed through Windows Explorer by appending “renamed” to the files name as shown below. Files were not modified or accessed during this experiment.








	PaintPNGrenamed.png	15/03/2024 18:08	PNG File	8 KB
	PDFFilerenamed.pdf	15/03/2024 18:06	Chrome HTML Do...	65 KB
	PowerPointrenamed.pptx	15/03/2024 18:04	Microsoft PowerP...	36 KB
	WordFilerenamed.docx	15/03/2024 18:00	Microsoft Word D...	12 KB
	TextFilerenamed.txt	15/03/2024 17:58	Text Document	1 KB

Figure 36 Files displayed in Windows Explorer after being renamed.

5.1.5 File Copied

The file copy operation was carried out in Windows Explorer by right clicking each file and selecting copy and then selecting paste. This appended copy to the name of the destination files, after they were pasted. This is due to them being in the same directory. Two files with the same name cannot exist in the same directory. During this experiment, I examined both the original and the new duplicate (pasted) files timestamps.











Name	Date modified	Type	Size
 PaintPNGrenamed.png	15/03/2024 18:08	PNG File	8 KB
 PDFFilerenamed.pdf	15/03/2024 18:06	Chrome HTML Do...	65 KB
 PowerPointrenamed.pptx	15/03/2024 18:04	Microsoft PowerP...	36 KB
 TextFilerenamed.txt	15/03/2024 17:58	Text Document	1 KB
 WordFilerenamed.docx	15/03/2024 18:00	Microsoft Word D...	12 KB
 PaintPNGrenamed - Copy.png	15/03/2024 18:08	PNG File	8 KB
 PDFFilerenamed - Copy.pdf	15/03/2024 18:06	Chrome HTML Do...	65 KB
 PowerPointrenamed - Copy.pptx	15/03/2024 18:04	Microsoft PowerP...	36 KB
 TextFilerenamed - Copy.txt	15/03/2024 17:58	Text Document	1 KB
 WordFilerenamed - Copy.docx	15/03/2024 18:00	Microsoft Word D...	12 KB

Figure 37File copy operation preformed.

5.1.6 File Move

I created a folder inside my test device. I then used Windows Explorer's drag and drop feature to move the original 5 test files into the newly created folder. In the image below you can see that the files are now located on my test device and inside the new folder name "Files_moved_here_folder".

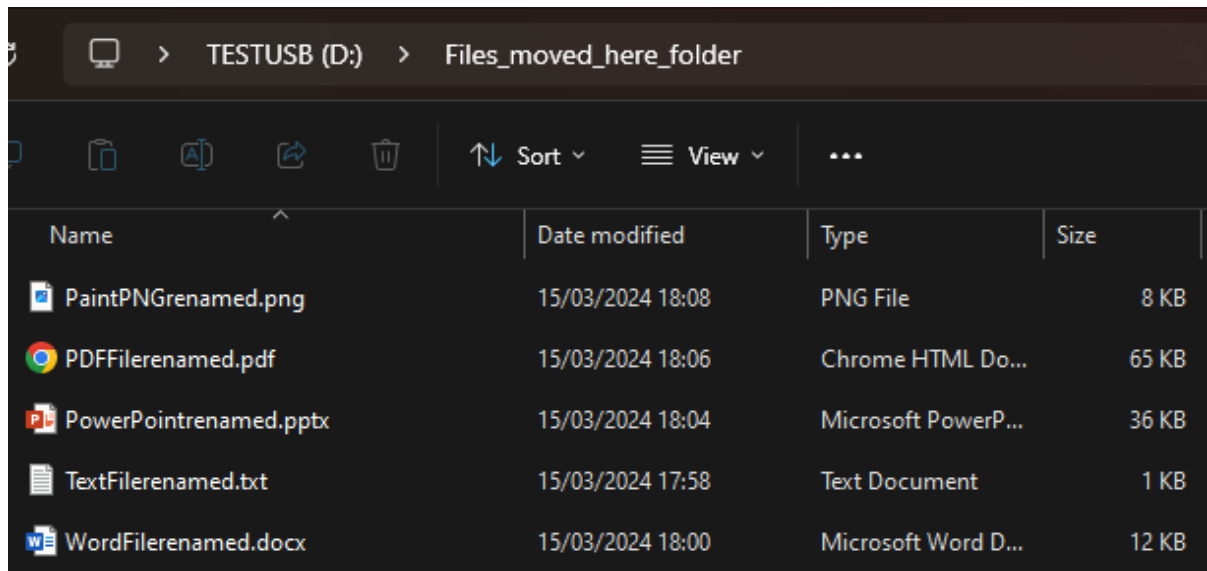


Figure 38 All 5 original test files moved into newly created folder.

5.1.7 File Deleted

To test the impact deletion had on a file's timestamps, I created a new text file with content called "file_to_be_deleted.txt". I was curious on how the timestamps would be affected and also how the MFT entry would be affected. I approached this experiment slightly differently to the rest. I will outline my approach now.

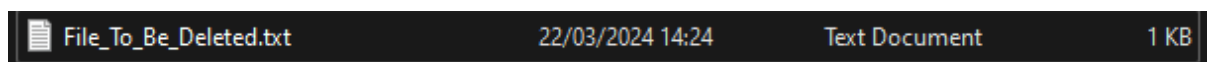


Figure 39 File to be deleted created Windows Explorer view.

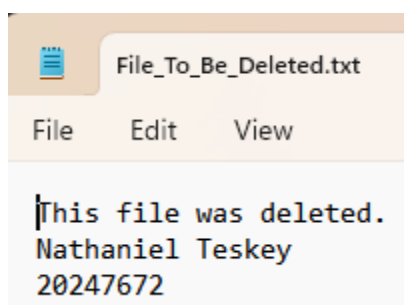


Figure 40 File to be deleted content added before deletion.

Files_moved_here_folder	21/03/2024 16:48	File folder	
This_is_a_folder	15/03/2024 21:38	File folder	
I_Will_Timestamp_This_File renamed.txt	16/02/2001 01:10	Text Document	0 KB
PaintPNGrenamed - Copy.png	15/03/2024 18:08	PNG File	8 KB
PDFFilerenamed - Copy.pdf	15/03/2024 18:06	Chrome HTML Do...	65 KB
PowerPointrenamed - Copy.pptx	15/03/2024 18:04	Microsoft PowerP...	36 KB
TextFilerenamed - Copy.txt	15/03/2024 17:58	Text Document	1 KB
WordFilerenamed - Copy.docx	15/03/2024 18:00	Microsoft Word D...	12 KB

Figure 41 File deleted and no longer present in Windows Explorer view.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileDeletedExperiments
$ cat MD5USBDeleted.txt
e5453426d51a32d6eb3ba5cc5a53e4a6 /dev/sdb1
sansforensics@siftworkstation: ~/Desktop/Evidence/FileDeletedExperiments
$ cat MD5ImageDeleted.txt
e5453426d51a32d6eb3ba5cc5a53e4a6 FileDeletedUSB.img
```

Figure 42 MD5 hashes calculated and matching in SIFT Workstation.

As you can see in the image below, after TSK's "fls" command was run, a seemingly normal parsed view of a portion of the MFT is presented. Note at MFT entry number 53, near the bottom of the image, that an asterisk is present this means that the file is no longer accessible via the standard Windows Explorer approach [5]. The dash (-) is also significant as this symbol, when present in an MFT entry means that the file has been marked for deletion however can still be recovered [5].

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileDeletedExperiments
$ fls -f ntfs FileDeletedUSB.img
r/r 4-128-1: $AttrDef
r/r 8-128-2: $BadClus
r/r 8-128-1: $BadClus:$Bad
r/r 6-128-4: $Bitmap
r/r 7-128-1: $Boot
d/d 11-144-4: $Extend
r/r 2-128-1: $LogFile
r/r 0-128-6: $MFT
r/r 1-128-1: $MFTMirr
r/r 9-128-8: $Secure:$SDS
r/r 9-144-11: $Secure:$SDH
r/r 9-144-14: $Secure:$SII
r/r 10-128-1: $UpCase
r/r 10-128-4: $UpCase:$Info
r/r 3-128-3: $Volume
d/d 52-144-7: Files_moved_here_folder
r/r 51-128-1: I_Will_Timestamp_This_File renamed.txt
r/r 40-128-1: PaintPNGrenamed - Copy.png
r/r 41-128-1: PDFFilerenamed - Copy.pdf
r/r 46-128-1: PowerPointrenamed - Copy.pptx
d/d 36-144-1: System Volume Information
r/r 47-128-1: TextFilerenamed - Copy.txt
d/d 49-144-1: This_is_a_folder
r/r 48-128-1: WordFilerenamed - Copy.docx
-/r * 53-128-1: File_To_Be_Deleted.txt
V/V 256: $OrphanFiles
```

Figure 43 File to be deleted viewed at MFT entry number 53. Marked with asterisks – no longer accessible and "-" signifying marked for deletion.

I also explored the results in Autopsy. Autopsy is a GUI front for TSK. I will speak more on my findings regarding the deleted file in the results section.

DEL	Type	NAME
	dir / in	
Error Parsing File (Invalid Characters?)		
V/V 256: \$OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC)		
✓	- / r	<u>File To Be Deleted.txt</u>
	d / d	<u>\$Extend/</u>
	d / d	<u>./</u>
	d / d	<u>Files moved here folder/</u>

Figure 44 Deleted file viewed in Autopsy, marked with the delete flag.

5.1.8 Folder Created

Using Windows Explorer I created a new empty folder on my test device. In this experiment the folder was not populated with any file. I wanted to study the behaviour of an empty folder and its effect on the timestamps. In the image below you can see the empty folder that I created. I followed the same procedure as in my file experiments for this experiment.


 This_is_a_folder	15/03/2024 21:22	File folder
--	------------------	-------------

Figure 45 Empty folder created viewed in Windows Explorer.

After approx. two and a half minutes had passed, I accessed the folder to see what timestamps would be affected by the access operation performed on the empty folder.

5.1.9 Folder Contents Created

I went back to my testing environment and modified the contents of the folder by creating a text file inside the folder. This file was created once again by Notepad, however I added content to the file before saving it. I then analysed the results of the effect creating a file, therefore modifying the folders contents, had on the folders timestamps.

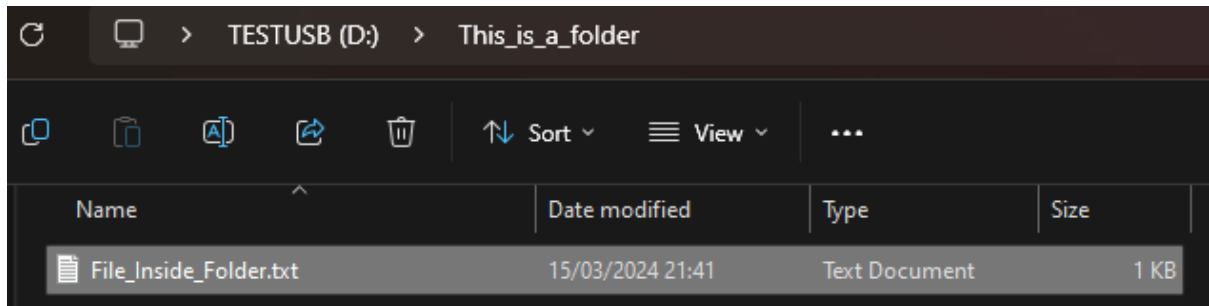


Figure 46 File created inside the folder named "This_is_a_folder".

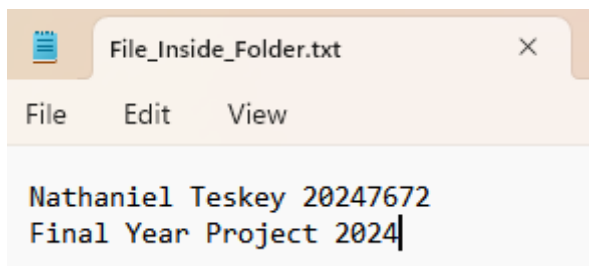


Figure 47 Contents of the file that was created inside the folder.

5.2.0 Basic Timestamp Forgery —

NewFileTime.exe

I created an empty text file using Notepad to test this basic method of timestamping. The file had no content.

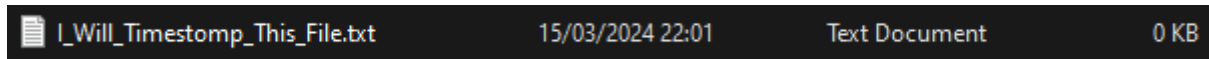


Figure 48 Basic timestamping test file created.

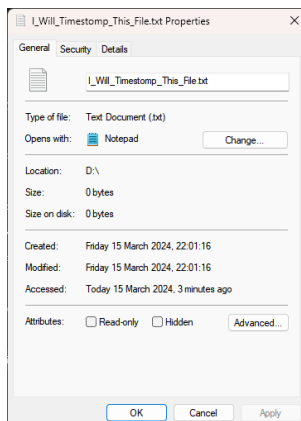


Figure 49 Original timestamps viewed in Windows properties view.

I followed the standard testing procedure for this file to view the timestamps before I went back to the testing environment to timestamp this file with NewFileTime.exe.

After recording the original SIA and FNA timestamps in SIFT Workstation, I proceeded to modify the timestamps with NewFileTime.

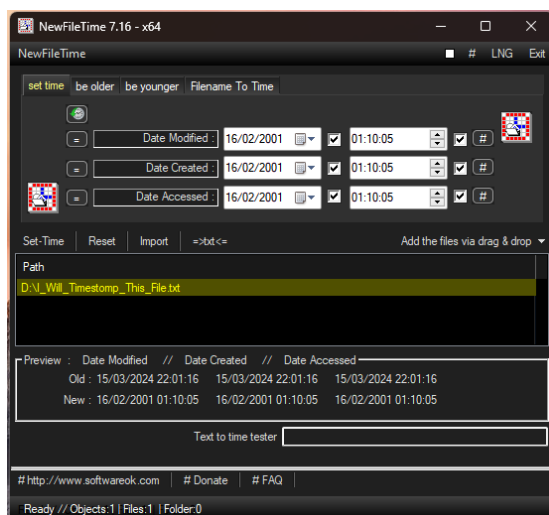


Figure 50 Test file opened in NewFileTime with modified timestamps about to be set.

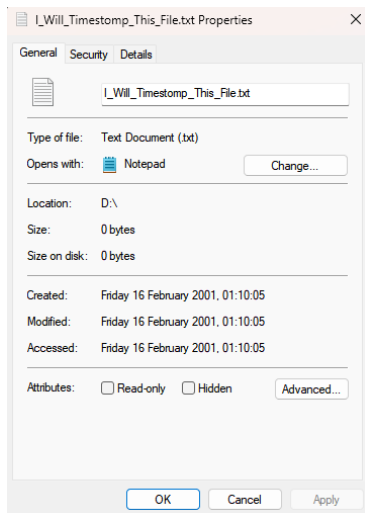


Figure 51 Test file opened in NewFileTime with modified timestamps about to be set.

I ran NewFileTime.exe on the test file again, to reset the SIA accessed timestamp to the forged one – as viewing in Windows Properties view alters this timestamp. I then proceeded with the standard evidence analysis procedure in SIFT Workstation, once again I will outline my findings in the results section.

5.2.1 NewFileTime and File Rename Operation

I was curious to see what the result would be if I renamed the file that had been time stamped. I knew from previous experiments that renaming the file will update some of the FNA attributes. I ran this experiment to see if the forged file times would be copied to the FNA or if they would be reverted to the original true timestamps. This was an interesting experiment as it would take advantage of the Windows kernels method of updating the FNA's along with possibly writing the forged timestamps, applied by NewFileTime.exe to the FNA file times.

```
Name: I_Will_Timestamp_This_File renamed.txt
```

Figure 52 Basic timestamped and then renamed file view in SIFT Workstation using TSK.

The experiment was carried out by simply renaming the text file that had been timestamped by NewFileTime in the previous experiment. I renamed the file through Windows Explorer. I did not access or modify the content of the file and evidence was analysed in the usual method through SIFT Workstation.

5.2.2 Advanced Timestomping with SetMace

I began this experiment by creating the test file (.txt) through Windows Explorer called SetMace_Test.txt. This file had similar text content to my other test files, containing my name and student number.

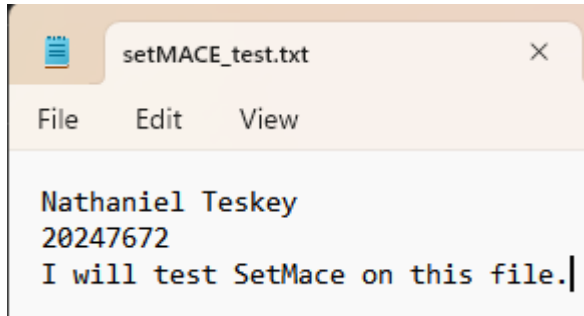


Figure 53 File created and contents modified but not timestomped with SetMace, Notepad view.

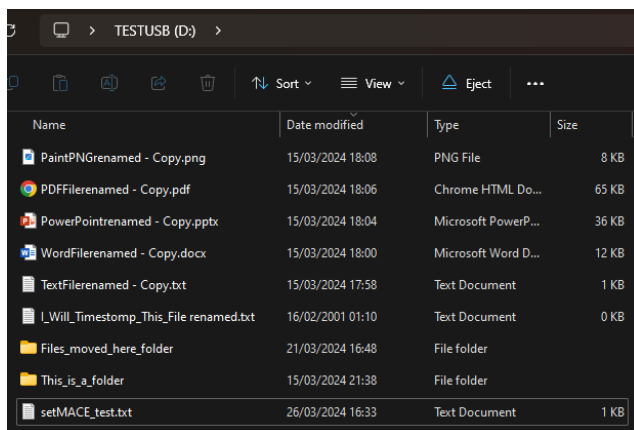


Figure 54 setMace_test.txt test file visible in Windows Explorer on my test device.

Once again, I followed my normal procedure on SIFT Workstation to acquire the evidence and view the original unmodified SIA and FNA timestamps before I ran setMace.exe to modify them.

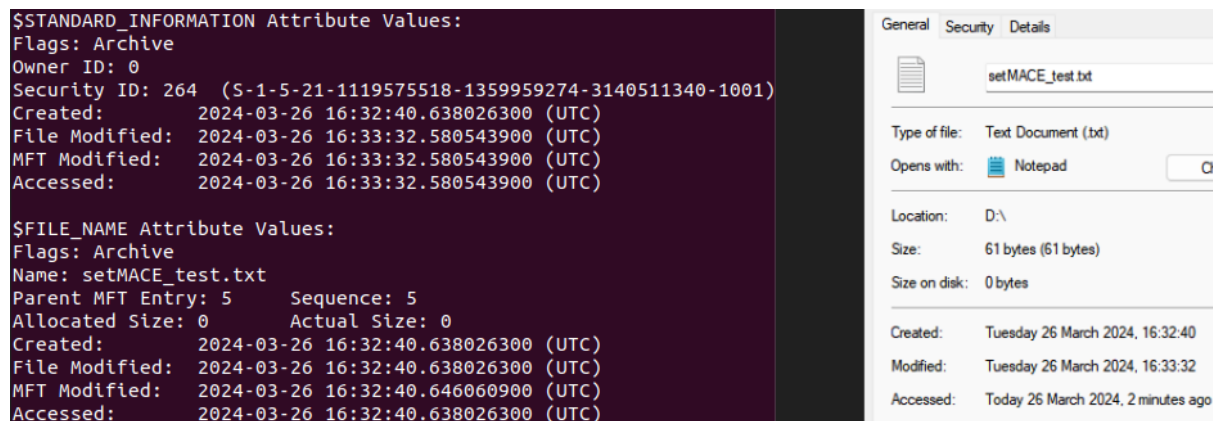


Figure 55 Side by side comparison of TSK's "istat" output of original unmodified timestamps and Windows properties view of the same original and unmodified timestamps.

After acquiring these original timestamps to act as a control for my experiment on SetMace.exe, I then began the procedure to timestamp using this advanced method.

After downloading SetMace, I read through its readme.txt file. I made sure to follow the correct steps to give SetMace the best chance of working on my Windows 11 test environment. I ensure that secure boot was disabled, as a pre-requisite along with downloading and running KPP Destroyer. KPP Destroyer, similar to SetMace is an outdated program that does not list compatibility with Windows 11. KPP Destroyer is required to patch the "ntoskrnl" driver [9], which was implemented by Microsoft after NT 6 to block SetMace from modifying FNA. Support is listed up to Windows 8. My computer restarted and but I saw no sign of the required "KPP Destroyer Boot". I attempted running it a number of times with antivirus disabled but it still would not work.

I then attempted to run SetMace.exe. I opened PowerShell in Administrator mode and navigated to the directory where the SetMace64.exe file was located. I then input the command in the correct format and pressed enter.

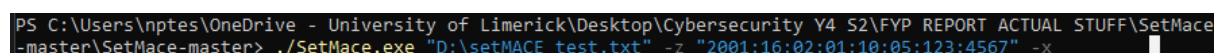


Figure 56 SetMace.exe command in Windows PowerShell Administrator mode.

SetMace appeared to have finished with an exit status of completed as it displayed "Success". The NtLoadDriver error is normal as this was outlined in the programs documentation that it does not use this driver anymore and instead writes to the disk without the need for the driver [9]. As you can see in the image below, SetMace correctly targets the setMace_test.txt file.

```

PS C:\Users\nptes\OneDrive - University of Limerick\Desktop\Cyber
3:4567" -x
Starting SetMace by Joakim Schicht
Version 1.0.0.16

Target filename: setMACE_test.txt
Target fileref: 53
Target MFT record offset: 0x01562400
Parent filename: .
Parent fileref: 5
Parent MFT record offset: 0x01556400

Start patching timestamps

Trying volume offset 0x01562400
Error: NtLoadDriver: 0xC0000428
Error: Could not load driver
Attempting write to physical disk without driver
Success dismounting D:
Success writing timestamps

Patching resident INDX records of parent ($INDEX_ROOT)

Trying volume offset 0x01556400
Warning: The index in $INDEX_ROOT is not resident any more.

Patching non-resident INDX records of parent ($INDEX_ALLOCATION)

Trying volume offset 0x00024000
Error: NtLoadDriver: 0xC0000428
Error: Could not load driver
Attempting write to physical disk without driver
Success dismounting D:
Success writing timestamps

File system cache cleared in RAM

Job took 2 seconds
PS C:\Users\nptes\OneDrive - University of Limerick\Desktop\Cyber

```

Figure 57 SetMace.exe output, appears to be a success. However, it has failed.

Upon inspecting the file in Windows Explorer after the SetMace command had finish running, I noticed that there was no timestamp visible in the Windows Explorer view. This indicated that the command had not worked.



Figure 58 setMace_test.txt viewed in Windows Explorer after SetMace.exe attempt. Note the missing timestamp value here.

Curious as to what the timestamp value would be in SIFT Workstation, I followed my standard procedure to examine the evidence. In the image below the failed timestomping technique is evident. It has set the timestamps for both the SIA and FNA to the same values, approximately 52 years into the future. This informed me that SetMace.exe did in-fact do something to alter the timestamps as the SIA and FNA file times were all updated, but it is not fooling anyone. This is very strange behaviour.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/AdvancedSetMACEExper
$ istat AdvancedSetMaceDONEusb.img 53
MFT Entry Header Values:
Entry: 53          Sequence: 4
$LogFile Sequence Number: 3148722
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created:          2076-11-29 09:01:44.495272900 (UTC)
File Modified:    2076-11-29 09:01:44.495272900 (UTC)
MFT Modified:     2076-11-29 09:01:44.495272900 (UTC)
Accessed:         2076-11-29 09:01:44.495272900 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: setMACE_test.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0       Actual Size: 0
Created:              2076-11-29 09:01:44.495272900 (UTC)
File Modified:        2076-11-29 09:01:44.495272900 (UTC)
MFT Modified:         2076-11-29 09:01:44.495272900 (UTC)
Accessed:             2076-11-29 09:01:44.495272900 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0)  Name: N/A  Resident  size: 72
Type: $FILE_NAME (48-3)  Name: N/A  Resident  size: 98
Type: $DATA (128-1)  Name: N/A  Resident  size: 61
```

Figure 59 SetMace_test.txt viewed in SIFT Workstation using TSK's "istat" command. Notice the strange timestamp values.

5.2.3 SetMace on Windows XP

Frustrated with this result, I approached SetMace from another angle. Using VirtualBox I installed an .iso image of Windows XP. I knew for sure that Windows XP was supported by SetMace as it was listed in the readme.txt file.

Once again, I disabled SecureBoot as required by SetMace [9], however KPP Destroyer was not required on this new VM testing environment as Windows XP runs Windows NT 5.1, which is natively supported by SetMace. I used VirtualBox to allow my test USB device to be mounted to the Windows XP virtual machine, I then connected it and my files were visible on the Windows XP VM.

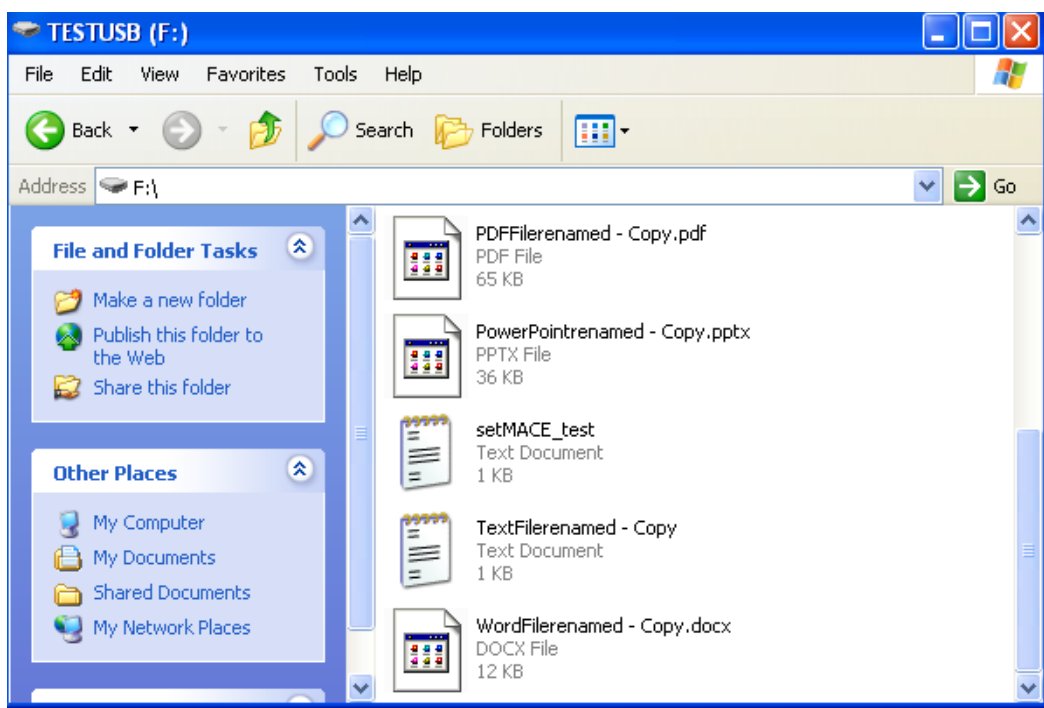


Figure 60 Test device mounted to Windows XP VM and setMace_test file correctly presented.

Once again, I ran the command and then received the terminal output of setMace a few seconds later. This time I got a different error. It mentioned an error in DeviceIoControl which is a driver used by Windows [21].

```

C:\Documents and Settings\Administrator\Desktop\SetMace-master>setmace.exe "F:\setMACE_test.txt" -z "2001:16:02:01:10:05:123:4567" -x
Starting SetMace by Joakim Schicht
Version 1.0.0.16

Target filename: setMACE_test.txt
Target fileref: 53
Target MFT record offset: 0x01562400
Parent filename: .
Parent fileref: 5
Parent MFT record offset: 0x01556400

Start patching timestamps

Trying volume offset 0x01562400
Volume resolved to \\.\PhysicalDrive1
Error in DeviceIoControl: The device is not connected.
Error writing timestamps to disk with driver
Attempting write to physical disk without driver
Success writing timestamps

Patching resident INDEX records of parent (<$INDEX_ROOT>)

Trying volume offset 0x01556400
Warning: The index in $INDEX_ROOT is not resident any more.

Patching non-resident INDEX records of parent (<$INDEX_ALLOCATION>)

Trying volume offset 0x00024000
Volume resolved to \\.\PhysicalDrive1
Error in DeviceIoControl: The device is not connected.
Error writing timestamps to disk with driver
Attempting write to physical disk without driver
Success writing timestamps

File system cache cleared in RAM

Job took 2.79 seconds

```

Figure 61 SetMace.exe input and output on Windows XP VM running in Administrator CMD, featuring DeviceIOController error.

I realised that this error was due to the Windows device input and output controller not working correctly to dismount and remount the test USB device required by SetMace. I attempted this multiple times, even mounting the USB to Windows and then linking it to the VM through a shared folder which was mounted on Windows XP as a network drive. None of my attempts worked.

I knew SetMace had not worked so for this experiment I did not bother to follow the rest of my standard procedure in SIFT Workstation. I viewed the setMace_test.txt file which should have been timestamped by setMace.exe on Windows XP's properties view and the result is shown below.

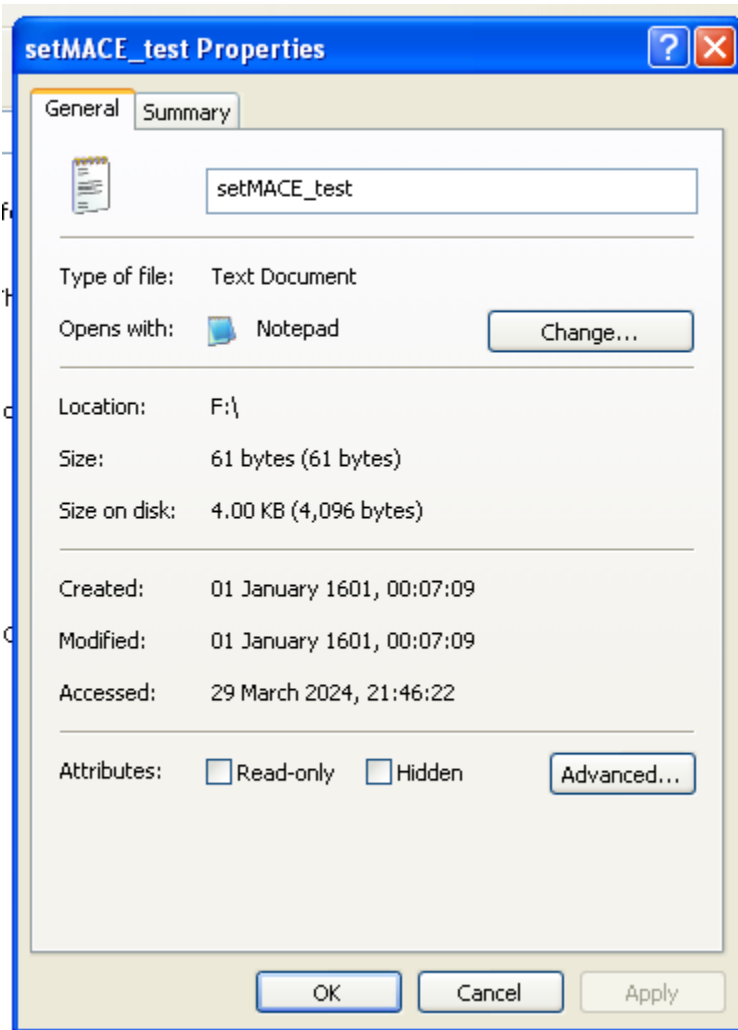


Figure 62 Windows XP Properties View showing failed SetMace attempt due to DeviceIOController error caused by running Windows XP in a VM and having the test device as a USB mounted through VirtualBox.

At this point I ended my attempts to get SetMace to modify the test file on my test device which was formatted by Windows 11 in NTFS. I believe SetMace is outdated and the vulnerabilities it operated on were patched by Microsoft with the release of Windows 11. Using SetMace's functionality on a file within Windows XP, would be out of the scope of this project as I was focused on its efficacy on a current release of Windows 11. I will speak about this more in my results.

Chapter 6 Discussion of Results

I will compare my results to the SANS time rules for Windows 7 [24] to see if they match. If my results match the SANS rules then the NTFS file time rules for Windows 11 have not changed. If the results do not match it is an indication that the NTFS file time rules have been updated on Windows 11.

Windows Time Rules \$STDINFO

SANS COMPUTER FORENSICS and INCIDENT RESPONSE							
File Rename	Local File Move	Volume File Move	File Copy	File Access	File Modify	File Creation	File Deletion
Modified – No Change	Modified – No Change	Modified – No Change	Modified – No Change	Modified – No Change	Modified – Change	Modified – Change	Modified – No Change
Access – No Change	Access – No Change	Access – Changed	Access – Changed	Access – Changed (No Change on Vista/Win7)	Access – No Change	Access – Change	Access – No Change
Creation – No Change	Creation – No Change	Creation – No Change	Creation – Changed	Creation – No Change	Creation – No Change	Creation – Change	Creation – No Change
Metadata – Changed	Metadata – Changed	Metadata – Changed	Metadata – Changed	Metadata – No Change	Metadata – No Change	Metadata – Change	Metadata – No Change

Figure 63 SANS time rules for Windows 7 SIA [24]

Windows Time Rules \$FILENAME

SANS COMPUTER FORENSICS and INCIDENT RESPONSE							
File Rename	Local File Move	Volume File Move	File Copy	File Access	File Modify	File Creation	File Deletion
Modified – No Change	Modified – Updated to \$STDINFO Mod Time	Modified – Changed	Modified – Changed	Modified – No Change	Modified – No Change	Modified – Change	Modified – No Change
Access – No Change	Access – No Change	Access – Changed	Access – Changed	Access – No Change	Access – No Change	Access – Change	Creation – No Change
Creation – No Change	Creation – No Change	Creation – Changed	Creation – Changed	Creation – No Change	Creation – No Change	Creation – Change	Creation – No Change
Metadata – No Change	Metadata – Updated to \$STDINFO Metadata Time	Metadata – Changed	Metadata – Changed	Metadata – No Change	Metadata – No Change	Metadata – Change	Metadata – No Change

Figure 64 SANS time rules for Windows 7 FNA [24]

Upon analysis and comparison of my permitted file operations on Windows 11, I have demonstrated that only the file create operation and the file delete operation match the rules in SANS Institute for Windows 7 file times. This is a clear indicator that NTFS file time rules have been updated somewhere between Windows 7 and Windows 11. Insights and differences visible below.

As shown in the Windows 7 rules from SANS Institute, it states that at every file operation, some values are changed in the SIA and FNA file times. On my experiments the only match were the two operations that produce no “change” in the file times. These are file creation and file deletion.

6.1 File Created (Control)

These results were displayed in my initial experiment, they serve as a control as this was the creation of the files. All timestamps are set to the expected values. These results are in line with the SANS rules above, rules unchanged since Windows 11.

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:14:31.165472300 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: TextFile.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:13:56.390481700 (UTC)
File Modified: 2024-03-15 15:13:56.390481700 (UTC)
MFT Modified:  2024-03-15 15:13:56.390481700 (UTC)
Accessed:      2024-03-15 15:13:56.390481700 (UTC)
```

Figure 65 Text File Created results SIA and FNA

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 15:14:54.146772200 (UTC)
Accessed:      2024-03-15 15:14:42.710238100 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: WordFile.docx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:14:42.710238100 (UTC)
File Modified: 2024-03-15 15:14:42.710238100 (UTC)
MFT Modified:  2024-03-15 15:14:42.710238100 (UTC)
Accessed:      2024-03-15 15:14:42.710238100 (UTC)

```

Figure 66 Word file created results SIA and FNA

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.122964600 (UTC)
Accessed:      2024-03-15 15:20:53.154764800 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PDFFile.pdf
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 28672     Actual Size: 25600
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.091672300 (UTC)
Accessed:      2024-03-15 15:16:57.091672300 (UTC)

```

Figure 67 PDF file created results SIA and FNA

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:31.158764000 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PowerPoint.pptx
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:15:20.429818200 (UTC)
File Modified: 2024-03-15 15:15:20.429818200 (UTC)
MFT Modified:  2024-03-15 15:15:20.429818200 (UTC)
Accessed:      2024-03-15 15:15:20.429818200 (UTC)

```

Figure 68 PPTC file created results SIA and FNA

```

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.736927400 (UTC)
MFT Modified:  2024-03-15 15:16:17.736927400 (UTC)
Accessed:      2024-03-15 15:16:17.736927400 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PaintPNG.png
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 15:16:17.705622300 (UTC)
File Modified: 2024-03-15 15:16:17.705622300 (UTC)
MFT Modified:  2024-03-15 15:16:17.705622300 (UTC)
Accessed:      2024-03-15 15:16:17.705622300 (UTC)

```

Figure 69 Image file created results SIA and FNA

6.2 File Accessed

Overview:

SIA file times experienced updates in File Accessed and or MFT Modified.

FNA file times remained unchanged.

These results do not match SANS rules. Windows 11 NTFS file time rules changed.

Table 1 Timestamp behaviour when file was accessed.

File Type	Behavior on Access (SIA)	Behavior on Access (FNA)
Text File	Unchanged	Unchanged
Word File	File Accessed and MFT Modified updated	Unchanged
PDF File	File Accessed updated	Unchanged
PPTX File	MFT Modified updated	Unchanged
Image File	File Accessed updated	Unchanged

6.3 File Contents Modified

Overview:

SIA file times experienced updates in File Modified, MFT Modified and File Accessed.

FNA file times remained unchanged or else experienced updates in File Modified, MFT Modified and File Accessed

These results do not match the SANS rules. Windows 11 has changed the rules for this operations file times too.

Table 2 Timestamp behaviour when file had contents modified.

File Type	Behavior on Contents Modified(SIA)	Behavior on Contents Modified(FNA)
Text File	File Modified, MFT Modified, and File Accessed updated	Unchanged
Word File	File Modified, MFT Modified, and File Accessed updated	File Modified, MFT Modified, and File Accessed updated
PDF File	File Modified, MFT Modified, and File Accessed updated	Unchanged
PPTX File	File Modified, MFT Modified, and File Accessed updated	File Modified, MFT Modified, and File Accessed updated
Image File	File Modified, MFT Modified, and File Accessed updated	Unchanged

6.4 File Renamed

Overview:

SIA file times always experienced updates in MFT Modified and File Accessed.

FNA file times always experienced updates in File Accessed.

All files displayed the exact same behaviour upon rename operation.

The experiment results do not match SANS rules. Windows 11 file time rules have also been changed for this operation.

Table 3 Timestamp behaviour when file was renamed.

File Type	Behavior on Rename (SIA)	Behavior on Rename (FNA)
Text File	MFT Modified and File Accessed updated	File Accessed updated
Word File	MFT Modified and File Accessed updated	File Accessed updated
PDF File	MFT Modified and File Accessed updated	File Accessed updated
PPTX File	MFT Modified and File Accessed updated	File Accessed updated
Image File	MFT Modified and File Accessed updated	File Accessed updated

6.5 File Copied (Original Files)

Overview:

SIA file times always experienced updates in File Accessed only.

FNA file times always remained unchanged.

All files displayed the same behaviour upon copy operation results do not match SANS rules.

Table 4 Timestamp behaviour on original file when file it was copied.

File Type	Behavior on Copy (Original) (SIA)	Behavior on Copy (Original) (FNA)
Text File	File Accessed was updated	Unchanged
Word File	File Accessed was updated	Unchanged
PDF File	File Accessed was updated	Unchanged
PPTX File	File Accessed was updated	Unchanged
Image File	File Accessed was updated	Unchanged

6.6 File Copied (Destination Files)

Overview:

SIA file times always experienced updates in File Created and File Accessed only.

FNA file times always had all file times updated.

All files displayed the exact same behaviour upon copy to destination operation.

Results do not match SANS rules.

Table 5 Timestamp behaviour on destination file when file it was copied.

File Type	Behavior on Copy (Destination) (SIA)	Behavior on Copy (Destination) (FNA)
Text File	File Created and File Accessed were updated	All file times were updated
Word File	File Created and File Accessed were updated	All file times were updated
PDF File	File Created and File Accessed were updated	All file times were updated
PPTX File	File Created and File Accessed were updated	All file times were updated
Image File	File Created and File Accessed were updated	All file times were updated

6.7 File Moved

Overview:

SIA file times always experienced updates in MFT Modified and File Accessed.

FNA file times always experienced updates in File Accessed.

All files displayed the exact same behaviour upon move operation.

Identical timestamp behaviour to rename operation, results do not match SANS Rules.

Table 6 Timestamp behaviour when file was moved.

File Type	Behavior on Move (SIA)	Behavior on Move (FNA)
Text File	MFT Modified and File Accessed updated	File Accessed updated
Word File	MFT Modified and File Accessed updated	File Accessed updated
PDF File	MFT Modified and File Accessed updated	File Accessed updated
PPTX File	MFT Modified and File Accessed updated	File Accessed updated
Image File	MFT Modified and File Accessed updated	File Accessed updated

6.8 File Deleted

Deleting a file had no impact on the timestamps at all. Neither FNA nor SIA file times changed. The files MFT entry did update to show it was marked for deletion. Also notice how the allocated space for the file is 0, the file is unallocated. These results do match SANS rules.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/FileDeletedExperiments
$ istat -f ntfs FileDeletedUSB.img 53
MFT Entry Header Values:
Entry: 53      Sequence: 3
$LogFile Sequence Number: 1053664
Not Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created:      2024-03-22 14:24:22.173107100 (UTC)
File Modified: 2024-03-22 14:24:58.420196300 (UTC)
MFT Modified:  2024-03-22 14:24:58.420196300 (UTC)
Accessed:      2024-03-22 14:24:58.420196300 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: File_To_Be_Deleted.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0      Actual Size: 0
Created:      2024-03-22 14:24:22.173107100 (UTC)
File Modified: 2024-03-22 14:24:22.173107100 (UTC)
MFT Modified:  2024-03-22 14:24:22.173107100 (UTC)
Accessed:      2024-03-22 14:24:22.173107100 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 110
Type: $DATA (128-1) Name: N/A Resident size: 50
```

Figure 70 TSK's istat command confirming that no timestamps were changed after deletion

```
-/r * 53-128-1: File_To_Be_Deleted.txt
```

Figure 71 TSK's "fls" command showing "-" and "*" signifying the file has been deleted.

```
Pointed to by file:
C:/File_To_Be_Deleted.txt (deleted)

File Type (Recovered):
ASCII text, with CRLF line terminators
MD5 of recovered content:
cad74d76da1048d9108c16c096d466f9 -

SHA-1 of recovered content:
225e73c67cf9d09b04a0b0392dd9681c5d302af8 -

Details:
MFT Entry Header Values:
Entry: 53 Sequence: 3
$LogFile Sequence Number: 1053664
Not Allocated File
Links: 1
```

Figure 72 Autopsy output showing file marked as deleted.

53	Good	Inactive	File	3	5	5/File_To_Be_Deleted.txt
----	------	----------	------	---	---	--------------------------

Figure 73 Parsed MFT entry shows file status as inactive.

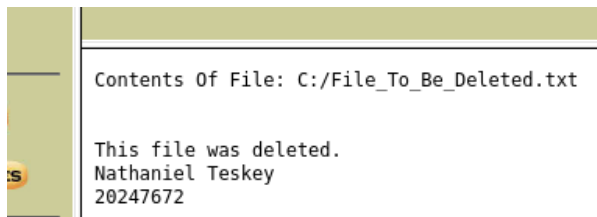


Figure 74 Autopsy showing contents of file after deletion, file recovery possible.

6.9 Folder Timestamps

I created an empty folder initially and recorded the timestamps. I then created a file inside the folder and added contents to it 2 minutes later. SIA file times of the folder experienced updates in File Modified, MFT Modified and File Accessed when the file inside the folder was created. FNA file times remained unchanged throughout. Adding content to the file inside the folder did not cause the folders timestamps to change.

```
sansforensics@workstation: ~/Desktop/Evidence/FolderBehaviourExperiments
$ istat -f ntfs FolderCreated.img 49
MFT Entry Header Values:
Entry: 49          Sequence: 2
$LogFile Sequence Number: 1055123
Allocated Directory
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags:
Owner ID: 0
Security ID: 266 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created: 2024-03-15 21:22:35.638090100 (UTC)
File Modified: 2024-03-15 21:22:35.638090100 (UTC)
MFT Modified: 2024-03-15 21:23:02.563723300 (UTC)
Accessed: 2024-03-15 21:25:06.977050900 (UTC)

$FILE_NAME Attribute Values:
Flags: Directory
Name: This_is_a_folder
Parent MFT Entry: 5          Sequence: 5
Allocated Size: 0          Actual Size: 0
Created: 2024-03-15 21:22:35.638090100 (UTC)
File Modified: 2024-03-15 21:22:35.638090100 (UTC)
MFT Modified: 2024-03-15 21:22:35.638090100 (UTC)
Accessed: 2024-03-15 21:22:35.638090100 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 98
Type: $INDEX_ROOT (144-1) Name: $I30 Resident size: 48
```

Figure 75 TSK's "istat" command displaying initial file times before the folders content was modified.

```
$ istat -f ntfs FileInFolderPopulated.img 49
MFT Entry Header Values:
Entry: 49          Sequence: 2
$LogFile Sequence Number: 1052247
Allocated Directory
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags:
Owner ID: 0
Security ID: 266 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created: 2024-03-15 21:22:35.638090100 (UTC)
File Modified: 2024-03-15 21:38:48.130866100 (UTC)
MFT Modified: 2024-03-15 21:38:48.130866100 (UTC)
Accessed: 2024-03-15 21:38:48.130866100 (UTC)

$FILE_NAME Attribute Values:
Flags: Directory
Name: This_is_a_folder
Parent MFT Entry: 5          Sequence: 5
Allocated Size: 0          Actual Size: 0
Created: 2024-03-15 21:22:35.638090100 (UTC)
File Modified: 2024-03-15 21:22:35.638090100 (UTC)
MFT Modified: 2024-03-15 21:22:35.638090100 (UTC)
Accessed: 2024-03-15 21:22:35.638090100 (UTC)
```

Figure 76 TSK's "istat" command displaying file times after the folders content was modified.

6.9.1 Basic Method: NewFileTime

Timestamping with NewFileTime changes the timestamps displayed in the Windows Properties view. Inside SIFT Workstation the SIA and FNA file times reflected this after timestamping occurred.

SIA file times for File Created, File Modified and File Accessed were set at the timestamped time. This correlates to what was shown in the Windows Properties view. SIA file time for MFT Modified still retained the original timestamp. In line with expectations, FNA file times also held the original time stamps and were not affected by NewFileTime's forgery.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/BasicNewFileTimeExperiments/AfterTimestomp
$ istat -f ntfs BasicTimestompedUSB.img 51
MFT Entry Header Values:
Entry: 51          Sequence: 1
$LogFile Sequence Number: 1053070
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created:      2001-02-16 01:10:05.000000000 (UTC)
File Modified: 2001-02-16 01:10:05.000000000 (UTC)
MFT Modified:  2024-03-15 23:00:33.995211400 (UTC)
Accessed:      2001-02-16 01:10:05.000000000 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: I_Will_Timestomp_This_File.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:      2024-03-15 22:01:16.119746200 (UTC)
File Modified: 2024-03-15 22:01:16.119746200 (UTC)
MFT Modified:  2024-03-15 22:01:16.119746200 (UTC)
Accessed:      2024-03-15 22:01:16.119746200 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 126
Type: $DATA (128-1) Name: N/A Resident size: 0
```

Figure 77 TSK's "istat" command displaying forged SIA file times.

In the image above it is obvious that timestamping has taken place as the timestamped files only have a granularity of "seconds", milliseconds and nanoseconds read 0's as NewFileTime cannot modify these.

6.9.2 NewFileTime and Rename Combination

When the file was renamed after timestomping had taken place there was a huge difference in the results. SIA file times remain the same as when NewFileTime alone was used however, the FNA file times were affected now also.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/BasicNewFileTimeExperiments/testingIfRenamingFileNew
!!TimestompFN
$ istat -f ntfs BasicTimestompedRenameUSB.ing 51
MFT Entry Header Values:
Entry: 51      Sequence: 1
$LogFile Sequence Number: 1051990
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created:      2001-02-16 01:10:05.000000000 (UTC)
File Modified: 2001-02-16 01:10:05.000000000 (UTC)
MFT Modified: 2024-03-15 23:14:55.897193200 (UTC)
Accessed:     2001-02-16 01:10:05.000000000 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: I Will Timestomp This File renamed.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0      Actual Size: 0
Created:      2001-02-16 01:10:05.000000000 (UTC)
File Modified: 2001-02-16 01:10:05.000000000 (UTC)
MFT Modified: 2024-03-15 23:00:33.995211400 (UTC)
Accessed:     2001-02-16 01:10:05.000000000 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-4) Name: N/A Resident size: 142
Type: $DATA (128-1) Name: N/A Resident size: 0
```

Figure 78 TSK's output for the "istat" command showing forged FNA file times after file renamed.

The FNA had its Created, File Modified and File Accessed timestamps "duplicated" from the respective SIA timestamps. This confirms my thinking that a rename would impact the FNA timestamps. Interestingly FNA MFT Modified still retains its original true timestamp of when the file was created. This is interesting to see, however the forgery is blatant to see.

6.9.3 Advanced Method: SetMace

The results for SetMace are disappointing, the application would not run on my machine therefore I believe Windows 11 has patched the vulnerability that this tool exploited in previous releases of Windows NT.

```
sansforensics@siftworkstation: ~/Desktop/Evidence/AdvancedSetMACEExperiments
$ istat AdvancedSetMaceDONEusb.ing 53
MFT Entry Header Values:
Entry: 53      Sequence: 4
$LogFile Sequence Number: 3148722
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119575518-1359959274-3140511340-1001)
Created:      2076-11-29 09:01:44.495272900 (UTC)
File Modified: 2076-11-29 09:01:44.495272900 (UTC)
MFT Modified: 2076-11-29 09:01:44.495272900 (UTC)
Accessed:     2076-11-29 09:01:44.495272900 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: setMACE_test.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0      Actual Size: 0
Created:      2076-11-29 09:01:44.495272900 (UTC)
File Modified: 2076-11-29 09:01:44.495272900 (UTC)
MFT Modified: 2076-11-29 09:01:44.495272900 (UTC)
Accessed:     2076-11-29 09:01:44.495272900 (UTC)

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 98
Type: $DATA (128-1) Name: N/A Resident size: 61
```

Figure 79 TSK's "istat" command output showing the failed SetMace attempt on Windows 11.

All timestamps within SIA and FNA are set to a date roughly 52 years in the future. The fact that the timestamps were modified proves that some elements of SetMace are still working however the application is fundamentally not fit for purpose. SetMace also failed to set the nanosecond values, showing that the timestamps are clearly forged even if it had somehow managed to set more realistic dates.

Chapter 7 Limitations

When undertaking any project there will be limitations, whether it be hardware, software or time limitations. A challenge for me was deciding on which route to take with the project. I spent a lot of time in the early days trying to figure out if I would develop a tool to retrieve timestamps from a parsed MFT in .csv outfit, I also examined the idea of creating a tool to dive into the NTFS MFT and pull the timestamps directly out of the SIA and FNA. I decided against both of them as the former is a glorified PowerShell script to select data from certain places in a spreadsheet and the latter was too involved for my experience with NTFS at the time, which was close to non-existent. Provided with more time and newly found experience, these would be interesting routes to explore.

My USB test device was a limiting factor in my results. I created a smaller partition within the USB for testing. This initially seemed like a good idea but it caused me to be dealing with an NTFS partition and not an entire NTFS volume. This resulted in me trying to use TSK's "mmls" command to view the partition table. I now know this is impossible because I was inside a partition. I also faced issues when trying to export the \$LogFile and \$UsnJrnl, I believe this is due to the same issue that I was operating on a partition and not a storage volume. In the end I found out a great deal regarding the operation of timestamps and I do not believe it held me back, but, it would have been nice to view an uncorrupted \$LogFile with logfile entries present.

The reason for splitting the partitions into two was for two main reasons. The calculation of a raw image on a 32GB USB takes a very long time. The "dd" command that is used to take a raw image of the evidence or test data also takes a very long time to run. I reduced the size so I could carry out my experiments back to back and in relatively quick time, as opposed to leaving my laptop run over night to generate one raw image and one md5 hash of the data. This is just not feasible. A computer with a more powerful CPU and graphics card would be better equipped to handle these resource intensive activities. In professional digital forensics this is not a limiting factor due to the money being invested to keep data secure.

Initially I had installed a Windows 11 virtual machine as my test environment, I quickly realised this was a bad idea as the system files take up a large amount of space, ~30GB and the minimum disk size on Windows 11 is 80GB. I scrapped this idea and tried an alternative method, this time I found a lightweight version of Windows 11, called Tiny11 Core [23]. This allowed for windows to be installed on just 6GB of a disk, with the file system taking up 2GB. I began to read the documentation on it and realised the reason it was so lightweight was because a lot of the functionality had been stripped out, there were even certain NTFS drivers missing from it. This may have hampered my results, but it also would not be fair on my experiments as I planned to carry out all of my experiments on a current and official release of Windows 11. Needless to say Tiny 11 Core, although small and mighty, is nowhere near an official version of Windows.

Even though it did not impact my results, I was missing the ability to put a write blocker on my test USB drive when I took my evidence to SIFT Workstation. Even though SIFT Workstation has a built-in software write blocker, using it would not have provided any additional safety. Even though I did not compromise any evidence, using a write blocker would have been a nice touch – sticking to true digital forensics procedures. A physical write blocker would be even better but it was really unnecessary for my test data and they also are expensive. The reason that using a write blocker built into SIFT Workstation would not have made a difference is because the USB drive was automatically mounted at “/dev/sdb1” by VirtualBox. I tried to figure out a way to manually mount the USB myself but then I realised I was just wasting time and decided to get on with my experiments instead.

Finally, I could have presented my results better. I was caught between displaying the results tidily and concisely but then I was worried in case my results would not come across as strong as they would have been missing the evidence supporting my claims. In the end I decided to provide a lot of detail on my test procedure and results, however I know it could have been displayed in more efficient way – perhaps using graphs or charts. I tried to think of a graph or chart to display the timestamp update data for SIA and FNA nicely, but it was not something I implemented.

Chapter 8 Conclusions and Future Work

The tools used such as TSK and Autopsy proved very beneficial in analysing the test data. The test USB device did a good job of fitting the requirement of the experiments. It allowed me to iterate through tests reasonably quickly which would not have been possible with a larger disk size.

The file operations and experiments I chose gave a large amount of test data all with various independent outcomes. After analysing the results of permitted file operations on various common file types I gained a deep understanding into the NTFS file time rules. Comparing the Windows 7 rules with the current Windows 11 behaviour from my experiments, shows starkly different results. This is indicative of a change in Microsoft file time decision making process. Timestamp rules have definitely been changed between Windows 7 and Windows 11 as the only operations that gave a similar result were the experiments that made no change to the file times at all, namely created and deleted operations.

Experiments into the basic method of timestomping with NewFileTime.exe went as expected. I would not use this tool to actually try to hide file times though, it is very obvious due to the trailing zeros.

Testing my experiment using NewFileTime in conjunction with a file rename operation proved very beneficial. This is not very practical in a normal environment; this experiment was more of a proof of concept than an actual timestomping guide. It is obvious as FNA MFT Modified holds the original birth timestamp after all steps are completed.

I was very eager to display my timestomping results with SetMace, at the time I did not realise it was compatible with Windows 11, but that is the benefit of experiments. No matter the outcome something is always learned. I planned to trace back the SetMace forgery through the \$LogFile and then get the time that the timestamp was modified. Next, I would have gone to the location where the terminal stores a record of the commands that it has executed, expanding on DOSKEY history [22] . Matching the \$LogFile time to the Windows CMD or PowerShell execution time could give a good understanding into the methodology behind the use of SetMace by a user. This could be carried out as future work.

Creation of a script to extract timestamp data from a parsed MFT could also be considered as future work. If the script could organise the file times then it would save lots of writing and it would also speed up the process of analysing file time changes.

Running the tests on a VM could also be an interesting idea, a user could examine SetMace's code and see where it is failing and possibly attempt to make a new version of KPP Destroyer to disable Windows security driver - although I am unsure of the ethical considerations of this.

References

[1]

“5 Free Software to Bulk Change File Timestamp in Windows,” I Love Free Software, Nov. 01, 2017. <https://www.ilovefreesoftware.com/01/featured/free-software-to-bulk-change-file-timestamp-windows.html> (accessed Apr. 02, 2024).

[2]

J. Gerend, “NTFS overview,” learn.microsoft.com, Dec. 23, 2021. <https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview>

[3]

“Understanding File Systems,” Kingston Technology Company. [https://www.kingston.com/en/blog/personal-storage/understanding-file-systems#:~:text=NTFS%20\(New%20Technology%20File%20System](https://www.kingston.com/en/blog/personal-storage/understanding-file-systems#:~:text=NTFS%20(New%20Technology%20File%20System) (accessed Apr. 02, 2024).

[4]

J. Bouma, H. Jonker, Vincent, and E. Van, “Reconstructing Timelines: From NTFS Timestamps to File Histories,” Aug. 2023, doi: <https://doi.org/10.1145/3600160.3605027>.

[5]

“The Sleuth Kit: Documents,” www.sleuthkit.org. <https://www.sleuthkit.org/sleuthkit/docs.php>

[6]

“SANS Digital Forensics and Incident Response Blog | NTFS: Attributes Part One | SANS Institute,” www.sans.org. <https://www.sans.org/blog/ntfs-attributes-part-one/>

[7]

D. Palmbach and F. Breitingner, “Artifacts for Detecting Timestamp Manipulation in NTFS on Windows and Their Reliability,” Forensic Science International: Digital Investigation, vol. 32, p. 300920, Apr. 2020, doi: <https://doi.org/10.1016/j.fsidi.2020.300920>.

[8]

“NewFileTime 7.16 Corrections and manipulation of timestamp,” www.softwareok.com. <https://www.softwareok.com/?seite=Microsoft/NewFileTime> (accessed Apr. 02, 2024).

[9]

J. Schicht, “jschicht/SetMace,” GitHub, Jan. 06, 2024. <https://github.com/jschicht/SetMace> (accessed Apr. 02, 2024).

[10]

“What Is NTFS and How Does It Work?,” [www.datto.com](https://www.datto.com/blog/what-is-ntfs-and-how-does-it-work#:~:text=NT%20file%20system%20(NTFS)%2C). [https://www.datto.com/blog/what-is-ntfs-and-how-does-it-work#:~:text=NT%20file%20system%20\(NTFS\)%2C](https://www.datto.com/blog/what-is-ntfs-and-how-does-it-work#:~:text=NT%20file%20system%20(NTFS)%2C)

[11]

“NTFS Basics,” [Ntfs.com](https://www.ntfs.com/ntfs_basics.html), 2019. https://www.ntfs.com/ntfs_basics.html

[12]

Anhansen, “Resilient File System (ReFS) overview,” [learn.microsoft.com](https://learn.microsoft.com/en-us/windows-server/storage/refs/refs-overview). <https://learn.microsoft.com/en-us/windows-server/storage/refs/refs-overview>

[13]

B. Carrier, File system forensic analysis. Boston, Mass. ; London: Addison-Wesley, 2005.

[14]

“Defragment your Windows 10 PC,” [support.microsoft.com](https://support.microsoft.com/en-us/windows/defragment-your-windows-10-pc-048aefac-7f1f-4632-d48a-9700c4ec702a). <https://support.microsoft.com/en-us/windows/defragment-your-windows-10-pc-048aefac-7f1f-4632-d48a-9700c4ec702a>

[15]

“How to Defrag your Hard Drive,” [Crucial](https://eu.crucial.com/articles/pc-users/how-to-defrag-hard-drive). <https://eu.crucial.com/articles/pc-users/how-to-defrag-hard-drive> (accessed Apr. 02, 2024).

[16]

alvinashcraft, “Master File Table (Local File Systems) - Win32 apps,” [learn.microsoft.com](https://learn.microsoft.com/en-us/windows/win32/fileio/master-file-table), 2021. <https://learn.microsoft.com/en-us/windows/win32/fileio/master-file-table>

[17]

REDMOND\markl, “Clock Skew,” [learn.microsoft.com](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/adrms_sdk/clock-skew), May 31, 2018. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/adrms_sdk/clock-skew (accessed Apr. 02, 2024).

[18]

A. C. Group and L.-A. Jaju, “Ethical Digital Forensics - Balancing Investigation Procedures With Privacy Concerns,” [Lexology](https://www.lexology.com/library/detail.aspx?g=fb018971-9604-405b-a13c-2ec2e3c19fcc#:~:text=One%20of%20the%20main%20ethical), Apr. 07, 2023. <https://www.lexology.com/library/detail.aspx?g=fb018971-9604-405b-a13c-2ec2e3c19fcc#:~:text=One%20of%20the%20main%20ethical>

[19]

Eric, “EricZimmerman/WxTCmd,” [GitHub](https://github.com/EricZimmerman/WxTCmd), Sep. 28, 2023. <https://github.com/EricZimmerman/WxTCmd> (accessed Apr. 02, 2024).

[20]

“Cheat Engine :: View topic - KPP Destroyer (Patchguard Disabler),” www.cheatengine.org.
<https://www.cheatengine.org/forum/viewtopic.php?p=5537540&sid=cb2f435167ccd0c1af5ffceaead64eb2> (accessed Apr. 02, 2024).

[21]

GrantMeStrength, “DeviceIoControl function (ioapiset.h) - Win32 apps,” learn.microsoft.com, Jul. 27, 2022. <https://learn.microsoft.com/en-us/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol> (accessed Apr. 02, 2024).

[22]

R. M. Stevens and E. Casey, “Extracting Windows command line details from physical memory,” *Digital Investigation*, vol. 7, pp. S57–S63, Aug. 2010, doi: <https://doi.org/10.1016/j.diin.2010.05.008>.

[23]

“Tiny11 Core Cuts Down ISO to 2GB & Installation to 3GB on a VM,” MiniTool, Nov. 03, 2023. <https://www.minitool.com/news/tiny11-core.html> (accessed Apr. 02, 2024).

[24]

“SANS Digital Forensics and Incident Response Blog | Windows 7 MFT Entry Timestamp Properties | SANS Institute,” www.sans.org. <https://www.sans.org/blog/windows-7-mft-entry-timestamp-properties/> (accessed Apr. 02, 2024).

Appendices

```
sansforensics@siftworkstation: /media/sansforensics  
$ sudo dd if=/dev/sdb1 of=~/Desktop/Evidence/FileCreatedExperiments/FileCreatedUSB.img  
131072+0 records in  
131072+0 records out  
67108864 bytes (67 MB, 64 MiB) copied, 0.925367 s, 72.5 MB/s
```

Figure 80 "dd" command to take raw image and output

Appendix A: Updated Project Gantt Chart

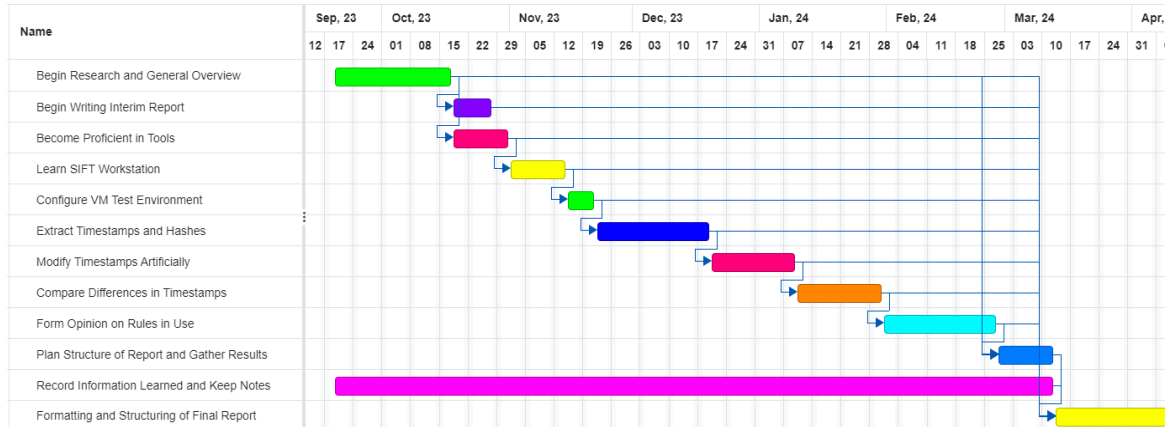
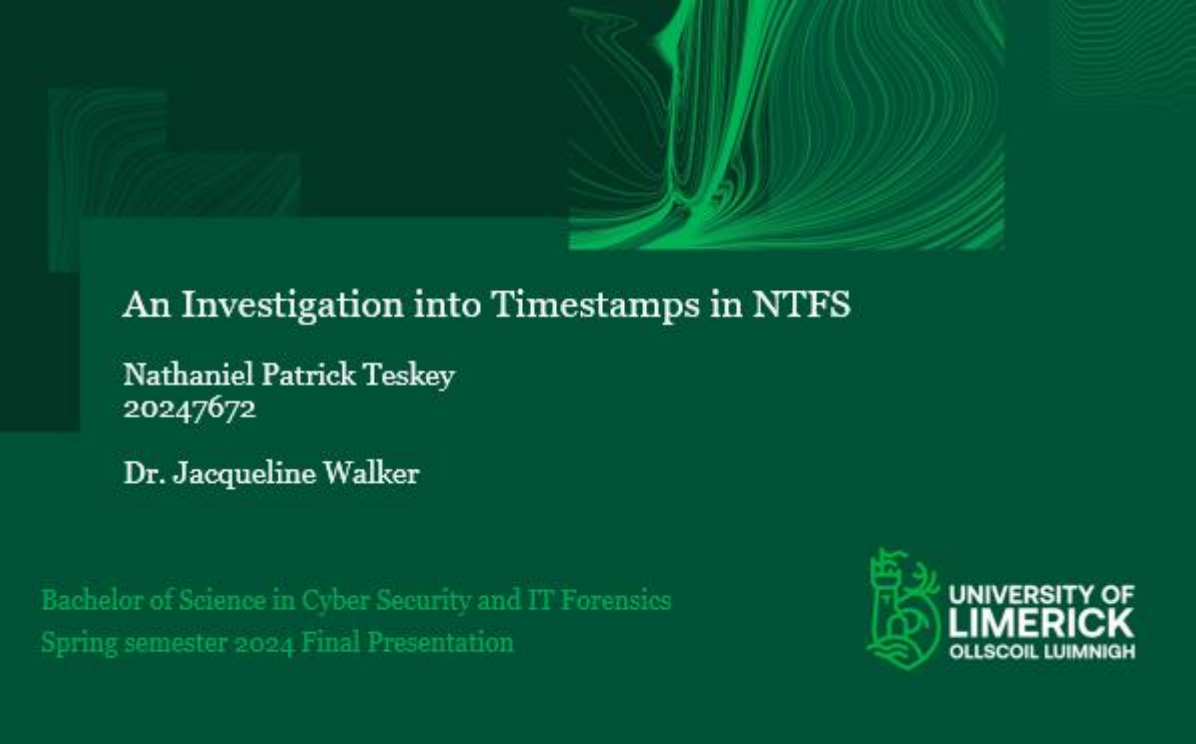


Figure 81 Updated Project Gantt Chart

Appendix B: Final Presentation Slides




An Investigation into Timestamps in NTFS

Nathaniel Patrick Teskey
20247672

Dr. Jacqueline Walker

Bachelor of Science in Cyber Security and IT Forensics
Spring semester 2024 Final Presentation



Presentation overview

- File Created
- File Accessed
- File Content Changed
- File Renamed
- File Copied (source and dest)
- File Moved
- File Deleted
- .txt
- .docx
- .pdf (PD24 Creator)
- .pptx
- .png
- Folder Behaviour, file inside folder then file populated outcome on timestamps.



Project overview (1)

- SIA (Win Properties)
 - Created
 - Modified
 - MFT Modified
 - Accessed
- FNA (Kernel)
 - Created
 - Modified
 - MFT Modified
 - Accessed

```
$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 264  ()
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.122964600 (UTC)
Accessed:      2024-03-15 15:20:53.154764800 (UTC)

$FILE_NAME Attribute Values:
Flags: Archive
Name: PDFFile.pdf
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 28672    Actual Size: 25600
Created:      2024-03-15 15:16:57.075977900 (UTC)
File Modified: 2024-03-15 15:16:57.091672300 (UTC)
MFT Modified:  2024-03-15 15:16:57.091672300 (UTC)
Accessed:      2024-03-15 15:16:57.091672300 (UTC)
```



SetMace
Error

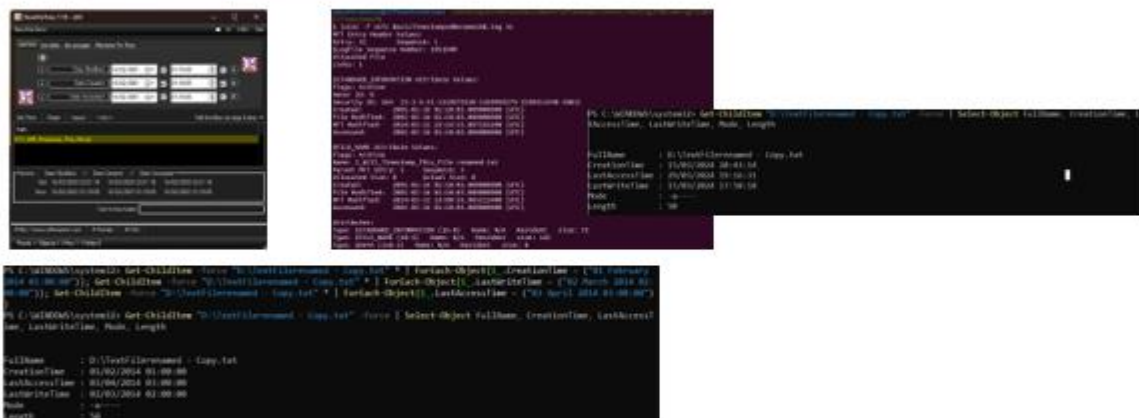
```
C:\Users\opres@redrive - University of Liverpool\Desktop>ls
lsstat AdvancedInterfaceOOMush.lng 53
NFT Entry Header Values:
Entry: 53 Sequence: 4
BlobFile Sequence Number: 3048722
Allocated File
links: 1

SSANDARD_ATTRIBUTE_VALUES:
Flags: Archive
Owner ID: 0
Security ID: 264 (S-1-5-21-1119075108-1359092774-354011340-1006)
Created: 2076-11-29 09:01:44.495272900 (UTC)
File Modified: 2076-11-29 09:01:44.495272900 (UTC)
NFT Modified: 2076-11-29 09:01:44.495272900 (UTC)
Accessed: 2076-11-29 09:01:44.495272900 (UTC)

SFFILE_NAME Attribute Values:
Flags: Archive
Name: setNACE test.txt
Parent NFT Entry: 5 Sequence: 5
Allocated Size: 0 Actual Size: 0
Created: 2076-11-29 09:01:44.495272900 (UTC)
File Modified: 2076-11-29 09:01:44.495272900 (UTC)
NFT Modified: 2076-11-29 09:01:44.495272900 (UTC)
Accessed: 2076-11-29 09:01:44.495272900 (UTC)

Attributes:
Type: STANDARD_INFORMATION (16-B) Name: N/A Resident size: 72
Type: SFFILE_NAME (48-3) Name: N/A Resident size: 90
Type: SHA1 (120-I) Name: N/A Resident size: 65
```

Project implementation



Key results

- Definite change in timestamp rules Win 7 > Win 10
- Timestamp results are not predictable but there are patterns such as MFT Modified when MFT entry changed due to rename, file moved etc.
- NewFileTime and rename can update \$File_Name Attribute timestamps by renaming after timestamping.
- SetMace is patched on Windows 11.
 - Compatible with 6.1 using KPP destroyer to disable Windows security driver.
 - Not compatible with Windows 11, last update 2014.
 - Windows 11 is running Windows NT version 10, far beyond 6.1.



Problems and solutions

- MD5 hash taking too long to generate and "dd" command taking too long.
 - Calculations / raw image copy takes a very long time on VM with 80GB disk.
- Created "test USB" as alternative.
 - Format NTFS
 - Host device Windows 11
 - Test partition size 64MB = ok
 - Problem is that it was a partition and not an entire NTFS volume which limited some experiments regarding \$LogFile and couldn't view partition table.
- Experimented with Windows 11 VM 80GB Disk, Tiny11 Core 2GB Disk and Windows XP (setMace)



Potential uses of the system and future work

- Forensic analysis or recovery of suspected forged timestamps
- Gathering evidence on file times to reinforce statement being brought forward
- Reconstruction of timeline of events on a Windows device
- Create script to pull timestamps in correct format from a parsed MFT .csv file
- Run same procedure that I undertook but on a more power system.
 - Take image of entire Windows 11 machine
 - Calculate hashes on it
 - Explore more metadata files like \$LogFile





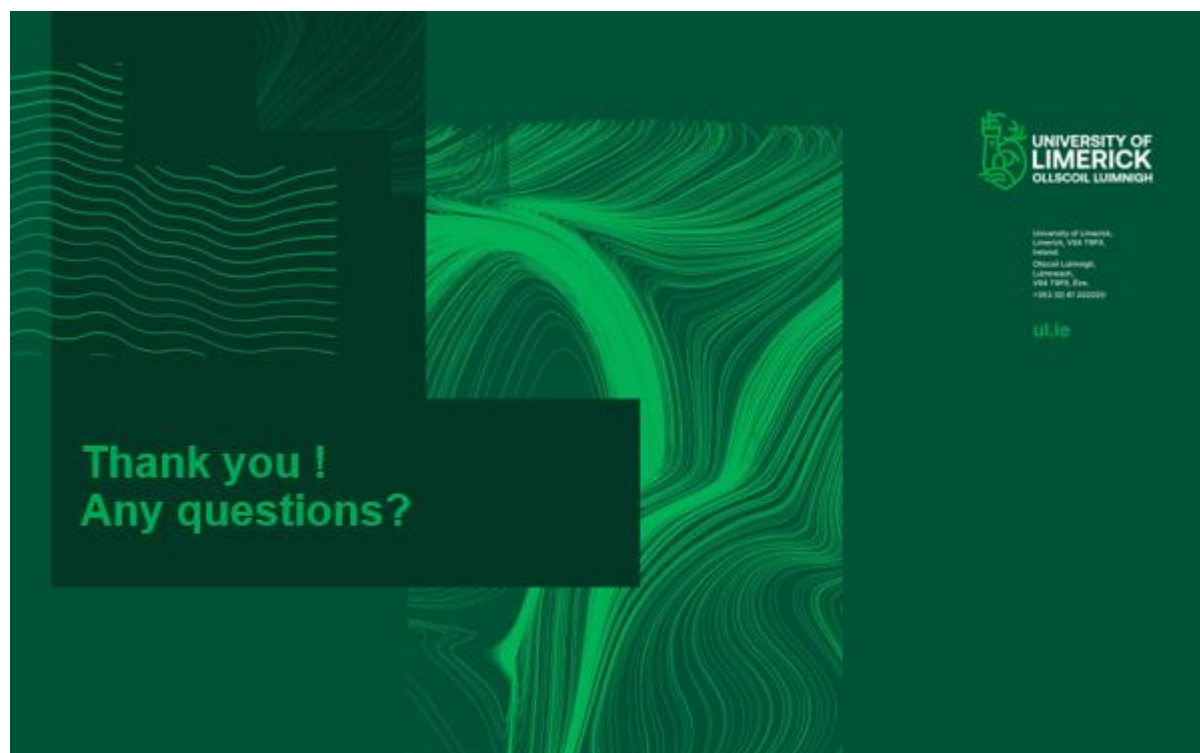
Demonstration : Evidence Acquisition in SIFT

- 1. SIFT Workstation is launched from VirtualBox.
- 2. TESTUSB is mounted to SIFT Workstation.
- 3. MD5 hash of mounted TESTUSB is calculated and saved to file.
- 4. Raw image is taken of TESTUSB using the "dd" command.
- 5. MD5 hash of raw image is calculated and saved to file.
- 6. Hashes are compared ensuring the evidence has not been altered during extraction.
- 7. TSK's "fls" command is run to view MFT entry number.
- 8. TSK's "istat" command is run to view MFT entry of test file.
- 9. Check for changes in SIA and FNA file time values.
- 10. Record and screenshot results.




Conclusions


- Good general understand of NTFS gained.
- Deeper understanding of timestamps gained.
- Knowledge on SetMace, KPP Destroyer, NewFileTime, anti-forensic tools.
- Better understanding of Linux, especially SIFT Workstation and tools.
- Really challenging project by also enjoyable when results weren't as expected.
- Hardest part was gaining an overview on how to tackle the timestamp
 - ie: Experiment setup, testing environment etc.



Appendix C: Project Poster



An Investigation into Timestamps in NTFS

**Department of
Electronic and
Computer Engineering**

Nathaniel Patrick Teskey
BSc. Cybersecurity and IT Forensics

Introduction

NTFS has been the foundation of Windows since the launch of the outcomes of operations Windows NT. I carried out experiments into common filetypes such as .txt, .docx, .pptx, .pdf, and .png, by recording such as file creation, access, modify, rename, copy, move and delete on the timestamp attributes. Following a strict experimental setup was essential to retain the integrity and accuracy of my results, to ensure this I calculated cryptographic hashes of my evidence to show that it was not inadvertently modified.

I then took a dive into the world of "timestomping", which is the anti-forensic technique of artificially modifying timestamps. I used both basic and more advanced methods to investigate such as NewFileTime and setMACE.


Aims

- Develop a solid understanding of timestamps within the NTFS filesystem.
- Investigate the behaviour of timestamps in relation to various file operations and filetypes.
- Utilize forensic tools to extract and analyse the evidence, adhering to best forensic practices.
- Examine the impact of both basic and advanced anti-forensic "timestomping" tools and evaluate their efficacy.
- Provide insight, based on my experiments, outlining the rules that impact timestamp behaviour on Windows 11.

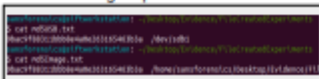
Method

Experiments were set up on a USB drive inserted to a laptop running version 10.0.22631 of Windows 11, while analysis was carried out through a Linux Virtual Machine running SIFT Workstation.

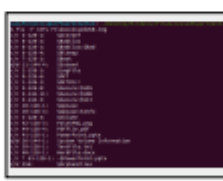
- **Create test files** on USB drive using Windows file explorer or other application.




- Perform a permitted **file operation** or **timestomping** on one test file at a time.
- Launch **SIFT Workstation** virtual machine and **mount** USB drive.
- Calculate **MD5 hash** of USB drive.
- Take **raw image** of USB drive for analysis.
- Calculate **MD5 hash** of raw image.
- **Compare hashes** ensuring they are **identical**, proving evidence has not been modified during acquisition.



- Use `fls` command on raw image to obtain the **\$MFT Entry address** of the test file.
- Using the `intact` command, specify the \$MFT Entry address to **view** the test file's **\$MFT Entry**.
- Record the **changes** in the timestamps within the **\$STANDARD_INFORMATION** and **\$FILE_NAME** attributes.



Output of `fls` command showing `TextFile.txt` at \$MFT Entry address 39.




Graphical user interface of a popular but basic anti-forensic timestomping tool, known as **NewFileTime**.

Results

Here I will explain one of my experiments results that I found to be quite interesting – see report for analysis of all results.

When carrying out experiments regarding timestomping with **NewFileTime**, the documentation provided with the application stated that it was only able to modify the **\$STANDARD_INFORMATION** timestamp. This is in line with information available online, stating that the **\$FILE_NAME** timestamp cannot be modified by anti-forensic tools as it is updated only by the Windows kernel. As you will see in my report – I have found this to be untrue, as it is possible to update the **\$FILE_NAME** timestamps even using a basic method like **NewFileTime**.

Following the method outlined previously, I ran **NewFileTime**. It worked as expected and updated the **\$STANDARD_INFORMATION** timestamps which are shown in Windows' Properties view.



Windows 11 Properties view of the file that had its timestamps modified. It is highly suspicious as **Created**, **Modified** and **Accessed** timestamps all display the same time of **01:10:05 on 16/02/01**.

Conclusion and Reflection


I really enjoyed undertaking this investigation into NTFS timestamps as it was a challenging topic that I knew very little about this time last year! I had to do background research into NTFS, then learn how to use tools such as The Sleuth Kit and SIFT Workstation and create a plan of file operations, file types, anti – forensic tools and correct forensic procedures to undertake insightful experiments.

In hindsight, I would have used a physical write blocker to further preserve the integrity of the evidence on the USB drive – however it is not absolutely necessary as the MD5 hashes were always a match.

In my report I have explained the results of all the experiments that I undertook. I also gave my own opinion on the reasons for timestamps being updated in a specific way on Windows.

Acknowledgements

I want to extend my sincere thanks to my supervisor Dr. Jacqueline Walker for her valuable advice and encouragement. I was delighted to see that this project had been provided as an FYP option and for that I am extremely grateful.

**UNIVERSITY OF
LIMERICK**
OLLSCOIL LUIMNIGH