Polytechnic of Leiria
School of Technology and Management
Computer Engineering Department
Master in Cybersecurity and Digital Forensics

ANALYSIS OF TIMESTAMP MANIPULATION DETECTION TOOLS

LUÍS PAULO MONTEIRO DOS SANTOS

Leiria, 2024

Polytechnic of Leiria
School of Technology and Management
Computer Engineering Department
Master in Cybersecurity and Digital Forensics

ANALYSIS OF TIMESTAMP MANIPULATION DETECTION TOOLS

LUÍS PAULO MONTEIRO DOS SANTOS
2220749

Projeto realizado sob orientação de:
Professor Doutor Miguel Cerdeira Marreiros Negrão
Professor Doutor Miguel Monteiro de Sousa Frade
Professor Doutor Patrício Rodrigues Domingues

Leiria, 2024

ACKNOWLEDGMENTS

I would like to thank everyone who contributed to the successful completion of this project.

A special thanks to my thesis advisors for their unwavering support, guidance, and the valuable knowledge they have shared throughout the development of this work. I am especially grateful to Professor Miguel Cerdeira Marreiros Negrão, who introduced me to programming, Professor Patrício Rodrigues Domingues, whose classes were my favorite throughout both my degree and master, and Professor Miguel Monteiro de Sousa Frade, who sparked my interest in the field of digital forensics.

Finally, I would like to express my deepest gratitude to my family, girlfriend and friends for their constant love, support, and encouragement throughout this journey and for help me find balance in life.

RESUMO

A deteção da manipulação de selos temporais em sistemas de ficheiros NTFS tem sido historicamente desafiante, em que as primeiras ferramentas para o efeito produziam resultados pouco fiáveis em cenários reais. Os métodos anteriores, comforme destacado pelos autores Oh et al., muitas vezes sofiram de limitações, como o número de falsos positivos, identificando erroneamente eventos normais do sistema de ficheiros como manipulação, ou sendo incapazes de detetar alterações intencionais nos selos temporais dos ficheiros. Dois exemplos são o NTFS Log Tracker 1.71 e a ferramenta Timestamp Analyser, que falharam na deteção dessas manipulações, e a última que gerou um elevado número de falsos positivos. No entanto, avanços recentes, como o lançamento do NTFS Log Tracker 1.9 em maio de 2024, demonstraram uma maior precisão. Esta ferramenta, conforme detalhado no artigo "Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation," integra vários algoritmos de deteção forense de alteração de selos temporais, retirando informação da $MFT, do $LogFile e do $UsnJrnl, juntamente aproveitando as entradas relacionadas com os artefactos do sistema como o Windows Prefetch e ficheiros LNK. Estas melhorias visam detetar mais eficazmente a manipulação dos selos temporais dos ficheiros em investigações digitais forenses. Este projeto explora estes avanços e fornece informações atualizadas sobre os efeitos das operações de ficheiros no selos temporais num sistema New Technology File System (NTFS).

ABSTRACT

Detecting timestamp manipulation on NTFS file systems has historically been challenging, with early tools producing unreliable results in real-world scenarios. Previous methods, as highlighted by Oh et al., often suffered from limitations such as generating false positives by misidentifying normal file system events as manipulation or being unable to detect intentional alterations in timestamps.Tools like NTFS Log Tracker v1.71 and TimestampAnalyser struggled to reliably identify such manipulations. However, recent advancements, such as the release of NTFS Log Tracker v1.9 in May 2024, have demonstrated improved accuracy. The updated tool, as detailed in "Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation," integrates multiple forensic detection algorithms by leveraging the $MFT, $LogFile, and $UsnJrnl, along with additional system artifacts like Windows Prefetch and LNK files. These enhancements aim to more effectively detect timestamp manipulation in digital forensic investigations. This project explores these advancements and provides updated information about the file operations effects on NTFS timestamps.

CONTENTS

Acknowledgments i
Resumo iii
Abstract v
Table of Contents vii
List of Figures xi
List of Tables xiii
List of Acronyms xv

1 Introduction 1
1.1 Motivation . . . 1
1.2 Objectives . . . 2
1.3 Contributions . . . 3
1.4 Document structure . . . 3

2 Background 5
2.1 NTFS on Linux . . . 5
2.2 NTFS Characteristics . . . 6
2.2.1 Size . . . 6
2.2.2 B-Tree . . . 6
2.2.3 Names . . . 6
2.2.4 Everything in NTFS is Stored in a File . . . 6
2.2.5 The Entire Logical Volume is a Data Area . . . 7
2.3 NTFS Features . . . 7
2.3.1 Master File Table (MFT) . . . 7
2.3.2 Alternate Data Streams . . . 8
2.3.3 Journaling . . . 9
2.3.4 $LogFile . . . 9
2.3.5 Change Journal - $UsnJrnl . . . 10
2.3.6 $I30 . . . 11
2.3.7 Compression . . . 11
2.3.8 Encryption . . . 11
2.3.9 Access Control List . . . 12
2.3.10 Quotas . . . 12
2.3.11 Cluster remapping . . . 13
2.3.12 File System Tunneling . . . 13
2.4 Timestamps . . . 13
2.4.1 Timestamp analysis . . . 13
2.4.2 NTFS Timestamps . . . 13
2.4.3 Last Access timestamp . . . 14
2.5 Methods to perform timestamp manipulations . . . 15

2.6 nTimestomp . . . 16
3 Literature Revision 17
3.1 File System Forensic Analysis . . . 17
3.2 A computer forensic method for detecting timestamp forgery in NTFS 17
3.3 Time for Truth: Forensic Analysis of NTFS Timestamps . . . 18
3.3.1 Reconstructing Timelines: From NTFS Timestamps to File Histories . 19
3.4 Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation . . . 19

4 Methodology 21
4.1 Setup . . . 21
4.1.1 Tools . . . 21
4.2 Test Cases . . . 23
4.2.1 TC01 - File Creation . . . 23
4.2.2 TC02 - File Access . . . 24
4.2.3 TC03 - File Modification . . . 24
4.2.4 TC04 - File Rename . . . 26
4.2.5 TC05 - File Copy . . . 26
4.2.6 TC06 - File local move . . . 27
4.2.7 TC07 - File Move to Different Volume using Command Line Interface . 28
4.2.8 TC08 - File Move to Different Volume using Windows Explorer . . . 28
4.2.9 TC09 - File Timestamps Manipulation . . . 29
4.2.10 TC10 - File Deletion . . . 30
4.2.11 TC11 - Office File Creation . . . 31
4.2.12 TC12 - Office File Modification . . . 31
4.2.13 TC13 - Convert File to PDF . . . 33
4.2.14 TC14 - File System Tunneling . . . 34
4.3 Data collection . . . 35
5 Results Analysis 37
5.1 Test Cases . . . 37
5.2 Tools Results . . . 40
5.3 Results Analysis . . . 45

6 Conclusions 47
6.1 Future Work . . . 48

Bibliography 49

Appendixes

A Appendix A 55
A.1 Main . . . 55
A.2 Init - Populate Files . . . 56

B Appendix B 59

B.1 Test Cases Results 59
B.2 Timestamp Analyser 63

Declaração 65

LIST OF FIGURES

Figure 2.1 NTFS Logical Layout 7
Figure 2.2 Master File Table (MFT) 15 first entries representation . . . 8
Figure 2.3 Win32 API Filetime structure 14
Figure 2.4 fsutil command displaying the possible values of NtfsDisableLastAccessUpdate 15
Figure 4.1 Test Case TC01 in PowerAutomate 24
Figure 4.2 Test Case TC02 in PowerAutomate 25
Figure 4.3 Test Case TC03 in PowerAutomate 25
Figure 4.4 Test Case TC04 in PowerAutomate 26
Figure 4.5 Test Case TC05 in PowerAutomate 27
Figure 4.6 Test Case TC06 in PowerAutomate 28
Figure 4.7 Test Case TC07 in PowerAutomate 28
Figure 4.8 Test Case TC08 in PowerAutomate 29
Figure 4.9 Test Case TC09 in PowerAutomate 30
Figure 4.10 Test Case TC10 in PowerAutomate 30
Figure 4.11 Test Case TC11-A in PowerAutomate, Excel file creation . . . 31
Figure 4.12 Test Case TC11-B in PowerAutomate, PowerPoint file creation 32
Figure 4.13 Test Case TC11-C in PowerAutomate, Word file creation . . . 32
Figure 4.14 Test Case TC12-A in PowerAutomate, Excel file modification 33
Figure 4.15 Test Case TC12-B in PowerAutomate, PowerPoint file modification 34
Figure 4.16 Test Case TC12-C in PowerAutomate, Word file modification 35
Figure 4.17 Test Case TC13-A in PowerAutomate, Word save as PDF . . . 35
Figure 4.18 Test Case TC13-B in PowerAutomate, Microsoft Print to PDF 36
Figure 4.19 Test Case TC14 in PowerAutomate 36
Figure 5.1 NTFS Log Tracker v1.71 main window 42
Figure 5.2 NTFS Log Tracker Suspicious Behavior Results 42
Figure 5.3 NTFS Log Tracker v1.9 main window 43
Figure 5.4 NTFS Log Tracker v1.9 suspicious Behavior Detections tab . 44
Figure 5.5 NTFS Log Tracker v1.9 detection of nTimestomp 44
Figure 5.6 NTFS Log Tracker v1.9 displaying the event found related with nTimestomp 44
Figure A.1 Main flow of the testing environment 56
Figure A.2 Initial population of files to the testing folder 57

LIST OF TABLES

Table 2.1 MACE Timestamps description 14
Table 4.1 Software used and respective versions. 22
Table 5.1 Test Cases Results table without TC11 and TC12. 37
Table 5.2 Test Cases 11 and 12 results table. 41
Table B.1 TC01_creation.txt $STANDARD_INFORMATION timestamps. . 59
Table B.2 TC02_access.txt $STANDARD_INFORMATION timestamps. . . 59
Table B.3 TC03_modified.txt $STANDARD_INFORMATION timestamps. 59
Table B.4 TC04 files $STANDARD_INFORMATION timestamps. 59
Table B.5 TC05 files $STANDARD_INFORMATION timestamps. 60
Table B.6 TC06_local_move.txt $STANDARD_INFORMATION timestamps. 60
Table B.7 TC07_volume_file_move_cli.txt $STANDARD_INFORMATION timestamps. 60
Table B.8 TC08_volume_file_move_explorer.txt $STANDARD_INFORMATION timestamps. 60
Table B.9 TC09_timestamp_manipulation.txt $STANDARD_INFORMATION timestamps. 60
Table B.10 TC10_file_deletion.txt $STANDARD_INFORMATION timestamps. 61
Table B.11 TC11-A, TC11_excel.xlsx $STANDARD_INFORMATION timestamps. 61
Table B.12 TC11-B, TC11_powerpoint.pptx $STANDARD_INFORMATION timestamps. 61
Table B.13 TC11-C, TC11_word.docx $STANDARD_INFORMATION timestamps. 61
Table B.14 TC12-A files $STANDARD_INFORMATION timestamps. 61
Table B.15 TC12-B files $STANDARD_INFORMATION timestamps. 62
Table B.16 TC12-C files $STANDARD_INFORMATION timestamps. 62
Table B.17 TC13-A files $STANDARD_INFORMATION timestamps. 62
Table B.18 TC13-B files $STANDARD_INFORMATION timestamps. 62
Table B.19 TC14_file_system_tunneling.txt $STANDARD_INFORMATION timestamps. 62

LIST OF ACRONYMS

ADS	Alternate Data Stream.
API	Application Programming Interface.
APT	Advanced Persistent Threats.
DACL	Discretionary Access Control List.
DDF	Data Decryption Fields.
DRF	Data Recovery Fields.
EFS	Encrypting File System.
FAT	File Allocation Table.
FEK	File Encryption Key.
HPFS	High Performance File System.
LSASS	Local Security Authority.
LSN	Logical Sequence Number.
MFT	Master File Table.
NTFS	New Technology File System.
OS	Operative System.
SID	Security Identifier.
URL	Uniform Resource Locator.
USN	Update Sequence Number.
UTC	Coordinated Universal Time.

1 INTRODUCTION

1.1 *Motivation*

A high usage of digital devices is observed every day, from the simplest IoT device to the supercomputers used to calculate meteorologic data. Whenever there is a need to explain any digital events related with these devices, whether resulting from an incident or a crime investigation, a digital investigation is crucial to understand what happened.

In the realm of digital forensics, the ability to accurately trace and understand the timeline of digital artifacts plays a critical role in criminal investigations, cybercrime analysis, legal proceedings, or a simpler non-crime related analysis. According to (Carrier, 2005), a digital investigation is a process where we develop and test hypotheses that answers questions about digital events. Also, it is known that forensic scientists rely on accurate timestamps in order to reconstruct the timeline of digital events of the filesystem under investigation (Galhuber and Luh, 2021). These timestamps provide temporal information related with file's creation and modifications, and their precision rely on the filesystem and the Operative System (OS) themselves.

In a investigation on a computer with the Windows OS installed, NTFS will most likely be the filesystem used, since it is the default one, not only in end-user devices, but also in servers running this OS. However, despite their utility, NTFS timestamps presents several challenges to forensic investigators, allied to the dynamic nature of timestamp updates and their potential manipulation, introducing uncertainty and complexity into the interpretation of timestamp data.

This complexity has been further exacerbated by the increasing sophistication of cyber threats who are able to maliciously modify timestamps, thus obfuscating their activities and complicating forensic efforts.

File timestamp manipulation is divided into two main types, one that relies in using an existing timestamp manipulation tool or a function backdoor to change the timestamps of a file previously created. In the other type, the manipulation is achieved using malware that uses a time manipulation API or Powershell to modify the timestamp of a file (Galhuber and Luh, 2024; Oh et al., 2024). Currently, SetFileTime() and NtSetInformationFile() are the API available to perform the manipulations on the timestamps, and there is the function Get-Item cmdlet on Powershell. Although there is another way to perform timestamp manipulation, using a direct disk access method,is less reliable since the disk cannot be the system drive, as it will be described further in the next chapter. Real examples file timestamp manipulation includes advanced persistent threat (APT) 17, APT19,

APT21, APT28, APT29, APT30, APT37, APT38, APT40, Dark Hotel, Kimsuky and Winnti (Oh et al., 2024).

With, the appearance of threats performing timestamp manipulations, there was a need to develop tools to detect this behaviors, to detect and confirm, for example, whether or not a proof in court was adulterated, or to find the origin of a cyber attack. Initially, timestamp manipulation detection tools were ineffective on real scenarios, as stated by Oh et al. "previous NTFS journal-based detection methods have limitations such as incorrectly identifying normal events as manipulation or detecting manipulation only in limited cases", meaning that they would generate a large number of false positives, or were unable to detect files with timestamps intentionally modified. Two examples of these tools are the `NTFS Log Tracker v1.71`, resulting from the paper “NTFS Data Tracker: Tracking file data history based on $LogFile” (Oh et al., 2021) and `TimestampAnalyser` from “Reconstructing Timelines: From NTFS Timestamps to File Histories” (Bouma et al., 2023). As of May 2024, `NTFS Log Tracker v1.9` was published along with the paper “Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation” (Oh et al., 2024). This time the authors mention that the results of their paper "are expected to help investigators detect file timestamp manipulation during the real-world digital forensic investigation", which will be studied ahead. Initiatives like SANS publishing "Windows Time Rules", mapped the way each file operation affect the timestamps, although, this information has to be revised, since each new Windows version can change some file operation effect on the file timestamps.

1.2 *Objectives*

This project aims to investigate the forensic relevance of NTFS timestamps, focusing on understanding and testing modern timestamp manipulation detection tools, as well as verifying the current state of timestamp update due to file operations and the development of an automated scenario to test them. The objectives of this study were:

1. Summarizing the current state of timestamp manipulation and detection tools.
2. Testing three tools against a close to a real world scenario and verifying their results.
3. Manipulating timestamps from a file and testing if the tools detect it.
4. Providing up-to-date documentation of how the operations effect the file’s timestamps.
5. Designing test cases to evaluate the file operations effect on timestamps.
6. Developing a test scenario, using the test cases above.
7. Designing a SANS like poster[1].

1 https://www.sans.org/posters/windows-forensic-analysis/

8. Delivering newer information related with Microsoft Office applications files timestamp behavior.

1.3 *Contributions*

To our knowledge, no other study has been published that tests these tools or provides the list of file operations in the OS version used.

The contributions of this study include:

- A SANS like poster, with the Windows 11 v22H2 compared with Windows 11 v23H2 timestamp operations effect.
- Nineteen test cases to evaluate the file operations effect on timestamps, and the effectiveness of timestamp manipulation detection tools.
- An automated testing mechanism, in PowerAutomate, that can be reused in the future to assist testing newer version of the tools, or newer versions of the OS, using the designed test cases.

1.4 *Document structure*

This project is strutured as follows:

- Starting with chapter 1, it introduces the topic, the project motivations and objectives.
- In the second chapter (2), background knowledge of file systems, in particular NTFS , and lastly, timestamp related information.
- The third chapter (3) focus on the literature review, containing five relevant sources of information used to start this project, as well as a brief explanation and insights of the contents of each source.
- Next, the chapter 4 outlines the the research methodology, including the experimental setup, tools used, test cases and the data collection techniques.
- Then, in the fifth chapter (5) the results and discussion, present the results of the experiment, analyzing the behavior of NTFS timestamps, and tools evaluating the difference.
- Finally, the conclusion (6) summarizes the findings, highlighting the contributions to the field and suggests avenues for future work.

2

BACKGROUND

File systems were created in order to solve a trivial problem; computers needed some way to handle the long-term storage and retrieval of data. Therefore, file systems provide a mechanism for users to store data in a hierarchy of files and directories, determining how files, folders, links, and the data are named, stored, and organized, such that the computer knows where to find them.

NTFS is a proprietary journaling file system and was first introduced in 1993. NTFS (Version 1.0) was introduced in Windows NT 3.1 and, at the time of writing, this filesystem remains the default file system used in the latest version of the Windows Operating System, Windows 11 (23H2) using version 3.1 of NTFS (Hermon et al., 2023; Microsoft, 2024d).

NTFS was introduced to improve reliability, which is especially desirable for high end systems and file servers, to overcome the limitations of the File Allocation Table (FAT) and High Performance File System (HPFS) file systems, in particular increasing the maximum size of files and volumes , and finally to support POSIX requirements. The file systems FAT and HPFS did not support the Portable Operating system Interface (POSIX) requirements unlike NTFS which is the most POSIX.1 compliant of the Windows supported file systems, compliant with the following POSIX.1 requirements: case-sensitive naming, file last accessed timestamp and hard links (Microsoft, 2024c). Additionally, the system was designed to be extensible.

It is considered a recoverable file system since it keeps track of transactions against the file system, storing them in a transaction log called "$LogFile". In a power failure or system crash, the operating system can roll back the changes or continue where it left (Elkhatib, 2022). In this file system, metadata files' name are prefixed with `$`, such as the `$MFT` and the `$LogFile` which will be described later.

2.1 *NTFS on Linux*

Since the early 2000's, several NTFS drivers including the ntfs3 legacy driver (Torvalds, 2023) and Captive (Kratochvil, 2023) have been included in the Linux kernel, providing alternative ways to access Windows' default file system on Linux. Other possible way to use NTFS in Linux relies on FUSE-based drivers such as ntfs-3g which run in user mode (Tuxera, 2023).

2.2 *NTFS Characteristics*

2.2.1 *Size*

NTFS has a theoretical file size limit of of 2^{64} - 1 KB bytes (16 Exabytes[1]) (Microsoft, 2023h). However, the maximum implemented size for files and volumes currently is 2^{32} - 2048KB (maximum cluster size) , totaling a maximum size of 8PB[2], since Windows Server 2019 and newer, and also Windows 10 version 1709 and newer (older versions support up to 256 TB cluster size of 64 KB) (Microsoft, 2023g). Note that the default cluster size is 4 KB (allowing up to 16 TB volumes). It is discouraged to use NTFS on a volume that is smaller than 400 MB.

2.2.2 *B-Tree*

In order to keep track of where the data is stored on disk, NTFS relies on a tree, specifically a B-tree (Carrier, 2005), which differs from a binary tree since it can hold more than two children per node. Each entry in the tree uses a data structure called an index entry, storing the values in each node. These index entries can differ from each other, but their header fields are always the same. Carrier (2005) gives the following example, "a directory index entry contains a few header values and a $FILE_NAME attribute. The index entries are organized into nodes of the tree and stored in a list. An empty entry is used to signal the end of the list". These nodes are referred as *inodes* (index nodes), and for a directory inode, we can find one or more $FILE_NAME index entries (Karresand, 2023).

2.2.3 *Names*

NTFS names use 16 bit Unicode and allow a maximum file name length (including folders) of 255 Unicode characters. Some characters such as \, /, :, *, ?, ", <, >, | and the UTF-16 enconded Unicode character with the value `0x0000`, NULL, are not allowed. The maximum file path size is 32,760 Unicode characters, but each path component cannot exceed the 255 characters limit (Microsoft, 2023a).

2.2.4 *Everything in NTFS is Stored in a File*

One core concept in the design of NTFS is that important data are allocated to files, including administrative data that are typically hidden by other file systems. These files that contain the administrative data can be located anywhere in the volume, like a normal file can.

1 18,446,744,073,709,550,592 bytes
2 9,007,199,252,643,840 bytes

2.2.5 *The Entire Logical Volume is a Data Area*

In NTFS the entire file system is considered a data area, and any sector can be allocated to a file, in contrast with FAT which has two distinct areas, system and data area. The layout found in a NTFS formatted volume is presented in figure 2.1.

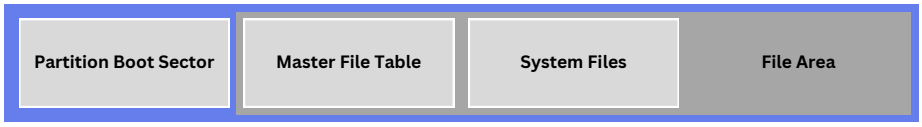

Figure 2.1: NTFS Logical Layout

According to Carrier (2005), the first sectors of the volume contain the boot sector and boot code (Partition Boot Sector). Afterwards, the files are stored in the File Area, starting by the $MFT, the system files and then every other file (including the users files).

2.3 *NTFS Features*

2.3.1 *Master File Table (MFT)*

NTFS contains a file denominated as the MFT, or $MFT and is considered a core file of this file system, containing the information about all files and directories. There is at least one entry for every file and directory present on the volume in the MFT, and in fact the first entry in the table is named $MFT and it describes the on-disk location of the MFT (Carrier, 2005; Microsoft, 2023f).
A visual representation of a MFT is shown by Figure 2.2.

All MFT records contain information related with the corresponding file/directory, which includes its size, timestamps, permissions, and content. An entry can be either fully stored on the MFT, or reference a space outside the MFT. The first 26 records in MFT are metadata files and each MFT record can be either 1 or 4 KiB in size (Carrier, 2005; Karresand, 2023).

Each table entry has a header, located in the first 56 Bytes, and afterwards, several attributes can be assigned, such as $STANDARD_INFORMATION ($SI) which is 96 bytes long followed by the $FILE_NAME ($FN) attribute which stores the file name with 66 bytes allocated and a variable offset from the 66^{th} byte until the maximum allowed number of characters, 255^{3} (Cho, 2019). The $STANDARD_INFORMATION attribute contains temporal (time and date stamps), ownership, security and quota information. This attribute also contains flags with general properties of the file status, such as read only, archive and system. It should be noted that, as Carrier (2005) states "Nothing in this attribute is essential to storing files, but many of the application-level features that Microsoft provides

3 Currently, from the command line, you can only create files containing names up to 253 characters (Microsoft, 2023h).

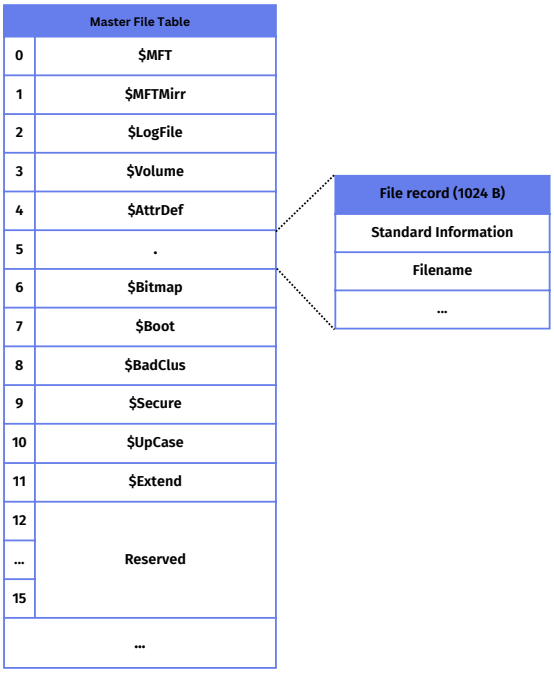

Figure 2.2: MFT 15 first entries representation

depend on it". Regarding the temporal information, this attribute contains four timestamps, which will be described further in section 2.4.2.

Every file and directory has, at least, one $FILE_NAME attribute in its $MFT entry. The $FILE_NAME attribute contains: file reference of parent directory, file creation time, file modification time, MFT modification time, file access time, allocated size of file, real size of file, flags, reparse value, length of name, namespace and name. Both $STANDARD_INFORMATION and $FILE_NAME are resident in the MFT.

Another important attribute is $Data which will be discussed further ahead in the subsection 2.3.2. MFT entry is the signature, and a standard entry will have the

2.3.2 *Alternate Data Streams*

Every file on an NTFS volume contains at least one stream, called the default stream and each stream is stored in an attribute of type $DATA. Carrier (2005) states that "When a file has more than one $DATA attribute, the additional attributes are sometimes referred to as Alternate Data Stream (ADS)". These streams can contain metadata attributes such as title, date etc (Hermon et al., 2023). In addition, if this content is larger than approximately 700 bytes in size, it becomes non-resident and is saved in external clusters. One example of ADS usage is Windows Internet Explorer using the stream named Zone.Identifier, to store Uniform Resource Locator (URL) security zones. This information is accessible through `<file name>.<extention>:Zone.Identifier:$DATA` (Microsoft, 2024a).

2.3.3 *Journaling*

Journaling is described by Cho (2019) as "an advanced file system integrity feature, which is employed in virtually all modern file systems", where each operation (called transaction) is made atomic. As files, directories, and other NTFS objects are added, deleted and modified, NTFS enters the changes into records (Microsoft, 2024b). Therefore, journaling employs the process of logging changes before performing them, so that, when a system failure, crash, or power outage occurs, "the scanning program reads the journal and locates the entries that were not completed. The program then either completes the changes or rolls them back to the original state" (Carrier, 2005). After a bad-sector error, the file system remaps the cluster that contains the bad sector, and allocates a new cluster for the data. It also marks the original cluster as bad, and no longer uses the old cluster. For example, after a server crash, NTFS can recover data by checking its log files and performing a consistency check (Microsoft, 2023g).

As represented above in the Figure 2.2, the third entry on the MFT is called $LogFile which is one of the journal files in the NTFS. These journals are relevant to find evidence of, for example, file creations, deletions, renames, and updates, and it should be noted that these artifacts can persist after anti-forensics tools have been used.

2.3.4 *$LogFile*

This log journal file is located in the third MFT entry, named $LogFile, and its default size is 65535 bytes. This entry does not have any special attributes and logged data is stored in the $DATA attribute (Carrier, 2005). Logs stored in this file have a Logical Sequence Number (LSN), which is a unique 64-bit value and is incremented in each record. Since this file has a fixed size, when there is no more space to store records at the end of the file, the oldest record is overwritten, making it a circular file, and in this scenario the records at the beginning of the file will have their LSN higher than the records at the end of the file. This file has areas that serve different purposes, such as the restart area, and the logging area. The logging area is also divided into the buffer page area and the normal page area (Oh et al., 2021).

- Restart Area: contains information on the last record, also referred as last operation and current operation, and resides in the first two consecutive pages of the $Logfile, where the second page is used as a backup. These pages begin with the magic number "RSTR", and both pages contain the LSN of the last record and the LFS (Log File Service) version.
- Logging Area: holds the operation records and the first two pages (0x2000 and 0x3000) are the Buffer page area, and the remainder of the space is the Normal page area.

- Buffer Page Area: buffer that holds the record until it is fully logged in the Normal Page area, in its first page(0x2000), and second page is used for backup purpose (0x3000).
- Normal Page Area: logging area, that holds the records sequentially and is circular, meaning that, when it is full, it will overwrite from the page 0x4000 with new records.

Each update record has two major fields in addition to its LSN value. The *Redo Operation* field contains information related with what will it perform, and the other field, the *Undo Operation* contains the opposite information, and shows how to rollback the previous operation (Carrier, 2005). These operations contain codes related with the operations, for example, AddIndexEntryAllocation, InitializeFileRecordSegment, DeleteIndexEntryAllocation, UpdateResidentValue, UpdateNonResidentValue, CreateAttribute and more.

Forensically, and according to Carrier (2005), this logging file could provide information about recent changes to the file system, but it is not known for how long the entries remain in the logfile before they are overwritten. It is also mentioned that the organization of the file is not known.

Evidences found on this file can be difficult to explain (Carrier, 2005), since the LSN number of a given entry in the MFT could be used to reconstruct the sequence in which the files were edited, although some events of that sequence could be already missing.

These events include:

- Creating a new file or directory
- Changing the content of a file or directory
- Renaming a file or directory
- Changing any of the data stored in the MFT entry of a file or directory (for example, user ID or security settings)

Note that, each event above can represent more than one operation in the $LogFile.

2.3.5 *Change Journal - $UsnJrnl*

The change journal is a file in the root of a volume located in `.\$Extend\$UsnJrnl` that records when changes are made to files and directories, commonly used by applications to determine which file have changed, which makes it a "High Level" journal, when compared to $LogFile. It contains two ADS, $J and $Max. The most relevant ADS is $J, which contains the transactions used in the timestamp manipulation detection tools algorithms, and it is a sparse file, containing a list of different sized data structures, called change journal entries. This journal uses the Update Sequence Number (USN) as the unique ID for each transaction, which is the reason for the name UsnJrnl (Carrier, 2005).

This journal tracks changes to files and directories, and logs its creations, deletion, renaming and data changes. The logged data format includes the time of the change, the file or directory name where the event took place, the event type (Oh et al., 2024). These event types are stores as operations, similar to the $LogFile. Here are some operations examples: FileCreate, FileDelete, RenameOldName, RenameNewName, DataOverWrite, DataExtend, Close, ObejectIdChange and more (Carrier, 2005; Microsoft, 2024b).

2.3.6 *$I30*

The $I30 file also known as Windows NTFS Index Attribute, storing indexes information related with $INDEX_ROOT, $INDEX_ALLOCATION attributes. It arranges file names along with other associated file metadata (including four timestamps) within a directory into a B-tree (balanced tree) form, and allows NTFS to efficiently obtain access to files contained within that particular directory. The timestamps present on this file are under the $FILE_NAME attribute, but these timestamps are not the same present on the $MFT, since it mirrors those that are in $STANDARD_INFORMATION (Carrier, 2005).

This file can be found in each directory, and its forensic value is associated with the presence of artifacts related with files that were deleted and still contain entries in the index file.

2.3.7 *Compression*

Even on early versions of NTFS, it allowed attributes to be written in a compressed format, even though, Microsoft states that only the $DATA attribute should be compressed, and only when it is non-resident. File compressed on an NTFS volume can be read and written by Windows based application without any another program intervention. Decompression occurs automatically when the file is read, and the file is compressed when it is closed or saved. This Compression feature is made entirely at the file system level and NTFS handles the real-time access to a compressed file.

The compression algorithm used in NTFS is Lempel-Ziv (LZ77), and on this file system, is designed to support cluster sizes of up to 4 KB. (Karresand, 2023; LSoft Technologies, 2023b; Microsoft, 2023g). When the cluster size is greater than 4 KB on a NTFS volume, none of the NTFS compression functions are available.

2.3.8 *Encryption*

Data at rest encryption is another feature provided by this file system. It is useful when one wants to protect stored data from unauthorized access. In NTFS this encryption is possible using the feature called Encrypting File System (EFS), and is achieved by setting the flag Encrypted (0x4000) for the file and creates

the $EFS attribute (0x100), where it stores the Data Decryption Fields (DDF) and Data Recovery Fields (DRF) (LSoft Technologies, 2023a). This way, an attacker with physical disk access will have to brute-force the drive in order to decrypt any files or folders in an EFS enabled NTFS volume without the owner's private key (Lo et al., 2023). Users can interact with the EFS features through a command-line utility called `cipher.exe` in Windows. Note that this feature is only possible using Windows kernel EFS driver and FSRTL, in conjuntion with the EFS service, running at the user mode (Microsoft, 2023d).

In NTFS, EFS works by encrypting the $DATA attribute and its contents are encrypted with a symmetric algorithm called DESX (Carrier, 2005). For each MFT entry with encrypted data, one random key is generated, and this key is called File Encryption Key (FEK). When there is multiple $DATA attributes in the MFT entry, they are encrypted with the same FEK. This FEK is store in an encrypted state in the $LOGGED_UTILITY_STREAM attribute.

Users of EFS are issued with a unique digital certificate containing a public and private key and EFS uses these keys for each user logged on to the local machine where the private key is stored (LSoft Technologies, 2023a).

2.3.9 *Access Control List*

Access Control List is another feature implemented in this file system, and provides base security for files and folders: Each user logged onto the system holds an access token with security information for that logon session. This access token is created when the user logs on. Thus, every process executed on behalf of this user has a copy of the access token (*How the System Uses ACLs - NTFS.com* 2024).

The token identifies: the user, the user groups and the user's privileges. It also contains a logon Security Identifier (SID) that identifies the current logon session.

When a thread tries to access a secured object, this access is granted or denied by the Local Security Authority (LSASS), searching the Discretionary Access Control List (DACL) in the SDS data stream (Microsoft, 2023g).

2.3.10 *Quotas*

Another feature that will be mentioned is the support for disk quotas, which is supported by NTFS, and are used to track and control the amount of disk space each individual user may use on an NTFS file system volume (Microsoft, 2023e). Carrier states that "In versions of NTFS prior to 3.0, there existed a $Quota file system metadata file in MFT entry 9, but in versions 3.0+ the same file exists in the $Extend directory and it can be in any MFT entry".

2.3.11 *Cluster remapping*

An sector error detection mechanism was implemented in this file system, triggering when errors are detected in read operations. NTFS keeps track of bad sectors located in clusters, by allocating them to a $DATA attribute of the $BadClus file system metadata file, located in the eighth entry of the MFT in order to prevent future usages of them (Lo et al., 2023).

2.3.12 *File System Tunneling*

An operating system optimization, called file system tunneling, caches temporarily file system metadata for a certain time, which is by default 15 seconds. For example, when a file is deleted and within the threshold time, a second file is created, moved or renamed, with the same name as the deleted file, this second file $Standard Information Creation Time timestamp will be changed to the cached one. This operations are common in temporary files, and in saving mechanisms that prevent file saving data corruptions. This different behavior was found to be a cause leading to false positives in the timestamp manipulation detection tools, since a record containing attribute update to the timestamps values of this second file is observable in the $Logfile (Oh et al., 2024). This feature affects an analyst's examination of the artifacts of the file system.

2.4 *Timestamps*

According to the RFC 3339 (Klyne and Newman, 2002), timestamp is a term used to "refer to an unambiguous representation of some instant in time".

2.4.1 *Timestamp analysis*

File timestamp manipulation is when a malicious user or attacker manipulates ate least one timestamp value of a file to avoid detection by a timeline analysis (Oh et al., 2024). Timeline analysis consists in an analysis of events extracted from the file system and artifacts during a digital forensic investigation, resulting a chronological timeline. (Oh et al., 2024)

2.4.2 *NTFS Timestamps*

NTFS stores, at least, eight timestamps per MFT entry. Four of those timestamps are located in the file's $STANDARD_INFORMATION (SI) attribute, and the remainder are stored in the $FILE_NAME (FN) attribute. These timestamps are referred

in the literature as MACE, as well as MACB[4] timestamps. In this document, the MACE term will be used, and each letter is described in Table 2.1.

Acronym	Timestamp description	Contains
M	Last **M**odified Time	Last update of file content of the $DATA or $INDEX
A	Last **A**ccessed Time	Last access to file content
C	File **C**reation Time	File creation timestamp, which is creation time of MFT entry
E	MFT **E**ntry Modified Time	Last MFT record change (MFT Modified Time)

Table 2.1: MACE Timestamps description

MACE timestamps are stored in *FileTime64*, corresponding to the number of 100-nanosecond intervals (10^{-7} seconds) since 1601-01-01 00:00:00.0000000 Coordinated Universal Time (UTC) (Galhuber and Luh, 2021). This date is known as as NT time epoch, and as such, timestamp values are not affected by local time zone or daylight saving time.

The *FileTime64* is a structure that holds two DWORD variables, and combined results in a 64 bit value, and this structure is represented in the figure 2.3 (Microsoft, 2023b).

```
typedef struct _FILETIME {
  DWORD dwLowDateTime;
  DWORD dwHighDateTime;
} FILETIME;
```

Figure 2.3: Win32 API Filetime structure

Below, an older example of how to get the 64 bit value from both DWORD variables is listed.

```
ULONGLONG ullTime=((ULONGLONG)ft.dwHighDateTime<<32) | ft.dwLowDateTime;
```

More recently, the Win32 Application Programming Interface (API) provided a structure (winnt.h) that makes the convertions natively easier, as shown by the below.

```
ULARGEINTEGER{ ft.dwLowDateTime, ft.dwHighDateTime }.QuadPart
```

To simplify, the notation used in other papers to refer to the MACE timestamps under $STANDARD_INFORMATION and $FILE_NAME is $SI:M $SI:A $SI:C $SI:E , where it is also applied to the timestamps on $FILE_NAME as , for example, $FN:M.

2.4.3 *Last Access timestamp*

The last access timestamp ("A") on MACE timestamps is particularly unique, since it can be configured to update or not to update. In earlier versions, this could be achieved by changing the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\NtfsDisableLastAccessUpdate` value to 1 or 0 (Carrier, 2005;

4 Modification, Access, Changed and Birth (Carrier, 2005)

Microsoft, 2007). For performance reasons the last access timestamp was disabled by default in Windows registry, starting with the release of windows Vista. However, with the Windows 10 20H1 release, last access updates were re-enabled again, by default (Galhuber and Luh, 2024). Also, when looking directly at the value of $SI:A, it can differ from the expected time, since NTFS maintains the new access time value in memory, and can update only after 1 hour, at maximum. Another aspect is that changes in this timestamp may not be directly associated with application interactions, since, it is known that background activities and file reading operations can trigger this timestamp to update (Galhuber and Luh, 2024).

```
PS C:\Users\dev> fsutil behavior set DisableLastAccess
Usage: fsutil behavior set disableLastAccess <0-3>

 Values: 0x0 - User Managed, Last Access Updates Enabled
         0x1 - User Managed, Last Access Updates Disabled
         0x2 - System Managed, Last Access Updates Enabled
         0x3 - System Managed, Last Access Updates Disabled

 - When "System Managed" is enabled it allows the system to enable/disable
   last access time updates based on system policy.
 - If group policy is in effect or this registry key is uninitialized then
   the "System Managed" state can not be set and is not displayed.
```

Figure 2.4: fsutil command displaying the possible values of NtfsDisableLastAccessUpdate

Since Windows 10, this registry key can hold one out of four possible values (0x0, 0x1, 0x2, 0x3), as shown in the figure 2.4. The current default value is 0x2 (Microsoft, 2023c).

2.5 *Methods to perform timestamp manipulations*

To achieve timestamp manipulation, the known methods are differ in difficulty and precision.

- Windows API - This API provides three functions to modify file timestamps. The methods are SetFileTime[5], NtSetInformationFile[6], and ZwSetInformationFile[7], and enable the modification of the FILE_BASIC_INFORMATION structure, which contains the timestamp information that can be found on $SI (Galhuber and Luh, 2021). The disadvantage of this method is that the Windows API doesn't provide direct way to modify the timestamps present on $FN.
- Direct Disk Access - This technique changes the temporal values directly in the MFT. This method is more evasive, thus, can cause inconsistencies

5 https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime
6 https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntsetinformationfile
7 https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-zwsetinformationfile

resulting in a corrupted file system. Also, this method currently in unavailable if the Windows system is running, since Microsoft released a patch to prevent direct access to the system drive. (Galhuber and Luh, 2021; Oh et al., 2024).

In the literature, timestamp manipulation is often referred to as timestomping, and this is the origin for the name of tools like timestomp and nTimestomp.

2.6 *nTimestomp*

nTimestomp is a tool found in the nTimetools suite of console tools developed to work with timestamps in Windows. This tool allows to overwrite MACE timestamps of files with 100-nanosecond precision. Forensic analysts are usually taught to spot 0s in the millisecond position as evidence that timestomping has occurred. This tool uses NtSetInformationFile API which means privileged access is not required and files on NTFS/FAT and mapped drives can also be timestomped. Note that this tool is only available for 64 bit systems (limbenjamin, 2024).

3

LITERATURE REVISION

One important aspect related with a digital forensic investigation, is that a timeline should result from the reconstruction of the events presented by the artifacts and evidences. One of those artifacts are given by the timestamps, because file operations lead to timestamps changes. The first findings was in 2007 when Chow et al. described the effects of file operations on timestamps, using the MAC timestamps from $SI on a NTFS file system.

Lees (2013) explained how to use the UsnJnrl to detect removal of forensic artefacts, by revealing patterns left by programs like CCleaner, or the usage of InPrivate browsing.

Jang et al. (2016) detailed how to detect timestamp manipulation behaviors, and state that "We cannot detect timestamp manipulation by simply using $MFT analysis", which will be also found in the approach of later papers, mentioned below.

Palmbach and Breitinger (2020) explained that not only the $MFT and $LogFile should be used for detecting timestamp manipulation, but also $USNjrnl, link files, prefetch files, and Windows event logs should be used to detect this behavior.

3.1 *File System Forensic Analysis*

Carrier (2005) has a plentiful of technical designs and explanations on how NTFS works and its interaction with the operative system, detailing how some of the features of the filesystem are implemented. This book has some of it's topics outdated since it was written in the Windows NT and XP era and, for example, some of the interactions with the drivers, do not produce the same output anymore, however it provides a solid understanding of this file system.

3.2 *A computer forensic method for detecting timestamp forgery in NTFS*

Cho (2013) documents the effect of seven file operations on the timestamps of their respective files, including a rule related with the MS Word application behavior, which he adds "MS Word file has a different timestamp change behavior". The rules are:

1. "When M time is equal to C time, the file has neither been modified nor copied from another disk location. It is suggested that the file is still intact and has not been updated";

2. "When M time is before C time, the file has been copied from one system into the same/another system or moved from one partition to another partition";
3. "In a folder, if files' M times are before C times and the files have "very close" C times, the files have been":
    a) "copied from one system to the same or another system in a batch";
    b) "moved from one partition to another partition in a batch";
    c) "extracted from a compressed file";
4. "When a large number of files with "close" A times are found inside the hard drive, the files are likely to be scanned by some tool, e.g. anti-virus software or file searching tool";
5. "If image/video files within a folder have "close" A times, and no other image files have similar A times, the concerned image/video files are likely to be accessed or opened by file previewing tool, e.g. windows explorer, as thumbnails for reviewing";
6. "When files within a folder have "scattered" A times, it is highly likely that the files are accessed individually";
7. "In a folder, if files' M times are equal to C times and the files have "very close" C (M) times, the files may have been downloaded in a batch from another system over the network";

The author does not analyze the MFT entry modified time in this paper, only the other three timestamps (Modify, Access, Create). Also the author does not refer which source the timestamps analyzed came from, he does not refer to $STANDARD_INFORMATION nor $FILE_NAME.

3.3 *Time for Truth: Forensic Analysis of NTFS Timestamps*

Galhuber and Luh (2021) tested and analyzed the effectiveness of 7 third party timestomping tools, as well as a PowerShell script. During their experimentations they verified that existing resources regarding the timestamps results from file operations are outdated, lack application specific rules, and that there is not much official information about the NTFS. The authors also showed that experiments with Microsoft Office applications deviated from the expected, since they do not follow the rules of previously mapped file operations enforced by the OS, and usually deviate from the other applications timestamps results, making them unreliable as forensic sources.

The paper also makes a distinction between two timestamp tampering methods, one approach using the Windows API, which is less effective, and another requiring direct disk access, which is much more effective, but also much more difficult to apply. The Windows API only allows modifications in the $SI timestamps using the methods SetFileTime, NtSetInformationFile, and ZwSetInformationFile, accordingly. Galhuber and Luh (2021) also state that "there is no direct way to

modify $FN timestamps through the Windows API", and in order to force the $FN timestamp to hold the same values as the $SI, the tools can trigger the Operative System to update them by moving the file from a place to another, or just wait for the normal operation of copying the $SI timestamp values to the $FN timestamps.

It is also stated that if a file is moved from a FAT volume, the timestamps will contain values truncated to the nearest second, adding the risk of being detected as a false positive.

3.3.1 *Reconstructing Timelines: From NTFS Timestamps to File Histories*

Bouma et al. (2023) proposed a different approach on how the timestamp forgery detection was performed, by not only looking to the last timestamps present, but also inferring a finite timeline that precedes the last change on the time stamps by looking at the timestamps in $STANDARD_INFORMATION and $FILE_NAME, for each MFT entry. They also noted that there is a lack of "consensus of in literature on a canonical list of file operations to include for timestamp research" and the reason is that too many external variables interact with the file system, such as different application behaviors and Windows.

3.4 *Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation*

The most recent related paper, states the origin problem that lead to their work; "previous NTFS journal-based detection methods have limitations such as incorrectly identifying normal events as manipulation or detecting manipulation only in limited cases" Oh et al. (2024). The authors not only published new algorithms to overcome the problem, but also updated the tool that were already published in a previous paper (Oh et al., 2021). This new version was tested with real scenarios and known Advanced Persistent Threats (APT), and compared with other previously published algorithms, including the ones that didn't provide their tool publicly, which the authors developed and made accessible themselves.

Their detection algorithm combined the usage of $Logfile, $MFT and $UsnJrnl:J files extracting traces of timestamp manipulation in each journal, and the possible problems/limitations related with each file. The authors also provide a detailed explanation on how to obtain each artifact and traces of timestamp manipulation in each journal.

4

METHODOLOGY

As part of the work to answer the questions and hypothesis proposed, the following sections describe the experiments scenario, test cases, how the results were generated and the experiments conducted.

4.1 *Setup*

The testing environment consists in a virtual machine, since it's useful to isolate the hardware dependent factors and reduce deviation from the expected results to the obtained ones. In this case VirtualBox Version 7.0.14 r161095 is the used virtualization software. Other useful feature gained by using virtual machines is the ability to use snapshots and easily revert changes to its original or previous states. In the virtual machine, Windows 11 Education was installed, from the image that was obtained in the official Microsoft's website[1] and it's integrity was verified[2]. VirtualBox's Unattended Installation was used in order to automate the installation process.

Before beginning all tests, and even though NTFS timestamps are not timezone dependent, this environment's default timezone was set to "GMT Standard Time". This was verified through Windows PowerShell with the command Get-Timezone.

The main PowerAutomate and testing file creation flows of the testing environment is further detailed on Appendix A.

4.1.1 *Tools*

After the initial setup, Windows updates were performed, and Python 3.12.3 and the SysInternals Suite 2024.2.1 were installed through the Windows Store, Microsoft 365 (Microsoft Office) was installed, using the setup executable when was transfered from the host computer to the environment using VirtualBox's drag and drop feature.

In order to perform the timestamp manipulations, the directory `C:\Tools` was created, containing nTimeStomp and nTimeView, which were added to the folder using Virtualbox's drag and drop feature, and a new entry in the environment variable `PATH` was added pointing to this new directory.

Another important suite of tools required for this testing is the Microsoft 365, formerly Microsoft Office, which includes Word, Excel and PowerPoint that will

1 https://www.microsoft.com/software-download/windows11
2 Win11_23H2_English_x64v2.iso SHA256: 36DE5ECB7A0DAA58DCE68C03B9465A543ED0F5498AA8AE60AB45FB7C8C4AE402

Software	Version
VirtualBox	7.0.14 r161095
Microsoft Windows 11 Education	10.0.22631 Build 22631 (v23H2)
Autopsy	4.21.0 (TSK 4.12.1)
Mft2Csv	2.0.0.50
LogFileParser	2.0.0.51
NTFS Log Tracker	1.7
NTFS Log Tracker	1.9
Timestamp analyser	Not Applicable
Python	3.12.3
Sysinternals	2024.2.1
Microsoft® Word for Microsoft 365 MSO	2407 Build 16.0.17830.20056 64-bit
Microsoft® Excel® for Microsoft 365 MSO	2407 Build 16.0.17830.20056 64-bit
Microsoft® PowerPoint® for Microsoft 365 MSO	2407 Build 16.0.17830.20056 64-bit
Microsoft Power Automate	2.43.204.241
nTimestomp	1.2
nTimeview	1.0

Table 4.1: Software used and respective versions.

be mentioned further ahead. For this suite of applications, an online account is required, so for the purpose of this experiments, my personal academic account was used, disabling the option "Allow my Organization to manage my device" and using the option "No, sign in to this app only".

Finally, to perform the test cases automatically Microsoft Power Automate v2.43.204.24107 was used, allowing to achieve the results simulating the user/command line interface behavior always with the same steps, reducing the possibility of human error, and allowing to perform this test in future versions of the OS.

4.1.1.1 *NTFS Log Tracker v1.71*

Oh et al. (2021) presents the tool *NTFS Data Tracker v1.0*[3]. Although, at the time of writing was not available to download, the website is still online, and the author provides a further improved version of the tool, *NTFS Log Tracker v1.71*[4].

This tool uses $LogFile, $MFT and $UsnJrnl and attempts to detect suspicious behavior. Moreover, this tool was already tested and was verified to find timestomped files, in controlled scenarios.

3 https://sites.google.com/site/forensicnote/ntfs-data-tracker
4 https://sites.google.com/site/forensicnote/ntfs-log-tracker

4.1.1.2 *Timestamp Analyser*

The tool associated to Bouma et al. (2023) is written in Java, and its source code is available at the author's GitHub[5].

This tool receives the $MFT as the only only input, and provides an output with the anomalies found on the analysis.

4.1.1.3 *NTFS Log Tracker v1.9*

Oh et al. (2024) describes version 1.9 of the NTFS Log Tracker tool. This tool also uses the $LogFile, $MFT and $UsnJrnl as sources to analyze the events present on these files, and provide the detection of suspicious behavior, if found. The difference is that this version, is more accurate on the findings the previous version, as will be discussed further in the tests results.

Before shutting down the virtual environment, the tool Sync from SysInternals was used, in order to insure that any modified data present in the cache is written to the disks (Russinovich, 2016).

4.2 *Test Cases*

To organize and structure the procedures and steps required to reach a result or artifact, test cases were written. In this case, there are 10 test cases matching the SANS Windows Time Rules[6] where its compared the results difference in 10 distinct operations between Windows 10 v1903 and Windows 11 v22H2. Since the test environment has another version of the operative system (v23H2), performing the same 10 operations will serve as a control to verify if any operation has a different effect on the $Standard Information timestamps.

Besides the SANS (2024) like test cases (TC01-TC08 and TC10), ten more test cases were elaborated in order to verify other operations involving some of the most common files found in a digital forensic investigation such as files from the Microsoft Office applications due to their specific behavior, PDF files created with Word, Microsoft print to PDF and, lastly, file system tunneling behavior (TC09, TC11(A,B,C), TC12 (A,B,C), TC13 (A,B) and TC14).

4.2.1 *TC01 - File Creation*

This test case serves as a control sample, to verify if a newly created file has the same value in all timestamps.
Simplified steps to reproduce:

1. Open Notepad application.

5 https://github.com/JelleBouma/TimestampAnalyser
6 https://www.sans.org/posters/windows-forensic-analysis/

2. Write *Lorem Ipsum* placeholder text.
3. Save the file as `C:\testing\TC01_creation.txt`.
4. Close the Notepad.

The figure 4.1 shows the TC01 implementation in PowerAutomate.

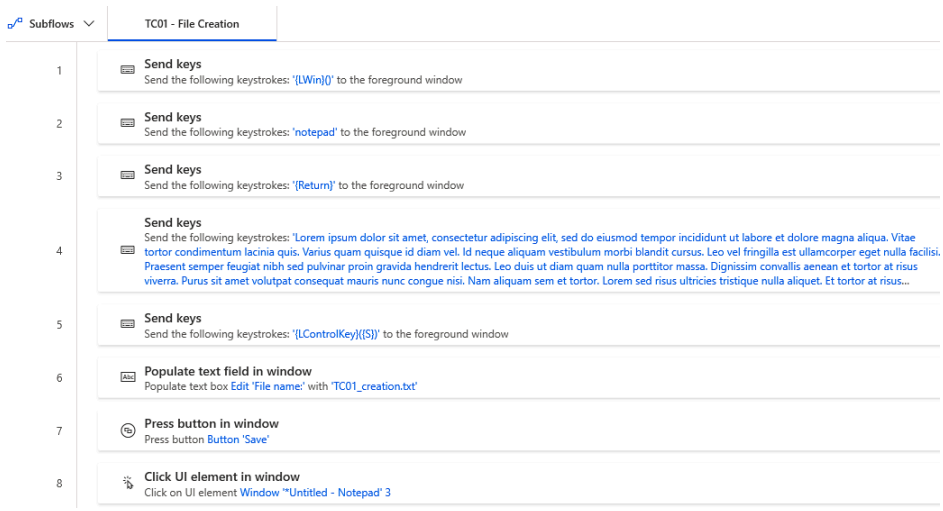

Figure 4.1: Test Case TC01 in PowerAutomate

4.2.2 *TC02 - File Access*

This test case, determines which timestamps are modified after accessing a file.
Simplified steps to reproduce:

1. Open testing directory `C:\testing\` on Windows Explorer.
2. Using the GUI, open the file `C:\testing\TC02_access.txt` with a left mouse double click action.
3. Close the Notepad application.

The figure 4.2 shows the TC02 implementation in PowerAutomate.

4.2.3 *TC03 - File Modification*

This test case, determines which timestamps are modified after modifying the contents of a file.
Simplified steps to reproduce:

1. Open the testing directory on Windows Explorer.
2. Using the GUI, open the file `C:\testing\TC03_modified.txt` with a left mouse double click action.

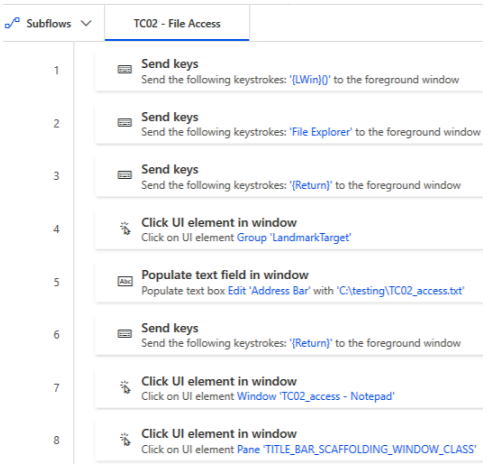

Figure 4.2: Test Case TC02 in PowerAutomate

3. Add *Lorem Ipsum* placeholder text.
4. Save the file.
5. Close the Notepad application.

The figure 4.3 shows the TC03 implementation in PowerAutomate.

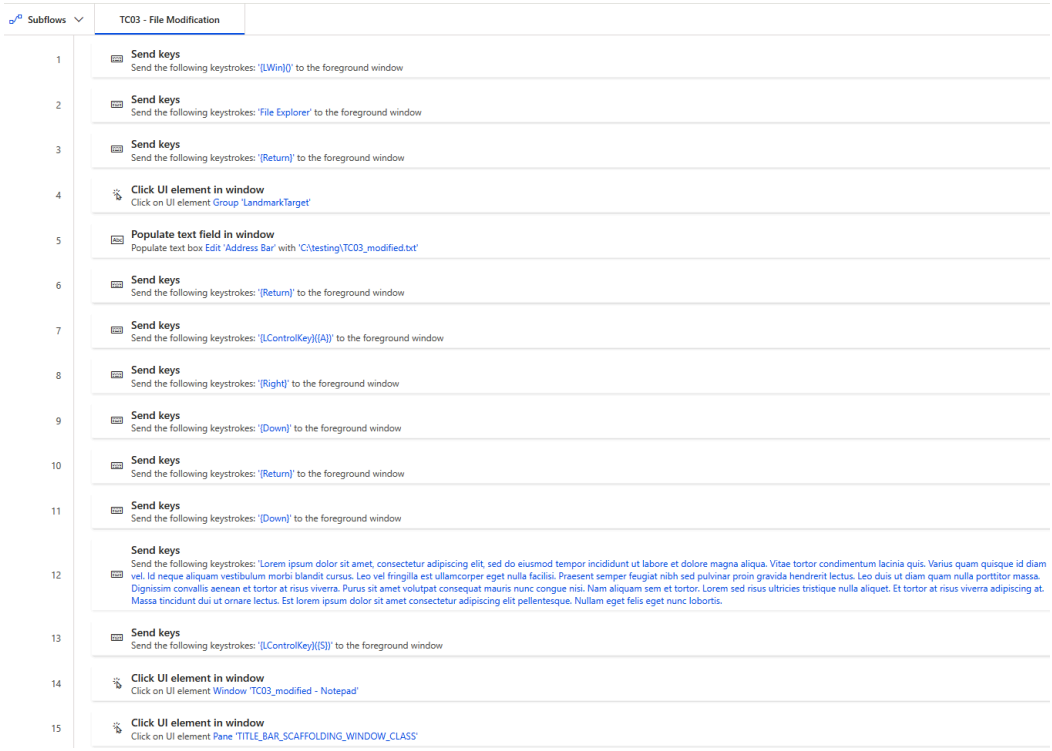

Figure 4.3: Test Case TC03 in PowerAutomate

4.2.4 *TC04 - File Rename*

This test case, determines which timestamps are modified after renaming the file.
Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.
2. Using the GUI, select `C:\testing\TC04_rename.txt` with a left mouse click action.
3. Press `F2` key to begin the rename action.
4. Input text `TC04_rename_renamed.txt`.
5. Press `Return` key.
6. Close Windows Explorer.

The figure 4.4 shows the TC04 implementation in PowerAutomate.

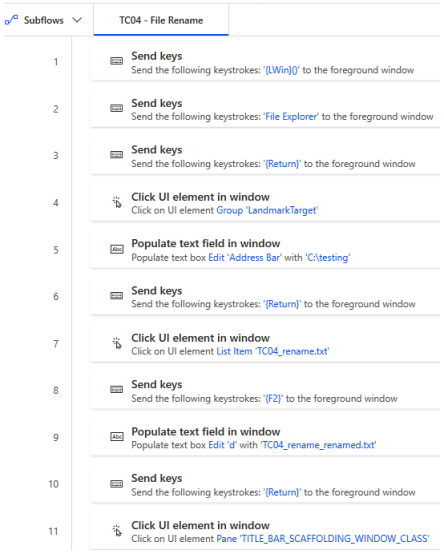

Figure 4.4: Test Case TC04 in PowerAutomate

4.2.5 *TC05 - File Copy*

This test case, determines which timestamps are modified between the original file and its copy.
Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.
2. Using the GUI, select `C:\testing\TC05_copy.txt` with a left mouse click action.
3. Press Copy Button on Windows Explorer Window.
4. Press Paste Button on Windows Explorer Window.

5. Close Windows Explorer.

The figure 4.5 shows the TC05 implementation in PowerAutomate.

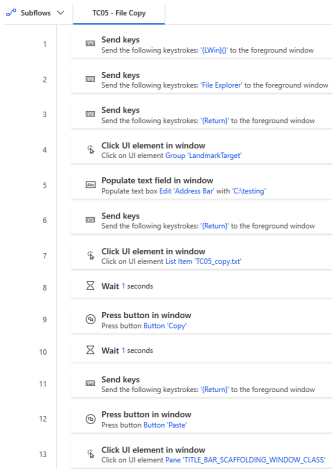

Figure 4.5: Test Case TC05 in PowerAutomate

4.2.6 *TC06 - File local move*

This test case, determines which timestamps are modified between the original file and after moving it to a directory under the testing directory.
Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.
2. Using the GUI, select `C:\testing\TC06_local_move.txt` with a left mouse click action.
3. Press Cut Button on Windows Explorer Window.
4. Using the GUI, change the directory to `C:\testing\TC06`
5. Press Paste Button on Windows Explorer Window.
6. Close Windows Explorer.

The figure 4.6 demonstrates the TC06 implementation in PowerAutomate.

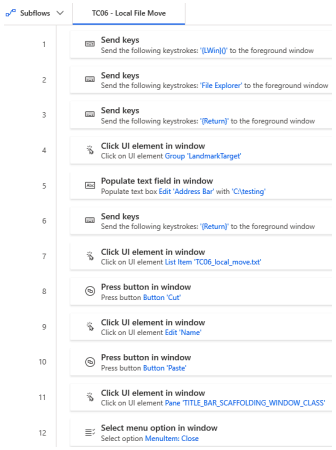

Figure 4.6: Test Case TC06 in PowerAutomate

4.2.7 *TC07 - File Move to Different Volume using Command Line Interface*

In this test case, it's verified which timestamps are modified between the original file and after moving it to a different mounted volume, using the Windows Power Shell.

Simplified steps to reproduce:

1. Execute the command:
   move C:\testing\TC07_volumefile_move_cli.txt F:\TC07_volumefile_move_cli.txt

The figure 4.7 shows the TC07 implementation in PowerAutomate.

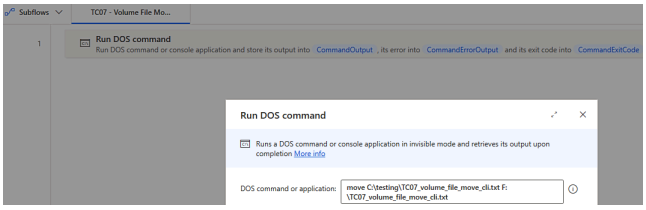

Figure 4.7: Test Case TC07 in PowerAutomate

4.2.8 *TC08 - File Move to Different Volume using Windows Explorer*

In this test case, it's verified which timestamps are modified between the original file and after moving it to a different mounted volume, using the Windows Explorer.

Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.

2. Using the GUI, select `C:\testing\TC08_volume_file_move_explorer.txt` with a left mouse click action.
3. Press Copy Button on Windows Explorer Window.
4. Using the GUI, change the directory to `F:\`
5. Press Paste Button on Windows Explorer Window.
6. Close Windows Explorer.

The figure 4.8 shows the TC08 implementation in PowerAutomate.

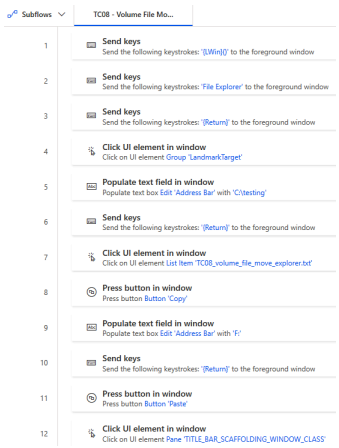

Figure 4.8: Test Case TC08 in PowerAutomate

4.2.9 *TC09 - File Timestamps Manipulation*

In order to verify which timestamps are modified between the original file and after performing intentional timestamp manipulation, the nTimestomp tool is used in this test case. It is also worth noting that this is the only time that timestamp manipulation is performed, so the purpose of this test case is also to provide a file which shall be the only one flagged as suspicious in the analysis tools.
Simplified steps to reproduce:

1. Run the nTimestomp command in the target file, in the target directory, to change the four $Standard_Information timestamps to "1999-08-18 12:34:56.7890123".

   The figure 4.9 shows the TC09 implementation in PowerAutomate.

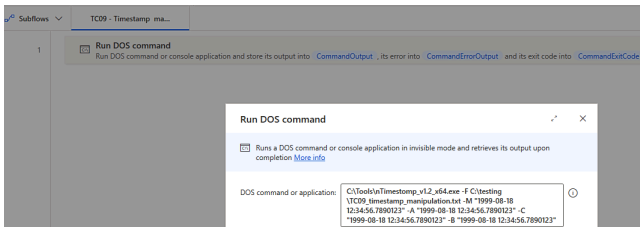

Figure 4.9: Test Case TC09 in PowerAutomate

4.2.10 *TC10 - File Deletion*

In this test case, it's verified which timestamps are modified between the original file and after deleting the file with the Shift+Delete action, which opens a window prompt asking "Are you sure you want to permanently delete this file?"
Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.
2. Using the GUI, select `C:\testing\TC10_file_deletion.txt` with a left mouse click action.
3. Press Shift+Delete.
4. Press Return.
5. Close Windows Explorer.

The figure 4.10 shows the TC10 implementation in PowerAutomate.

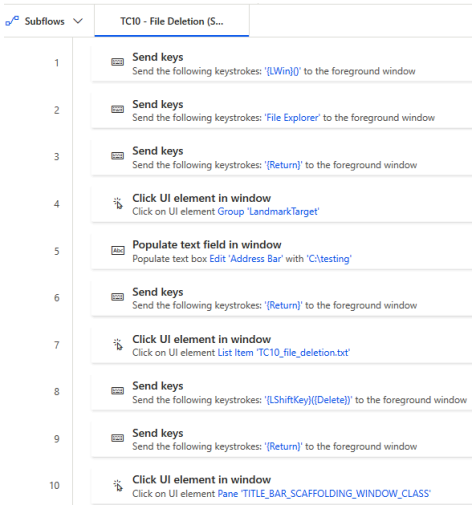

Figure 4.10: Test Case TC10 in PowerAutomate

4.2.11 *TC11 - Office File Creation*

Since the Microsoft Office applications have been reported papers to behave differently than the Notepad application (Cho, 2013), in this test case, it's verified the file creation behavior, tested in three applications, Word, Excel and PowerPoint.
Simplified steps to reproduce:

1. Open the designated office application.
2. Write the first three paragraphs of *Lorem Ipsum* placeholder text to file.
3. Save the file as `C:\testing\TC11_<ApplicationName>.<Extension>`
4. Close the application.

In this case, PowerAutomate only has native support for Excel and Word, as that, the test performed with PowerPoint was made differently.

The figure 4.11 shows the TC11-A implementation in PowerAutomate, it creates a file with Excel.

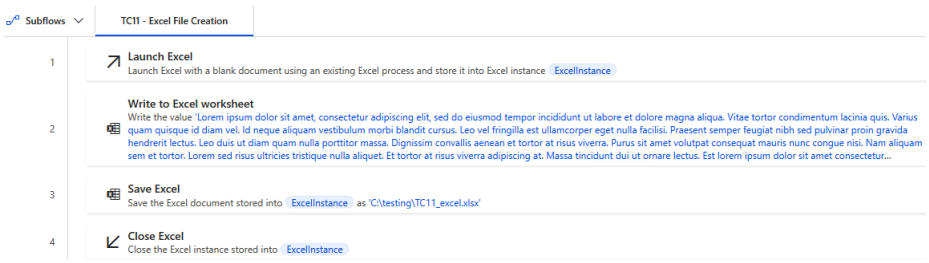

Figure 4.11: Test Case TC11-A in PowerAutomate, Excel file creation

Then, TC11-B was the file creation with PowerPoint, as the figure 4.12 shows.

Finally, TC11-C creates a file with Word and can be seen in the figure 4.13.

4.2.12 *TC12 - Office File Modification*

This test case, determines the file modification behavior, tested in three applications, Word, Excel and PowerPoint.
Simplified steps to reproduce:

1. Open testing directory on Windows Explorer.
2. Using the GUI, Open `C:\testing\TC12_<ApplicationName>.<Extension>` with a left mouse double click action.
3. Add the two paragraphs of *Lorem Ipsum* placeholder text to the file.

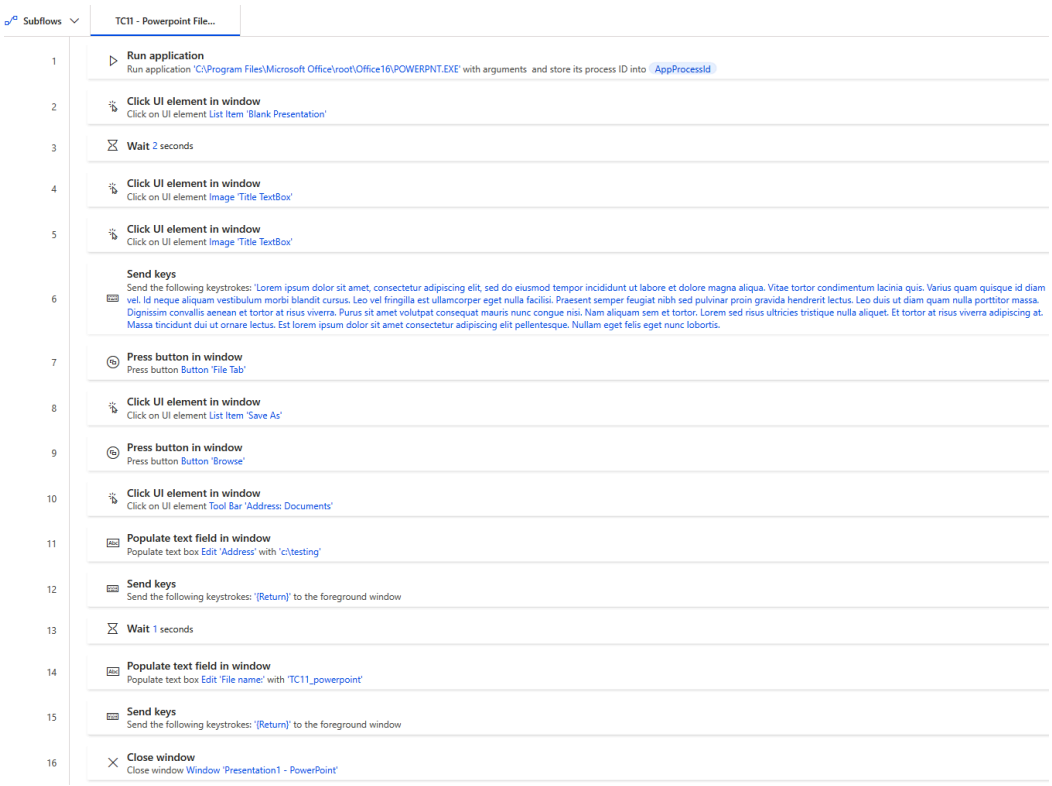

Figure 4.12: Test Case TC11-B in PowerAutomate, PowerPoint file creation

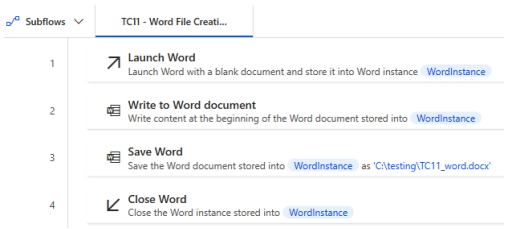

Figure 4.13: Test Case TC11-C in PowerAutomate, Word file creation

4. Save the file as `C:\testing\TC12_<ApplicationName>.<Extension>`
5. Close the application.

Similiar with TC11, the test performed with PowerPoint was made differently, since PowerAutomate only has native support for Excel and Word.

The implementation of this test case with Excel can be seen in the figure 4.14.

The PowerPoint test (see figure 4.15), opens the existing file created in TC11-B, and adds two paragraphs.

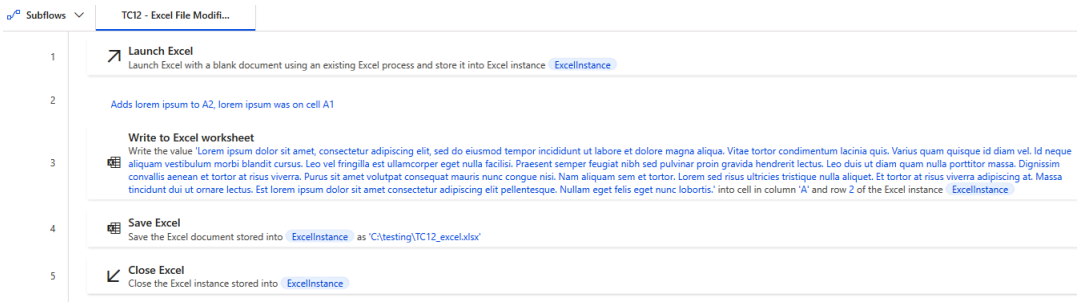

Figure 4.14: Test Case TC12-A in PowerAutomate, Excel file modification

Lastly, modifying a Word file is tested in TC12-C, and can be seen in the figure 4.16.

4.2.13 *TC13 - Convert File to PDF*

This test is performed twice, using Microsoft Print to PDF and using Microsoft Word save as PDF. As stated in Test case 11 and 12, Microsoft Office applications have been reported to behave differently than the text applications, so the creation of PDF files is tested twice, one using Word save as PDF, and the other using Microsoft Print to PDF feature.

[TC13-A] Word save as PDF - simplified steps to reproduce:

1. Create and open a Word file.
2. Write two paragraphs of *Lorem Ipsum* placeholder text to the file.
3. Save the file as `C:\testing\TC13_word.pdf` in PDF format.
4. Close the file.

This test is shown by the figure 4.17, using the native commands on PowerAutomate for Word.

[TC13-B] Microsoft print to PDF - simplified steps to reproduce:

1. Create and open a Word file.
2. Write two paragraphs of *Lorem Ipsum* placeholder text to the file.
3. Save the file as `C:\testing\TC13_ms_print_to_pdf.docx` as docx.
4. Close the file.

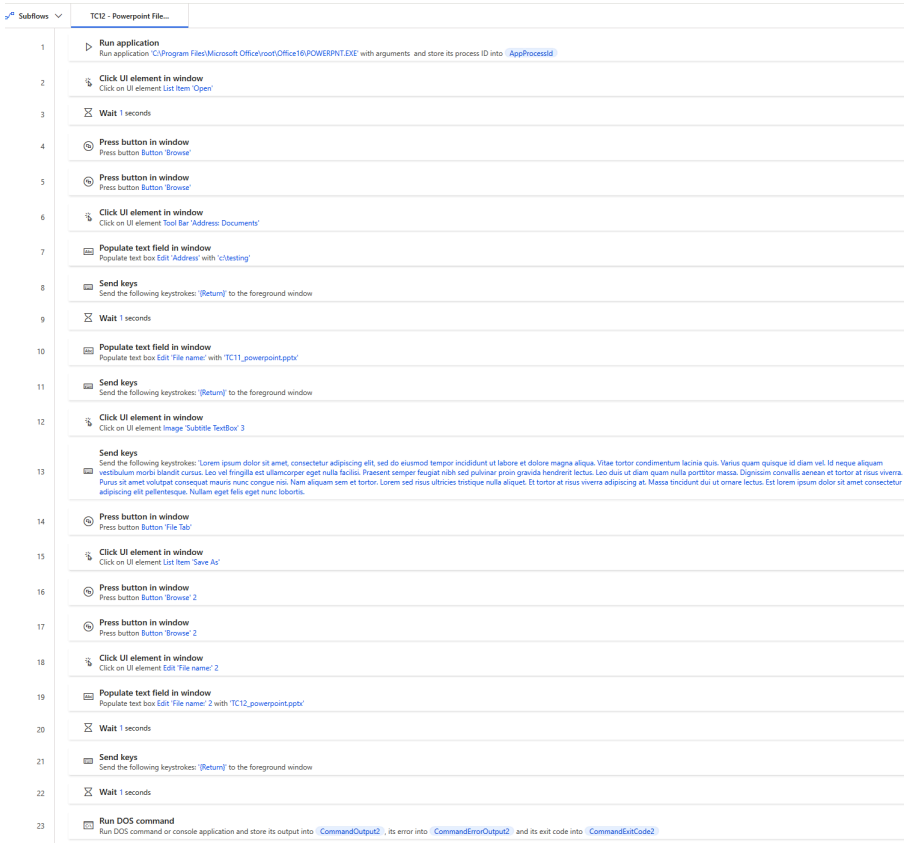

Figure 4.15: Test Case TC12-B in PowerAutomate, PowerPoint file modification

5. Print the file with Power Automate's "Print Document" function with the printer "Microsoft Print to PDF".
6. Save the file as `C:\testing\TC13_ms_print_to_pdf.pdf`.

This test is shown by the figure 4.18, it sets the default printer as Microsoft Print to PDF, and then proceeds to save the file as PDF using the feature.

4.2.14 *TC14 - File System Tunneling*

This test case, determines which timestamps are modified between the original file and after deleting the file and creating a new one with the same name as the original file, within less than 15 seconds, in order to force the File System Tunneling feature.

1. Delete the file `C:\testing\TC14_file_system_tunneling.txt` with Power Automate's Delete File function.

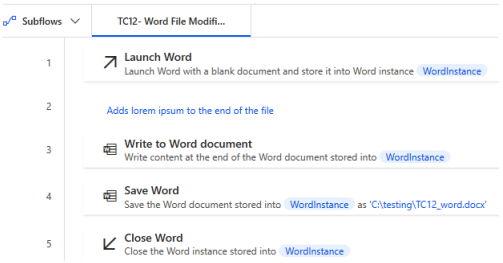

Figure 4.16: Test Case TC12-C in PowerAutomate, Word file modification

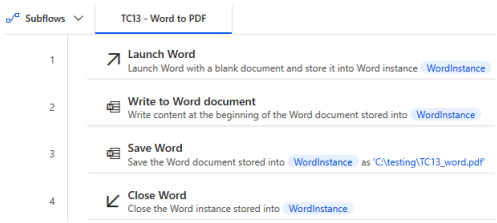

Figure 4.17: Test Case TC13-A in PowerAutomate, Word save as PDF

2. Output the command `Get-Date -Format "yyyy-MM-dd HH:mm:ss.fffffff"` to the text.
3. Save the file as `C:\testing\TC14_file_system_tunneling.txt`

This test is shown by the figure 4.19.

4.3 *Data collection*

In order to obtain a snapshot of the file system in order to further analyze, VirtualBox's `clonehd` command was used to create an image of the virtual disk. The command is shown below.

```
VBoxManage clonehd "<path-to-vm-environment>\Snapshots\{uuid}.vdi" "<
    destination-path>.dd" --format RAW
```

Afterwards, Autopsy was used to open the created virtual disk images, to verify the data and to extract the artifacts $Logfile, $UsnJrnl:J and $MFT. After extracting these artifacts, they were processed by the analysis tools NTFS Log Tracker v1.71, NTFS Log Tracker v1.9 and Timestamp Analyser.

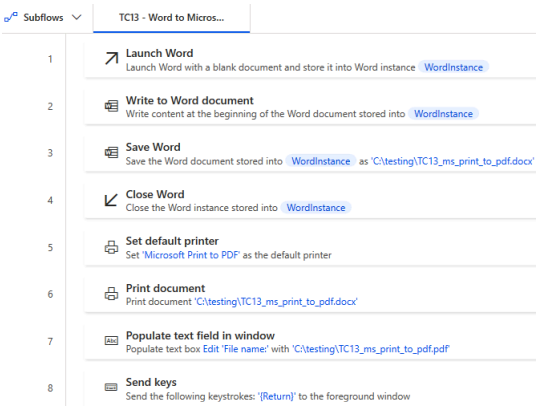

Figure 4.18: Test Case TC13-B in PowerAutomate, Microsoft Print to PDF

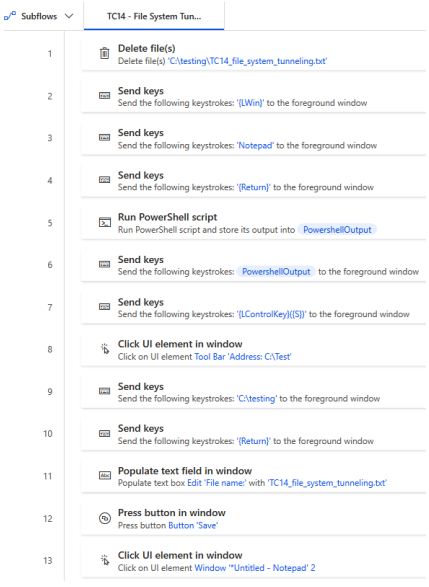

Figure 4.19: Test Case TC14 in PowerAutomate

5

RESULTS ANALYSIS

In this chapter, the results of the tests performed will be presented and discussed.

5.1 *Test Cases*

The results of the timestamp analysis is presented in Table 5.1, and all the MACE timestamps were taken from the $STANDARD_INFORMATION attribute of each file.

Table 5.1: Test Cases Results table without TC11 and TC12.

Test Case ID	Test Case Description	Expected Results	Actual Results	Pass
TC01	File Creation	M - Time of file creation A - Time of file creation E - Time of file creation C - Time of file creation	M - Approx. Time of file creation A - Approx. Time of file creation E - Approx. Time of file creation C - Time of file creation	x
TC02	File Access	M - No Change A - Time of file access E -No Change C -No Change	M - No Change A - Time of file access E - Approximately Time of Access C -No Change	x
TC03	File Modification	M - Time of Modification A - Time of Modification E - Time of Modification C - No Change	M - Time of Modification A - Time of Modification E - Time of Modification C - No Change	✓

Continued on next page

Table 5.1: Test Cases Results table without TC11 and TC12. (Continued)

TC04	File Rename	M - No Change A - No Change E - Time of file rename C - No Change	M - No Change A - No Change E - Time of file rename C - No Change	✓
TC05	File Copy	M -Same as Original A - Time of file copy E - Same as Original C - Time of file copy	M - Same as Original A - Time of file copy E - Same as Original C -Time of file copy	✓
TC06	Local File Move	M -Same as Original A - Time of file move E - Time of file move C -Same as Original	M - Same as Original A - Same as Original E - Same as Original C -Time of file move	x
TC07	Volume File Move (CLI)	M - Inherited from Original A - Time of file move via CLI E - Time of file move via CLI C - Time of file move via CLI	M - Inherited from Original A - Time of file move via CLI E - Inherited from Original C - Time of file move via CLI	x
TC08	Volume File Move (Explorer)	M - Inherited from Original A - Time of file move via GUI E - Time of file move via GUI C - Inherited from Original	M - Inherited from Original A - Time of file move via GUI E - Inherited from Original C - Inherited from Original	x

Continued on next page

Table 5.1: Test Cases Results table without TC11 and TC12. (Continued)

TC09	Timestamp Manipulation with nTimetools	M - Time of manipulation A - Time of manipulation E - Time of manipulation C - Time of manipulation	M - Time of manipulation A - Time of manipulation E - Time of manipulation C - Time of manipulation	✓
TC10	File Deletion (Shift+Delete)	M - No Change A - No Change E - No Change C - No Change	M - No Change A - No Change E - No Change C - No Change	✓
TC13-A	Word to PDF (Word save as PDF)	M - Time of PDF file creation A - Time of PDF file creation E - Time of PDF file creation C - Time of PDF file creation	M - Time of PDF file creation A - Time of PDF file creation E - Time of PDF file creation C - Time of PDF file creation	✓
TC13-B	Word to PDF (Microsoft Print to PDF)	M - Time of PDF file creation A - Time of PDF file creation E - Time of PDF file creation C - Time of PDF file creation	M - Approx. Time of file creation A - Approx. Time of file creation E - Approx. Time of file creation C - Time of PDF file creation	x
TC14	File System Tunneling with Windows Explorer	M - Time of Modification A - Time of Modification E - Time of Modification C - No Change	M - Time of Modification A - Time of Modification E - Time of Modification C - No Change	✓

The test cases TC3, TC4, TC5, TC9, TC10, TC13-A and TC14 matched the results with the expected results. On the other hand, the rest of the tests performed did not meet the expected results.

In TC01, file creation performed with GUI usage and the Notepad application, the output file has the M, A and E timestamps were approximately close to the Creation File timestamp, with the exact same value.

In TC02, the expected result was the change of $SI:A value to the time of access operation. The actual results show that the Access Time in fact changed to the time of access operation, but $SI:E also changed, with a time value greater than $SI:A. As that, the test did not verify the behavior published by SANS.

In the case of TC06, moving a file to a folder contained within the same NTFS volume was expected to inherit the $SI:M and $SI:C values from the original file, and change values on $SI:A and $SI:E time of the operation. Although, the results shows that $SI:E was inherited and $SI:C was modified, to the time of copy, thus, this results do not verify the previous Windows 11 v22H2 information found, published by SANS.

TC07 reveals that the MFT entry modified time was inherited from the original file, instead of changing to the time of the operation.

In TC8, only the Access time changed, moving a file with (Cut/paste) using the GUI to another NTFS volume.

Saving as PDF in Microsoft Word and using Microsoft Print to PDF feature behaved differently, since in TC13-A the four timestamps hold the same value, however, in TC13-B, the Creation File timestamp as one value, and the $SI:M, $SI:A and $SI:E have approximately the same value as $SI:C, where the value of $SI:M = $SI:A = $SI:E.

On the other hand, in TC11 (A,B,C) and TC12 (A,B,C), the expected results were designed based in the findings of Galhuber and Luh (2021), including the $FILE_NAME timestamps in the analysis, and are presented in Table 5.2.

The timestamps used in this chapter are presented in appendix B.

As shown in Table 5.2, only TC11-A had a rule with a pattern that matched the timestamps present on the file after the test.

5.2 *Tools Results*

After obtaining the three files ($MFT, $Logfile and $UsnJrnl:$J) from the Windows installation virtual disk using Autopsy, these were processed using the analysis tools in order to understand if they detect the manipulation correctly, and their false positives.

5.2.0.1 *NTFS Log Tracker v1.71*

This earlier version of the tool contains five possible inputs, but in order to perform the analysis only the three files mentioned above are used as the figure 5.1 shows.

This tool has a feature called "Suspicious Behavior Detection", where it parses the files and outputs the occurrences that it classified as suspicious, and where the file modified in TC09 should be found.

Table 5.2: Test Cases 11 and 12 results table.

Test Case ID	Test Case Description	Expected Results	Actual Results	Pass
TC11-A	Office File Creation (Excel)	$FN:C = $SI:C < $FN:A = $FN:E = $FN:M = $SI:M < ($SI:A, $SI:E)	$FN:C = $SI:C < $FN:A = $FN:E = $FN:M = $SI:M = $SI:A = $SI:E	X
		$FN:C = $SI:C < $FN:A = $FN:M = $SI:M < $FN:E < $SI:E		
		$FN:C = $SI:C < $FN:A = $FN:M ⩽ $SI:A		
		$FN:C = $SI:C < $FN:A = $FN:M = $SI:M < $FN:E < ($SI:A, $SI:E)		
TC11-B	Office File Creation (PowerPoint)	$FN:A = $FN:C = $FN:E = $FN:M = $SI:C < $SI:M < ($SI:A, $SI:E)	$FN:C = $SI:C < $FN:A = $FN:E = $FN:M < $SI:M < $SI:E < $SI:A	X
		$FN:A = $FN:C = $FN:M = $SI:A = $SI:C = $SI:M ≋ $FN:E < $SI:E		
TC11-C	Office File Creation (Word)	$FN:A = $FN:C = $FN:M = $SI:A = $SI:C = $SI:M ≋ $FN:E < $SI:E	$FN:C = $SI:C < $FN:A = $FN:M = $FN:E = $SI:M = $SI:A < $SI:E	X
		$FN:C = $SI:C < $FN:A = $FN:M = $SI:M < $FN:E < ($SI:A, $SI:E)		
TC12-A	Office File Modification (Excel)	($FN:C, $SI:C) < $FN:A = $FN:M = $SI:M < ($FN:E, $SI:A, $SI:E)	$FN:C = $SI:C < $FN:A = $FN:M = $FN:E = $SI:M < $SI:A < $SI:E	X
TC12-B	Office File Modification (PowerPoint)	($FN:C, $SI:C) < $FN:A = $FN:M = $SI:A = $SI:M < ($FN:E, $SI:E)	$FN:C = $SI:C < $FN:A = $FN:M = $FN:E < $SI:M = $SI:A = $SI:E	X
TC12-C	Office File Modification (Word)	($FN:C, $SI:C) < $FN:A = $FN:M = $SI:M < ($FN:E, $SI:A, $SI:E)	$FN:C =$SI:C < $FN:A = $FN:M = $FN:E = $SI:M < $SI:A = $SI:E	X

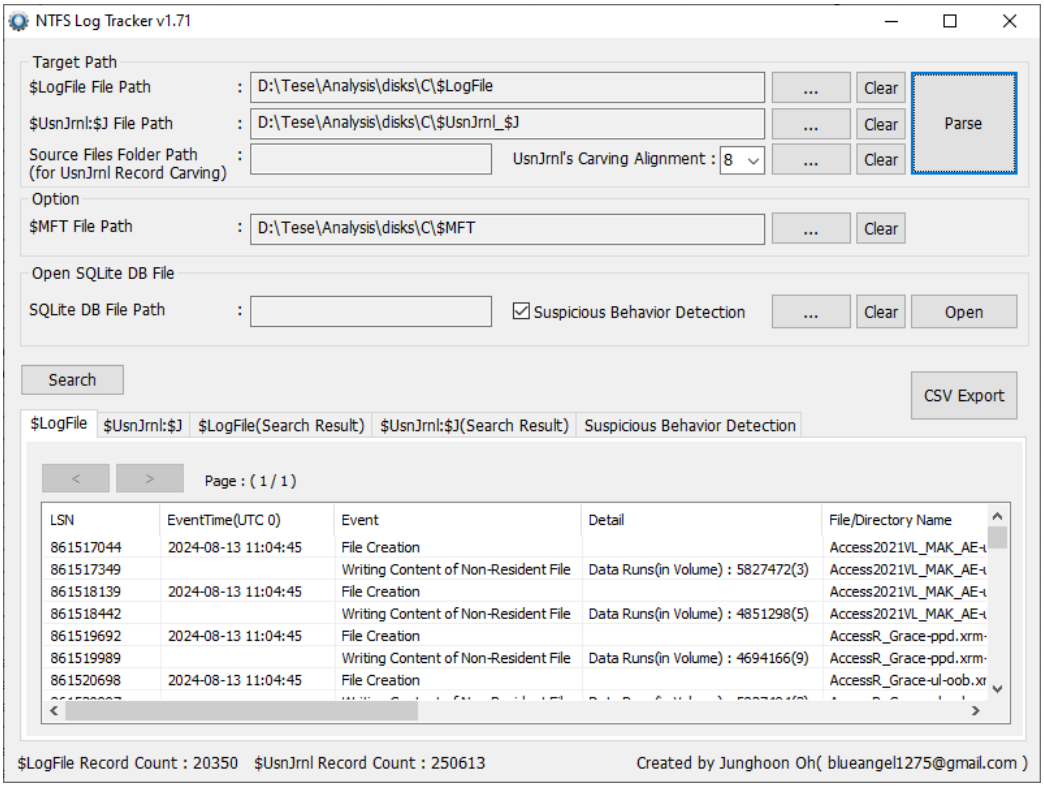

Figure 5.1: NTFS Log Tracker v1.71 main window

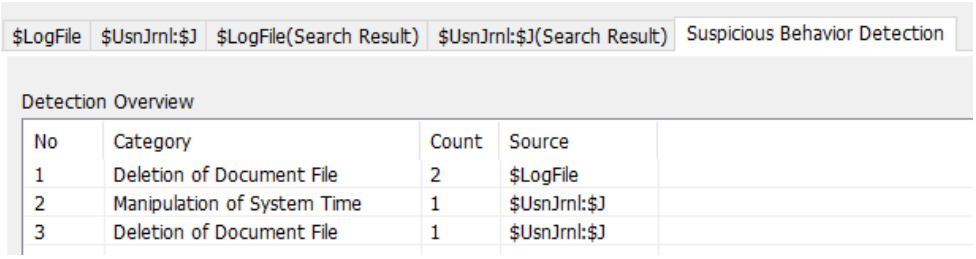

Figure 5.2: NTFS Log Tracker Suspicious Behavior Results

The figure 5.2 shows the output tab of the Suspicious Behavior Detection feature, aggregatted by categories, with a total of four detections.

Under the "Deletion of Document File" the tool indicates the file "TC12_powerpoint.pptx" and "TC12_ms_print_to_pdf.pdf" have been deleted, and provides the LSNs to the records found the in $Logfile.

In the "Manipulation of System Time" category, parsed from the change journal, there is an event of "system time reversal has occurred" with the date in the format (YYYY-MM-DD HH:mm:ss -> YYYY-MM-DD HH:mm:ss), and also the detection location on the journal.

Lastly, "Deletetion of Document File" was found on the change journal, indicating that "TC11_powerpoint.pptx" was deleted.

Keep in mind that, the artifacts were extracted from the OS installation disk, thus, they contain events related with the timestamp manipulation performed in TC09, and the tool did not flag any event related with this test as suspicious.

5.2.0.2 *NTFS Log Tracker v1.9*

Released in may, this later version of NTFS Log Tracker has practically the same look, with exception for the email at the bottom right as seen in the figure 5.3.

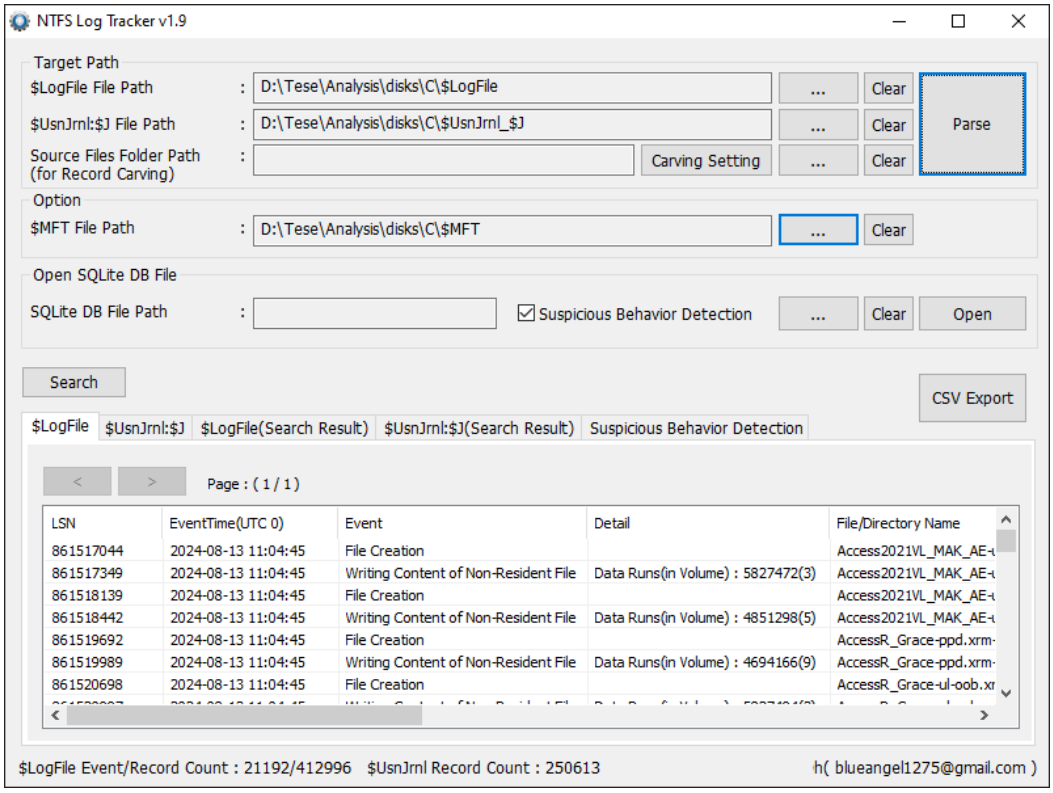

Figure 5.3: NTFS Log Tracker v1.9 main window

Parsing the same artifacts in this version with the suspicious behavior feature, led to five detections. Four of these detections were the exact same as the previous version v1.71, and a newer one with the category "Execution of Suspicious Programs", parsed from the change journal as the 5.4 demonstrates.

This time evidences of TC09 were classified as suspicious by the tool as shown by the figure 5.5. The USN provided points to a Windows Prefetch event containing the flags "File_Created / Data_Added / File_Closed".

The figure 5.6 demonstrates, the entry displayed in the tool correspondent to the detection of nTimestomp execution.

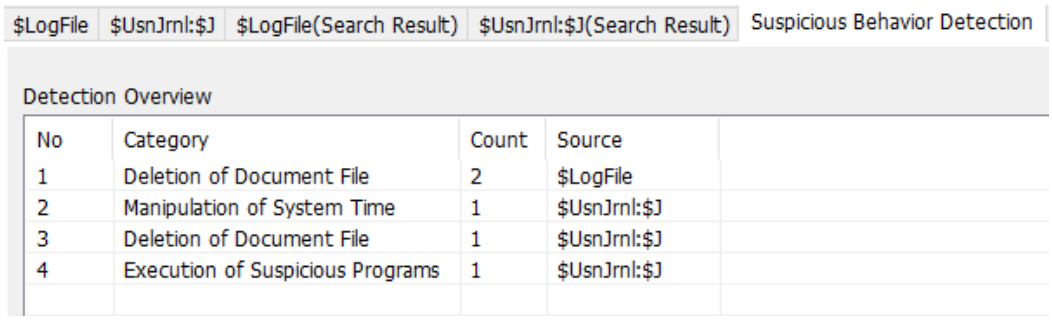

Figure 5.4: NTFS Log Tracker v1.9 suspicious Behavior Detections tab

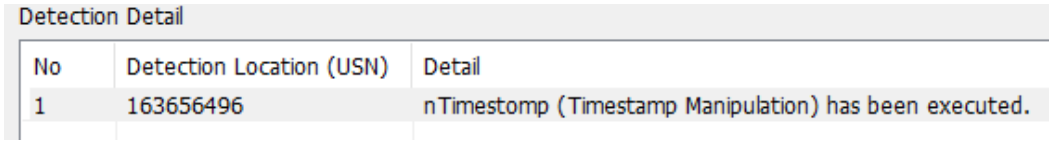

Figure 5.5: NTFS Log Tracker v1.9 detection of nTimestomp

Figure 5.6: NTFS Log Tracker v1.9 displaying the event found related with nTimestomp

Note that, this tool was developed and originally tested by the author in Windows 10 Pro x64, as that, being tested in Windows 11 23H2 confirms that it still functions correctly in a different Windows version.

5.2.0.3 *Timestamp Analyser*

This tool has no graphical user interface and was executed using the IntelliJ IDEA, this project contains a file explaining the inputs and outputs of the tool.

Using the same $MFT provided to the other tools, this one outputs a text file with all the entries processed and event sequence for each entry on the MFT, using its attributes.

In 678366 entries, only 130 were related with the test cases files. These entries, contain records related with all the test (TC01 - TC14) except for TC07 and TC10. Records with anomalies were identified with "irregular time-stamps", one of those files was the result of TC09. It correctly identified the purposely manipulated file. The problem is that this output file provided 52699 entries with "irregular time-stamps", including entries which were created by File system Tunneling and Microsoft Office Files. Only 3 out of 52699 entries categorized with "irregular time-stamps" are related to test cases files, which were TC09, TC11-C and TC12-A.

The results obtained shows that in a disk that was created to be similar to a real disk with only one file intentionally manipulated, this tool will generate high numbers of false positives to investigate.

Another problem is the false information present regarding the origin, or the actions performed in the events. For example, there is an entry in the output regarding the TC11-A file `.\testing\TC11_excel.xlsx` which contains "Copy from FAT volume | Copy from FAT volume with last access update enabled". The tests were all performed in NTFS volumes, thus, no FAT volume was involved with this particular file.

Examples of the output provided by this tool will be listed on the appendix B, section B.2.

5.3 *Results Analysis*

TC01, although it failed in the test performed using the Notepad via GUI, the other test files (TC02 - TC10 and TC14) created using the PowerAutomate's function "Write text to file" matched the timestamp pattern published by SANS.

Regarding to the SANS (2024) information related with timestamps pattern of file operations, it was observed that TC02, TC06, TC07 and TC08 did not behave as expected, this could be explained by the usage of different applications/commands to perform the test for TC02 and TC06, however, moving a file to a different NTFS volume using the Windows Explorer, should have the same behavior, since it was performed in the same application, using the same actions (cut and paste). This divergence could be explained by the tests being performed in a different Windows version.

The difference in timestamps of a newly created file between Microsoft Word and Notepad observed was also mentioned in (Galhuber and Luh, 2021) and (Bouma et al., 2023). It was also observed that $SI.A and $SI.E are the two most recently adapted, being greater or equal compared to $SI:C and $SI:M, as mentioned in Galhuber and Luh (2021) , and the authors justified this behavior describing that during the saving operation, when a file name is provided, the time value for $SI.E is adjusted, and, since the file is still opened, its content is accessed at the same time, thus, influencing the value of $SI:A. Susceptible on the processing time of the steps performed, saving the file and closing the application, the two timestamps values vary.

TC11-A was the test with a timestamp pattern more similar to the ones defined in the expected results, if and only if $SI:A and $SI:E are disregarded, since they are equal to $SI:M, and not greater than as expected. In this case, the timestamps result pattern would be: `$FN:C = $SI:C < $FN:A = $FN:E = $FN:M = $SI:M`. If the pattern did not include the timestamps mentioned earlier ($SI:A and $SI:E) the test would have passed.

One goal of the tests was to evaluate the tools capability of detecting timestamp manipulations in a close to real scenario disk image. With the results obtained, it can be stated that NTFS Log Tracker v1.71 was the only tool to not mention the specific timestamp manipulation test case (TC09) as "suspicious". The newer version 1.9 of this same tool did not detect the manipulation, but it detected the execution of the nTimestomp executable, on a Windows Prefetch entry on the $UsnJrnl.

The Timestamp Analyzer tool managed to classify the TC09 file with "irregular time-stamps". However, this tool, provided an output with 678366 entries, where 52699 entries included the string "irregular time-stamps". Note that in the total number of entries (678366) of the output, none referred the files related TC07 and TC10.

6

CONCLUSIONS

This project, demonstrated that the timestamp behavior of file operations in Windows 11 v23H2 changed relating to the previous published version v22H2, as shown by the results of TC02, TC06, TC07 and TC08, when compared with the information by SANS (2024). It was also observed difference on the timestamp patterns observed in TC11 (A,B,C) and TC12 (A,B,C) when regarding the information in Galhuber and Luh (2021), and could be explained by the difference in the version of the OS, since in the authors tested them on version 2004 of Microsoft Windows 10 Education, build 19041.508.

Microsoft plays a crucial role in NTFS-based digital forensics, as any changes in NTFS's implementation or how the OS interacts with file timestamps significantly affects the accuracy of timestamp manipulation detection tools. Most importantly, the behavior of timestamps for files created or modified by Microsoft Office applications differs from the default behavior due to specific API and internal implementations.

Correlate file timestamps and file operations is essential to further improve findings in this field. As Bouma et al. (2023) mentioned, there is a "lack consensus of in literature on a canonical list of file operations to include for timestamp research", thus, creating automated environments to quickly verify these file operations on newer Windows versions is a great way to achieve a canonical list, where most used applications could be mapped, and further examined.

Also, previous $LogFile-based detection methods were based on the assumption that the $Standard_Information Creation timestamp never changed after the creation for a file (Oh et al., 2024), yet, as demonstrated in TC11 and TC12 that condition can and will happen in a normal OS operation (Oh et al., 2024). This assumption leads to a high number of false positives, or more dangerously, true negatives, since it could not detect TC09 in NTFS Log Tracker v1.71 and v1.9, for example.

The NTFS Log Tracker v1.9, shows improvements in the field of analyzing file systems artifacts and journals to detect timestamps manipulation. This tool combines several detection algorithms, combining the $MFT, $LogFile and $UsnJrnl to perform its detections, but also looks for entries in these log files containing references to Windows Prefetch and LNK files to detect the execution of timestamp manipulation tools.

Lastly, one of the challenges encountered during this project is that the documentation related with NTFS is old, and part of it was obtained performing reverse engineering in the Windows XP era, thus, after several Windows major versions, part of this information is outdated, as mentioned in Bouma et al. (2023).

6.1 *Future Work*

To address the challenges related with detecting timestamp manipulation, or timestomping, several key areas of future research and tool development can be pursued:

The derived results from the automated testing environment, such as the one developed in this project, can be used for designing detection patterns for various file operations and common applications, like Microsoft Office, to update existing forensic tools. These can then be tested and optimized for better accuracy in the detection of timestomping and other forms of timestamp manipulation. New tools or versions should be released regularly to reflect changes in Windows' implementation of file operations.

Also, machine learning presents a promising avenue for detecting patterns in log files that may indicate timestamp manipulation. By training models on known timestomping behavior, machine learning algorithms could help detect anomalies in large datasets more effectively than traditional signature-based methods. Future research could explore the integration of machine learning techniques into log file analysis, such as $MFT, $LogFile, and $UsnJrnl, looking for patterns that might indicate timestomping activity.

Finally, integrating the automated testing environment along with machine learning could significantly accelerate the deployment of trained models for identifying timestamp manipulation. Future projects could explore this method to enhance detection capabilities, streamline the analysis process, and adapt more efficiently to changes in new Windows versions.

BIBLIOGRAPHY

Bouma, Jelle et al. (2023). "Reconstructing Timelines: From NTFS Timestamps to File Histories." In: *Proceedings of the 18th International Conference on Availability, Reliability and Security*. ARES '23. Benevento, Italy: Association for Computing Machinery. DOI: 10.1145/3600160.3605027. URL: https://doi.org/10.1145/3600160.3605027.

Carrier, Brian (Jan. 2005). *File System Forensic Analysis*. ISBN: 0321268172.

Cho, Gyu-Sang (May 2013). "A computer forensic method for detecting timestamp forgery in NTFS." In: *Computers & Security* 34, pp. 36–46. ISSN: 0167-4048. DOI: 10.1016/j.cose.2012.11.003.

Cho, Gyu-Sang (Sept. 2019). "A Digital Forensic Analysis of Timestamp Change Tools for Windows NTFS." In: *Journal of the Korea Society of Computer and Information* 24.9, pp. 51–58.

Chow, K. P. et al. (n.d.). "The Rules of Time on NTFS File System." In: *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*. IEEE, pp. 10–12. DOI: 10.1109/SADFE.2007.22.

Cicero, M.T. and H. Rackham (1914). *De Finibus Bonorum Et Malorum*. Loeb classical library. W. Heinemann. ISBN: 9780674990449. URL: https://books.google.pt/books?id=C0ZuoEMeeeAC.

Elkhatib, Asmaa (June 2022). *Transaction Log*. [Online; accessed 23. Oct. 2023]. URL: https://forensafe.com/blogs/windowslogfile.html.

Galhuber, Michael and Robert Luh (Aug. 2021). "Time for Truth: Forensic Analysis of NTFS Timestamps." In: *ARES '21: Proceedings of the 16th International Conference on Availability, Reliability and Security*. New York, NY, USA: Association for Computing Machinery, pp. 1–10. ISBN: 978-1-45039051-4. DOI: 10.1145/3465481.3470016.

Galhuber, Michael and Robert Luh (July 2024). "Timestamp-based Application Fingerprinting in NTFS." In: *ACM Other conferences*. New York, NY, USA: Association for Computing Machinery, pp. 1–10. DOI: 10.1145/3664476.3670890.

Hermon, Rahul, Upasna Singh, and Bhupendra Singh (2023). "NTFS: Introduction and Analysis from Forensics Point of View." In: *2023 International Conference for Advancement in Technology (ICONAT)*, pp. 1–6. DOI: 10.1109/ICONAT57137.2023.10080271.

How the System Uses ACLs - NTFS.com (Sept. 2024). [Online; accessed 24. Sep. 2024]. URL: https://www.ntfs.com/ntfs-permissions-acl-use.htm.

Jang, Dae-il, Gail-Joon Hwang, and Kibom Kim (July 2016). "Understanding Anti-forensic Techniques with Timestamp Manipulation (Invited Paper)." In: pp. 609–614. DOI: 10.1109/IRI.2016.94.

Karresand, Nils Martin Mikael (2023). "Digital Forensic Usage of the Inherent Structures in NTFS." PhD thesis. Norway: NTNU. ISBN: 978-82-326-7045-1. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3069265.

Klyne, G. and C. Newman (July 2002). *Date and Time on the Internet: Timestamps*. [Online; accessed 21. Nov. 2023]. DOI: 10.17487/RFC3339.

Kratochvil, Jan (Nov. 2023). *Jan Kratochvil: Captive: The first free NTFS read/write filesystem for GNU/Linux*. [Online; accessed 7. Nov. 2023]. URL: https://www.jankratochvil.net/project/captive.

Lees, Christopher (Dec. 2013). "Determining removal of forensic artefacts using the USN change journal." In: *Digital Invest.* 10.4, pp. 300–310. ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.10.002.

limbenjamin (Sept. 2024). *nTimetools*. [Online; accessed 25. Sep. 2024]. URL: https://github.com/limbenjamin/nTimetools.

Lo, Edward et al. (2023). "Fuzzing the Latest NTFS in Linux with Papora: An Empirical Study." In: *arXiv*. DOI: 10.48550/arXiv.2304.07166. eprint: 2304.07166.

LSoft Technologies, Inc. (Mar. 2023a). *$EFS Attribute - NTFS.com*. [Online; accessed 14. Nov. 2023]. URL: http://ntfs.com/attribute-encrypted-files.htm.

LSoft Technologies, Inc. (Jan. 2023b). *NTFS File Types - Data Recovery Concept*. [Online; accessed 14. Nov. 2023]. URL: http://active-undelete.com/ntfs_file_types.htm#ntfs-compressed-files.

Microsoft (2007). *Performance Tuning Guidelines for Microsoft Services for Network File System*. URL: https://learn.microsoft.com/en-us/previous-versions/tn-archive/bb463205(v=technet.10)?redirectedfrom=MSDN.

Microsoft (Nov. 2023a). *File System Functionality Comparison - Win32 apps*. [Online; accessed 21. Nov. 2023]. URL: https://learn.microsoft.com/en-us/windows/win32/fileio/filesystem-functionality-comparison#limits.

Microsoft (Nov. 2023b). *FILETIME (minwinbase.h) - Win32 apps*. [Online; accessed 24. Nov. 2023]. URL: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime.

Microsoft (2023c). *fsutil behavior*. URL: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/fsutil-behavior.

Microsoft (Nov. 2023d). *How EFS Works*. [Online; accessed 14. Nov. 2023]. URL: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc962103(v=technet.10)?redirectedfrom=MSDN.

Microsoft (Nov. 2023e). *Managing Disk Quotas - Win32 apps*. [Online; accessed 14. Nov. 2023]. URL: https://learn.microsoft.com/en-us/windows/win32/fileio/managing-disk-quotas.

Microsoft (Oct. 2023f). *Master File Table (Local File Systems) - Win32 apps*. [Online; accessed 3. Oct. 2023]. URL: https://learn.microsoft.com/en-us/windows/win32/fileio/master-file-table.

Microsoft (Nov. 2023g). *NTFS overview*. [Online; accessed 13. Nov. 2023]. URL: https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview.

Microsoft (Nov. 2023h). *Overview of FAT, HPFS, and NTFS File Systems - Windows Client*. [Online; accessed 14. Nov. 2023]. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systems#ntfs-overview.

Microsoft (Sept. 2024a). *[MS-FSCC]: Zone.Identifier Stream Name*. [Online; accessed 23. Sep. 2024]. URL: https://learn.microsoft.com/en-us/openspecs/

windows_protocols/ms-fscc/6e3f7352-d11c-4d76-8c39-2516a9df36e8?redirectedfrom=MSDN.

Microsoft (Feb. 2024b). *Change Journal Records - Win32 apps*. [Online; accessed 19. Jun. 2024]. URL: https://learn.microsoft.com/en-us/windows/win32/fileio/change-journal-records.

Microsoft (June 2024c). *Overview of FAT, HPFS, and NTFS File Systems - Windows Client*. [Online; accessed 11. Jul. 2024]. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systems#ntfs-overview.

Microsoft (Sept. 2024d). *Windows 11 - release information*. [Online; accessed 23. Sep. 2024]. URL: https://learn.microsoft.com/en-us/windows/release-health/windows11-release-information.

Oh, Junghoon, Sangjin Lee, and Hyunuk Hwang (Dec. 2021). "NTFS Data Tracker: Tracking file data history based on $LogFile." In: *Forensic Science International: Digital Investigation* 39, p. 301309. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2021.301309.

Oh, Junghoon, Sangjin Lee, and Hyunuk Hwang (2024). "Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation." In: *IEEE Access*, pp. 1–1. DOI: 10.1109/ACCESS.2024.3395644.

Palmbach, David and Frank Breitinger (Apr. 2020). "Artifacts for Detecting Timestamp Manipulation in NTFS on Windows and Their Reliability." In: *Forensic Science International: Digital Investigation* 32, p. 300920. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2020.300920.

Russinovich, Mark (July 2016). *Sync - Sysinternals*. [Online; accessed 29. Sep. 2024]. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/sync.

SANS (Aug. 2024). *Windows Forensic Analysis | SANS Poster*. [Online; accessed 20. Aug. 2024]. URL: https://www.sans.org/posters/windows-forensic-analysis.

Torvalds, Linus (Nov. 2023). *Documentation - kernel/git/torvalds/linux.git - Linux kernel source tree*. [Online; accessed 7. Nov. 2023]. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/ntfs.rst.

Tuxera (Nov. 2023). *ntfs-3g*. [Online; accessed 7. Nov. 2023]. URL: https://github.com/tuxera/ntfs-3g.

APPENDIXES

APPENDIX A

This chapter demonstrate the PowerAutomate flows and subflows, which were used to test the test cases design in this project, and were not described earlier in the document.

In the creation and modifications of the testing files, the placeholder, often referred to as *lorem ipsum* (Cicero and Rackham, 1914), extract text used was :

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Vitae tortor condimentum lacinia quis. Varius quam quisque id diam vel. Id neque aliquam vestibulum morbi blandit cursus. Leo vel fringilla est ullamcorper eget nulla facilisi. Praesent semper feugiat nibh sed pulvinar proin gravida hendrerit lectus. Leo duis ut diam quam nulla porttitor massa. Dignissim convallis aenean et tortor at risus viverra. Purus sit amet volutpat consequat mauris nunc congue nisi. Nam aliquam sem et tortor. Lorem sed risus ultricies tristique nulla aliquet. Et tortor at risus viverra adipiscing at. Massa tincidunt dui ut ornare lectus. Est lorem ipsum dolor sit amet consectetur adipiscing elit pellentesque. Nullam eget felis eget nunc lobortis.

Id interdum velit laoreet id donec. Lacus sed viverra tellus in hac habitasse platea dictumst. Tempor orci eu lobortis elementum nibh tellus molestie. Orci phasellus egestas tellus rutrum tellus pellentesque eu. Interdum velit euismod in pellentesque massa placerat duis ultricies lacus. Scelerisque eu ultrices vitae auctor eu augue ut lectus. Mi in nulla posuere sollicitudin aliquam ultrices. Metus aliquam eleifend mi in. Quis enim lobortis scelerisque fermentum dui faucibus in ornare. Et malesuada fames ac turpis egestas. Tortor aliquam nulla facilisi cras. Lectus nulla at volutpat diam ut venenatis tellus in metus. Tristique senectus et netus et malesuada fames ac turpis egestas. Sed adipiscing diam donec adipiscing tristique risus nec feugiat in. Odio facilisis mauris sit amet massa vitae. Id donec ultrices tincidunt arcu non sodales. Eu lobortis elementum nibh tellus molestie."

The reason to use this text was to force the files to be non-resident on the MFT. A file is resident when the it's $DATA attribute is under 700 bytes in size, thus can be stored directly in the MFT. A non resident file is roughly over 700 bytes in size, thus, it is saved in external clusters (Carrier, 2005).

A.1 *Main*

In the figure below (A.1), the main flow of the testing environment is shown.

This flow includes all the other subflows (TC01- TC14) and Init - Populate Files.

Figure A.1: Main flow of the testing environment

Initially it uses the "Init - Populate Files", which is described in the next section (A.2), then the test subflows are executed, and finally, a powershell script to get the date at the time of execution is saved to `.\testing\testing.log` for debugging purposes.

A.2 *Init - Populate Files*

The "Init - Populate Files" subflow, creates the files and directories to be used in each test, then it runs the nTimeview to get the initial timestamps value of each file which stores the values in a log file. This subflow is shown by the figure A.2.

It saves the start date for error detection purposes (line 1 and 2).

Figure A.2: Initial population of files to the testing folder

The last action on this subflow, "Run PowerShell script", combines the usage of powerShell commands with the external tool, nTimeview.

The script is listed below:

```
$files = Get-ChildItem "C:\testing"
$out=""
foreach ($f in $files){
    $out+=$f.FullName+"`n"
    $aux= & nTimeview_v1.0_x64.exe $f.FullName
    $out=$out+$aux
}
$out | Out-File -FilePath "C:\testing\initial_file_timestamps.txt"
```

This script retrieves the $STANDARD_INFORMATION timestamps for every file and directory in the `C:\testing\` directory, with the usage of nTimeview, and stores the information on `initial_file_timestamps.txt`.

B

APPENDIX B

B.1 *Test Cases Results*

In this section the file timestamps obtained and used to perform the analysis in chapter 5 will be presented in tables.

Table B.1: TC01_creation.txt $STANDARD_INFORMATION timestamps.

Timestamps After Test
[M] 2024-09-26 16:22:30.7884497 UTC
[A] 2024-09-26 16:22:30.7884497 UTC
[E] 2024-09-26 16:22:30.7884497 UTC
[C] 2024-09-26 16:22:30.7083537 UTC

Table B.2: TC02_access.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:00.9696976 UTC	[M] 2024-09-26 16:20:00.9696976 UTC
[A] 2024-09-26 16:20:00.9696976 UTC	[A] 2024-09-26 16:22:36.9923747 UTC
[E] 2024-09-26 16:20:00.9696976 UTC	[E] 2024-09-26 16:22:37.0216540 UTC
[C] 2024-09-26 16:20:00.9696976 UTC	[C] 2024-09-26 16:20:00.9696976 UTC

Table B.3: TC03_modified.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:01.1385635 UTC	[M] 2024-09-26 16:23:13.0060289 UTC
[A] 2024-09-26 16:20:01.1385635 UTC	[A] 2024-09-26 16:23:13.0060289 UTC
[E] 2024-09-26 16:20:01.1385635 UTC	[E] 2024-09-26 16:23:13.0060289 UTC
[C] 2024-09-26 16:20:01.1385635 UTC	[C] 2024-09-26 16:20:01.1385635 UTC

Table B.4: TC04 files $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:01.3067680 UTC	[M] 2024-09-26 16:20:01.3067680 UTC
[A] 2024-09-26 16:20:01.3067680 UTC	[A] 2024-09-26 16:20:01.3067680 UTC
[E] 2024-09-26 16:20:01.3067680 UTC	[E] 2024-09-26 16:23:24.0451478 UTC
[C] 2024-09-26 16:20:01.3067680 UTC	[C] 2024-09-26 16:20:01.3067680 UTC

Table B.5: TC05 files $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:01.4712688 UTC	[M] 2024-09-26 16:20:01.4712688 UTC
[A] 2024-09-26 16:20:01.4712688 UTC	[A] 2024-09-26 16:23:36.7812531 UTC
[E] 2024-09-26 16:20:01.4712688 UTC	[E] 2024-09-26 16:20:01.4712688 UTC
[C] 2024-09-26 16:20:01.4712688 UTC	[C] 2024-09-26 16:23:36.7812531 UTC

Table B.6: TC06_local_move.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:01.6424035 UTC	[M] 2024-09-26 16:20:01.6424035 UTC
[A] 2024-09-26 16:20:01.6424035 UTC	[A] 2024-09-26 16:20:01.6424035 UTC
[E] 2024-09-26 16:20:01.6424035 UTC	[C] 2024-09-26 16:23:48.2888963 UTC
[C] 2024-09-26 16:20:01.6424035 UTC	[B] 2024-09-26 16:20:01.6424035 UTC

Table B.7: TC07_volume_file_move_cli.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:01.9822594 UTC	[M] 2024-09-26 16:20:01.9822594 UTC
[A] 2024-09-26 16:20:01.9822594 UTC	[A] 2024-09-26 16:23:50.8764403 UTC
[E] 2024-09-26 16:20:01.9822594 UTC	[E] 2024-09-26 16:20:01.9822594 UTC
[C] 2024-09-26 16:20:01.9822594 UTC	[C] 2024-09-26 16:23:50.8764403 UTC

Table B.8: TC08_volume_file_move_explorer.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:02.1500443 UTC	[M] 2024-09-26 16:20:02.1500443 UTC
[A] 2024-09-26 16:20:02.1500443 UTC	[A] 2024-09-26 16:24:01.7467705 UTC
[E] 2024-09-26 16:20:02.1500443 UTC	[C] 2024-09-26 16:20:02.1500443 UTC
[C] 2024-09-26 16:20:02.1500443 UTC	[B] 2024-09-26 16:20:02.1500443 UTC

Table B.9: TC09_timestamp_manipulation.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:02.3209943 UTC	[M] 1999-08-18 12:34:56.7890123 UTC
[A] 2024-09-26 16:20:02.3209943 UTC	[A] 1999-08-18 12:34:56.7890123 UTC
[E] 2024-09-26 16:20:02.3209943 UTC	[E] 1999-08-18 12:34:56.7890123 UTC
[C] 2024-09-26 16:20:02.3209943 UTC	[C] 1999-08-18 12:34:56.7890123 UTC

Table B.10: TC10_file_deletion.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:02.4924735 UTC	[M] 2024-09-26 16:20:02.4924735 UTC
[A] 2024-09-26 16:20:02.4924735 UTC	[A] 2024-09-26 16:20:02.4924735 UTC
[E] 2024-09-26 16:20:02.4924735 UTC	[E] 2024-09-26 16:20:02.4924735 UTC
[C] 2024-09-26 16:20:02.4924735 UTC	[C] 2024-09-26 16:20:02.4924735 UTC

Table B.11: TC11-A, TC11_excel.xlsx $STANDARD_INFORMATION timestamps.

Initial Timestamps
[M] 2024-09-26 16:20:08.1501885 UTC
[A] 2024-09-26 16:20:08.1501885 UTC
[E] 2024-09-26 16:20:08.1501885 UTC
[C] 2024-09-26 16:20:08.1366288 UTC

Table B.12: TC11-B, TC11_powerpoint.pptx $STANDARD_INFORMATION timestamps.

Initial Timestamps
[M] 2024-09-26 16:21:56.4546756 UTC
[A] 2024-09-26 16:21:56.4546756 UTC
[E] 2024-09-26 16:21:56.4705076 UTC
[C] 2024-09-26 16:21:56.2800904 UTC

Table B.13: TC11-C, TC11_word.docx $STANDARD_INFORMATION timestamps.

Initial Timestamps
[M] 2024-09-26 16:20:05.9530303 UTC
[A] 2024-09-26 16:20:05.9530303 UTC
[E] 2024-09-26 16:20:05.9667560 UTC
[C] 2024-09-26 16:20:05.9391918 UTC

Table B.14: TC12-A files $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:08.1501885 UTC	[M] 2024-09-26 16:24:14.7718085 UTC
[A] 2024-09-26 16:20:08.1501885 UTC	[A] 2024-09-26 16:24:14.7866099 UTC
[E] 2024-09-26 16:20:08.1501885 UTC	[E] 2024-09-26 16:24:14.7886629 UTC
[C] 2024-09-26 16:20:08.1366288 UTC	[C] 2024-09-26 16:24:14.7554900 UTC

Table B.15: TC12-B files $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:21:56.4546756 UTC	[M] 2024-09-26 16:25:17.9340794 UTC
[A] 2024-09-26 16:21:56.4546756 UTC	[A] 2024-09-26 16:25:17.9340794 UTC
[E] 2024-09-26 16:21:56.4705076 UTC	[E] 2024-09-26 16:25:17.9340794 UTC
[C] 2024-09-26 16:21:56.2800904 UTC	[C] 2024-09-26 16:25:17.7261695 UTC

Table B.16: TC12-C files $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:20:05.9530303 UTC	[M] 2024-09-26 16:24:18.3345581 UTC
[A] 2024-09-26 16:20:05.9530303 UTC	[A] 2024-09-26 16:24:18.3498332 UTC
[E] 2024-09-26 16:20:05.9667560 UTC	[E] 2024-09-26 16:24:18.3498332 UTC
[C] 2024-09-26 16:20:05.9391918 UTC	[C] 2024-09-26 16:24:18.3187748 UTC

Table B.17: TC13-A files $STANDARD_INFORMATION timestamps.

Timestamps After Test
[M] 2024-09-26 16:25:46.6367696 UTC
[A] 2024-09-26 16:25:46.6367696 UTC
[E] 2024-09-26 16:25:46.6367696 UTC
[C] 2024-09-26 16:25:45.1421384 UTC

Table B.18: TC13-B files $STANDARD_INFORMATION timestamps.

Timestamps After Test
[M] 2024-09-26 16:25:21.2218907 UTC
[A] 2024-09-26 16:25:21.2218907 UTC
[E] 2024-09-26 16:25:21.2218907 UTC
[C] 2024-09-26 16:25:21.2218907 UTC

Table B.19: TC14_file_system_tunneling.txt $STANDARD_INFORMATION timestamps.

Initial Timestamps	Timestamps After Test
[M] 2024-09-26 16:21:56.8913273 UTC	[M] 2024-09-26 16:25:54.3552898 UTC
[A] 2024-09-26 16:21:56.8913273 UTC	[A] 2024-09-26 16:25:54.3552898 UTC
[E] 2024-09-26 16:21:56.8913273 UTC	[E] 2024-09-26 16:25:54.3552898 UTC
[C] 2024-09-26 16:21:56.8913273 UTC	[C] 2024-09-26 16:21:56.8913273 UTC

B.2 *Timestamp Analyser*

In this section some entries of the output of this application will be listed. From the entries, only lastest entry related with each test cases will be listed below, since usually combines the information of the previous entries.
Here are the selected entries:

```
140939 .\testing\TC01_creation.txt (At 2024-AUGUST-13 11:3:6.1124601 UTC
    : Create)

139760 .\testing\TC02_access.txt (At 2024-AUGUST-13 11:3:12.0939622 UTC:
     Attribute change) <- (At 2024-AUGUST-13 11:3:12.0601791 UTC: Access
    with last access update enabled) <- (Between 2024-AUGUST-13
    11:1:59.2328756 UTC and 2024-AUGUST-13 11:3:12.0601791 UTC: Move in
    the same volume | File name change) <- (At 2024-AUGUST-13
    11:1:59.2328756 UTC: Create)

139761 .\testing\TC03_modified.txt (At 2024-AUGUST-13 11:3:44.7698498
    UTC: Update with last access update enabled) <- (At 2024-AUGUST-13
    11:1:59.4371611 UTC: Copy | Copy with last access update enabled |
    Copy with quirk)

139764 .\testing\TC04_rename_renamed.txt (At 2024-AUGUST-13
    11:3:52.9205317 UTC: Attribute change) <- (Between 2024-AUGUST-13
    11:1:59.6428147 UTC and 2024-AUGUST-13 11:3:52.9205317 UTC: Move in
    the same volume | File name change) <- (At 2024-AUGUST-13
    11:1:59.6428147 UTC: Create)

141865 .\testing\TC05_copy - Copy.txt (At 2024-AUGUST-13 11:4:4.4664426
    UTC: Copy with quirk) <- (At 2024-AUGUST-13 11:1:59.8484728 UTC:
    Create | Create with file tunneling | Update | Update with last
    access update enabled) possibly on other volume

139769 .\testing\TC06 (At 2024-AUGUST-13 11:4:17.0387721 UTC: Attribute
    change) <- (At 2024-AUGUST-13 11:4:17.0222688 UTC: Update directory)
    <- (At 2024-AUGUST-13 11:2:0.2544069 UTC: Copy | Copy with last
    access update enabled | Copy with quirk)

139773 .\testing\TC08_volume_file_move_explorer.txt (At 2024-AUGUST-13
    11:2:0.6817060 UTC: Create)

139776 .\testing\TC09_timestamp_manipulation.txt irregular time-stamps

140917 .\testing\TC11_excel.xlsx (At 2024-AUGUST-13 11:2:6.5329171 UTC:
    Create with file tunneling) <- (Between 2024-AUGUST-13 11:2:6.5329171
     UTC and 60056-MAY-28 5:36:10.9551615 UTC: Move in the same volume |
    File name change) <- (At 2024-AUGUST-13 11:2:6.5168472 UTC: Copy |
    Copy with last access update enabled | Copy with quirk)

141036 .\testing\TC11_powerpoint.pptx (At 2024-AUGUST-13 11:3:3.7408003
    UTC: Update with last access update enabled) <- (At 2024-AUGUST-13
    11:3:3.6621178 UTC: Copy with file tunneling) <- (Between 2024-AUGUST
    -13 11:3:3.6621178 UTC and 2024-AUGUST-13 11:3:3.7408003 UTC: Move in
     the same volume | File name change) possibly on other volume <- (At
    2024-AUGUST-13 11:3:3.5488657 UTC: Copy | Copy with last access
    update enabled | Copy with quirk)
```

```
139817 .\testing\TC11_word.docx irregular time-stamps

140916 .\testing\TC12_excel.xlsx irregular time-stamps

142660 .\testing\TC12_word.docx (At 2024-AUGUST-13 11:4:51.2741970 UTC:
    Attribute change) <- (At 2024-AUGUST-13 11:4:51.2721245 UTC: Access
    with last access update enabled) <- (Between 2024-AUGUST-13
    11:4:51.2578775 UTC and 2024-AUGUST-13 11:4:51.2721245 UTC: Move in
    the same volume | File name change) <- (At 2024-AUGUST-13
    11:4:51.2578775 UTC: Create)

149194 .\testing\TC13_word.pdf (At 2024-AUGUST-13 11:5:54.9894663 UTC:
    Create)

171309 .\testing\TC13_ms_print_to_pdf.pdf (At 2024-AUGUST-13
    11:6:24.5982962 UTC: Update with last access update enabled) <- (At
    2024-AUGUST-13 11:6:23.8023164 UTC: Copy with file tunneling) <- (
    Between 2024-AUGUST-13 11:6:23.8023164 UTC and 2024-AUGUST-13
    11:6:24.5982962 UTC: Move in the same volume | File name change)
    possibly on other volume <- (At 2024-AUGUST-13 11:6:23.6260091 UTC:
    Copy | Copy with last access update enabled | Copy with quirk)

113418 .\testing\TC14_file_system_tunneling.txt (At 2024-AUGUST-13
    11:6:32.8474872 UTC: Create with file tunneling) <- (Between 2024-
    AUGUST-13 11:6:32.8474872 UTC and 60056-MAY-28 5:36:10.9551615 UTC:
    Move in the same volume | File name change) <- (At 2024-AUGUST-13
    11:3:4.1761641 UTC: Copy | Copy with last access update enabled |
    Copy with quirk)
```

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Analysis of Timestamp Manipulation Detection tools"*, é original e foi realizado por Luís Paulo Monteiro dos Santos (2220749) sob orientação de Professor Doutor Miguel Cerdeira Marreiros Negrão, Professor Doutor Miguel Monteiro de Sousa Frade e Professor Doutor Patrício Rodrigues Domingues.

Leiria, 2024

Luís Paulo Monteiro dos Santos

Assinado por: Luís Paulo Monteiro dos Santos
Identificação: BI14936951
Data: 2024-09-30 às 11:38:40