Time for Truth: Forensic Analysis of NTFS Timestamps

Michael Galhuber
is171306@fhstp.ac.at
St. Pölten University of Applied Sciences
Austria

Robert Luh
robert.luh@univie.ac.at
University of Vienna &
St. Pölten University of Applied Sciences
Austria

ABSTRACT

Timeline forgery a widely employed technique in computer anti-forensics. Numerous freely available and easy-to-use tampering tools make it difficult for forensic scientists to collect legally valid evidence and reconstruct a credible timeline. At the same time, the large number of possible file operations performed by a genuine user can result in a wide variety of timestamp patterns that pose a challenge when reconstructing a chain of events, especially since application-specific discrepancies are often disregarded.

In this paper, we investigate timestamp patterns resulting from common user operations in NTFS, providing a much needed update to the Windows time rules derived from older experiments. We show that specific applications can cause deviations from expected behavior and provide analysts with a comprehensive set of behavioral rules for all permissible NTFS file operations. Finally, we analyze the effect and efficacy of 7 third party timestamp forgery tools as well as a custom PowerShell solution, and highlight forensic artifacts pointing at data falsification.

CCS CONCEPTS

• **Applied computing → System forensics**; **Investigation techniques**.

KEYWORDS

digital forensics, windows, NTFS, timestamps, anti-forensics

ACM Reference Format:
Michael Galhuber and Robert Luh. 2021. Time for Truth: Forensic Analysis of NTFS Timestamps. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3465481.3470016

1 INTRODUCTION

Forensic scientists rely on accurate timestamps to reconstruct a timeline of events for a filesystem under investigation [20]. Within a local operating system (OS), timestamps provide temporal information about file creation and modification, enable the tracking of move and copy operations within and across volumes, and can even help identify changes in file properties.

ARES 2021, August 17–20, 2021, Vienna, Austria

ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/3465481.3470016

To undermine the trust in file operation timelines, attackers seek to obscure their activities and make it as difficult as possible to find evidence of their malicious actions surrounding a specific incident. Due to the multitude of available and easy-to-use tools for file time manipulation, it has become increasingly common for attackers to falsify digital evidence by re-dating certain operations [25].

In order to spot such forgeries, forensic analysts need to understand the intricate rules of timestamp updates during normal user interaction and how tools might replicate them to hide malicious activity from the eyes of an investigator. For the Windows world in particular, timestamp management is highly complex and not consistently documented [6]. Existing resources [18] attempt to provide insight but often fail to consider application-specific deviations from the rules – or are simply outdated.

In this paper, we investigate the recognition of manipulated timestamps in Microsoft's New Technology File System (NTFS) and provide a detailed overview of the current time rules as used by version 2004 of Windows 10. As per Microsoft's information [23], timestamp updates are triggered when applications are creating, accessing, writing, or modifying files in various ways. Specifically, this occurs upon closing the handle which is associated to the respective file, which is generally done at the application's behest. Experiments in the past have shown [7, 8] that the creation and modification of files through Microsoft Office in particular does not follow the rules of the operating system or the filesystem [3] and may cause application-specific deviations, which are often disregarded in forensic sources.

This research contributes by a) providing an up-to-date set of time rules that describe possible filesystem operations in general as well as on an application-specific level, and by b) identifying the timestamp characteristics caused by the most widely employed timestamp forgery tools for analyst reference.

Specifically, we endeavor to answer the following questions:

- Are the time rules provided by forensic reference material still valid for current versions of Windows 10?
- Do user-side applications cause timestamp deviations and can timestamps be used to identify file types or apps?
- What are the practical limitations of timestamp forgery tools and how can they be detected?

For the purpose of answering these questions we have conducted a number of experiments using various file types such as text files, Office documents, PDF, and picture files. Every known NTFS file operation was executed and their metadata extracted from the master file table (MFT) for subsequent analysis. For timestamp falsification, we expanded on the work of Cho [9] and assessed 8 tools operating on user and kernel levels.

The remainder of this paper is structured as follows: Section 2 provides a brief overview of NTFS and its MFT structure in regards

to timestamps. Section 3 introduces information about the two main experiments and their setup, whereas Sections 4 and 5 present specific findings and the inferred time rules. Finally, Section 6 discusses related work, followed by concluding remarks.

2 BACKGROUND

Microsoft has released version 3.1 of NTFS in 2001 [22]. The filesystem itself has remained unchanged since, with the exception of the journaling logfile format, which has been updated accompanying the release of Windows 8.1. The main premise of NTFS is that every piece information is held in files, which applies to both files and their metadata [6]. The central repository for NTFS file records, which includes all kinds of timestamps, is the so-called master file table (MFT).

2.1 Master File Table

The MFT is located in the root directory of the file system. In the MFT, there is at least one entry for each file and directory – the so-called file record or *MFT entry*. The size per entry is specified in the boot sector; for Microsoft NTFS implementations it is fixed at 1024 bytes, whereas the first 42 bytes contain the MFT entry header [1]. The rest of the record is used to store different attributes, which come with their own attribute header. Figure 1 provides a schematic overview.

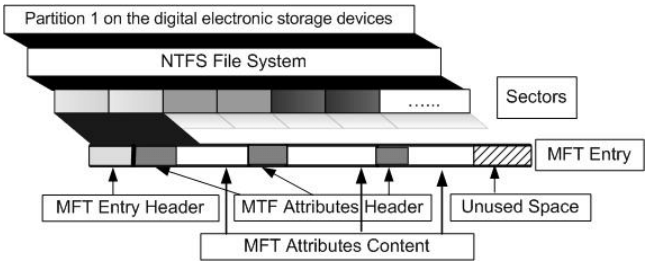

Figure 1: Structure of NTFS Master File Table [1]

There are many standard MFT attributes in the NTFS file system. Each attribute has its own structure and contains different data. Their identifiers, names and sizes are defined in the file system metadata file *$AttrDef*. With regard to timestamp analysis, the most important attributes are *$STANDARD_INFORMATION* ($SI) and *$FILE_NAME* ($FN) [26].

The $SI attribute [6] holds basic metadata for a file or a directory. It contains information about the owner, security ID, and flags describing general file properties. The size of this attribute is set to 48 bytes in current Windows versions. Within the $FN attribute, we can find information about the file or directory names as well as the reference to its parent directory, which can be useful to determine the absolute file path. The size of $FN is variable; it has a length of 66 bytes plus the length of the file name.

NTFS supports four namespaces – one for Windows, DOS 8.3, and POSIX names, as well as one for an additional shared namespace for congruent DOS and Windows names. Therefore, a file record can have a maximum of three $FN attributes [25]. Besides this specific information for files and directories, each attribute – $SI and $FN – contains four timestamps, which represent the primary target of attack for timestamp manipulation tools. If more than one $FN attribute exists, they all hold the same time values.

2.2 NTFS Timestamps

All timestamps [23] in $SI and $FN are calculated from January 1, 1601, 12:00 A.M. They always represent Coordinated Universal Time (UTC). Therefore they are not influenced by time zones or daylight saving time (DST). NTFS timestamps have an accuracy in the hundreds of nanoseconds (see also Figure 2).

The 4 available timestamps in $SI and $FN are called *MACB*[1] and provide the time value for the following specific events:

- **Modified**: last update of file content
- **Accessed**: last access to file content
- **Changed**: last change of file metadata (change in MFT entry)
- **Birth**: file creation time (creation time of MFT entry)

Both attributes keep MACB time information. Timestamps within the attribute content of $SI are starting at an offset of 0x00 while in the $FN offset is 0x08. Each field of a time value has a size of 8 bytes, which results in a total of 32 bytes per attribute across all timestamps.

The temporal values for modification (M), access (A) and creation (B) of a file or directory are easily accessible trough the 'Properties' dialog of the Windows Explorer shell. These particular timestamps are retrieved from the values stored in the $SI attribute. The change time of a file's metadata (C) is generally hidden and not presented to the user.

The timestamps contained within $FN do not implicitly hold the same values as the ones in $SI and are typically not visible to the user. If a file is created from scratch, all timestamps in $SI and $FN will be set to the same time. However, while the modification of timestamps in $SI can be triggered by any file operation, the timestamps in $FN cannot be updated directly [6]. Instead, they are overwritten by the OS with the respective values from $SI whenever there are changes in the $FN attribute itself, namely when creating, copying, moving, or renaming files or directories [25]. Forgery-wise, it is significantly more difficult to change $FN timestamps than their user-level $SI counterpart due to a lack of API support.

2.3 Timestamp Forgery

Generally, we can distinguish between two timestamp tampering methods: One approach utilizes the Windows API, while the other method requires direct disk access [25].

The Windows API only allows for the modification of $SI timestamps through the functions `SetFileTime`, `NtSetInformationFile`, and `ZwSetInformationFile`, respectively. They enable the modification of the `FILE_BASIC_INFORMATION` structure, which contains all necessary timestamp information that is found within the $SI attribute [9].

There is no direct way to modify $FN timestamps through the Windows API. While it is possible to move the target file in order to force the OS to overwrite current values with the ones from $SI, a more established approach utilizes direct disk access: Here,

[1] *MACB* is also referred to as *MACE* in some literature: *M*odification, *A*ccess, *C*reation and *E*ntry Modified.

changes to the temporal values are written directly to the MFT. This technique is fairly invasive and can cause inconsistencies that may result in a corrupted file system [25].

We classify forgery tools by their methods and approach, inspired by the work of Cho [9]. Tool selection is based on the papers by both Cho and Palmbach and Breitinger [27]:

- **Basic forgery tools**: These tools use the `SetFileTime` function from the Windows API for timestamp tampering. They are limited to altering M, A, and B times in $SI. Examples include *FileTouch*, *NewFileTime*, *SKTimeStamp*, *BulkFileChanger*, *Change Timestamp*, and *eXpress TimeStamp Toucher*. Basic tools are limited to modifying full-second values when writing timestamps due to API function limitations [24].
- **Intermediate forgery tools**: Intermediate tools utilize `NtQueryFile` and `NtSetInformationFile` for their purposes [15]. This allows them to alter all 4 timestamps within the $SI attribute. The prime examples are *Timestomp* [9] and *nTimestomp* [19]. The latter is able to write timestamps that are accurate to the hundreds of nanoseconds.
- **Advanced forgery tools**: In our context, advanced tools are able to write both $SI and $FN attributes using direct disk access. Certain OS features such as 'PatchGuard' may interfere with the respective tools, which prominently includes *SetMace* [27] by Joakim Schicht.

Please refer to Section 5 for more information about the tools employed in our experiments.

3 METHODOLOGY

This section provides an overview of the setup as well as the timestamp extraction process. The testing procedure used for our two experiments is detailed at the beginning of the respective section.

Observations in the performed experiments and the interpretative evaluation of the resulting timestamps follow the principle of qualitative research. The experimental data collection and analysis enables the verification of existing knowledge from previous studies. Moreover, new knowledge is generated by taking into account the current state of the art at the time of this study. Existing results of other researchers are taken into account and compared with our own findings.

3.1 Testing Environment

All experiments were conducted on a VMware virtual machine running version 2004 of Microsoft Windows 10 Education, build 19041.508. The operating system is installed on a 60 GB system volume on virtual disk 1 (HDD1), which is not involved in the testing. Two additional disk drives (HDD2 & HDD3) with 1 GB in size each hold the NTFS test volumes used for performing file operations. All hard drives are stored in VMDK format.

The registry value `NtfsDisableLastAccessUpdate`, which influences whether $SI.A timestamps are tracked, was left at its default setting for the duration of our experiments. For this version of Windows, this means that the 'accessed' time is managed by the system and disabled by default [11]. This means that $SI.A is not tracked. In practice, however, several applications still caused changes in this timestamp.

For use in our experiments, we installed Microsoft Office 365 in addition to several required Visual C++ runtime libraries. For all other file operations we utilized Windows standard applications such as PowerShell, Notepad, Paint, and the Office PDF printer.

3.2 Timestamp Extraction

The timestamps of the examined files are extracted directly from the $SI and $FN attributes of the MFT. The whole procedure is performed automatically by a custom PowerShell script for both non-system disks, which combine a number of functions that implement the following extraction process:

(1) Calculation of an initial SHA256 hash for all hard drives,
(2) Creation of a copy of the original disk to avoid working with the original,
(3) Recalculation of the disk hash and comparison to the original, which ensures that no changes were made to the original disks during analysis (forensic integrity),
(4) Extraction of the partition table using `mmls` from TSK [5],
(5) Specification of the start sector for the data partition,
(6) Extraction of the MFT in raw format using `icat`,
(7) Another recalculation & comparison of the disk hash,
(8) Parsing of the raw MFT file with *Mft2Csv* [28]. The tool transforms the information from hexadecimal notation into a human-readable format and automatically creates a CSV file for further analysis,
(9) Extraction of the overall timestamp pattern using a custom script (see below).

All timestamps are represented in the format shown in Figure 2.

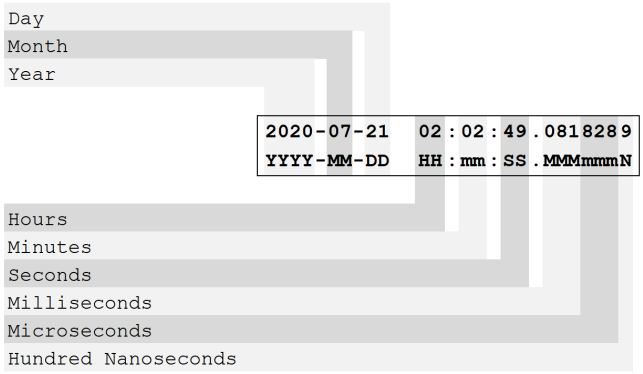

Figure 2: Timestamp format for analysis

For determining the overall timestamp pattern, all four timestamps of each attribute are analysed by a separate script that parses the CSV file and sorts the timestamps for each MFT record by their value, attribute name, as well as timestamp designation [25].

For each experiment, our tool determines a timestamp pattern that compares the extracted values. Said pattern may look as follows: $\$SI.B = \$SI.M < \$FN.A = \$FN.B = \$FN.C = \$FN.M < \$SI.C < \$SI.A$, whereas $\$SI$ and $\$FN$ represent the MFT's $\$STANDARD\_INFORMATION$ and $\$FILE\_NAME$ attributes and M, A, C, and B refer to the respective modified, access, MFT changed, and born timestamps. The pattern, as well as a log file documenting the process, is stored for analysis and reproducibility.

4 PERMISSIBLE TIMESTAMP PATTERNS

The first experiment is intended to create representative patterns for all valid NTFS file operations that can later be used as baseline when looking for falsified timestamps. This section discusses the testing procedure and experiment results.

4.1 Procedure

The general process encompasses a) running the timestamp extraction script introduced in Section 3, b) executing a desired file operation, c) running the timestamp extractor again, d) comparing the timestamps, and e) assessing the resulting timestamp pattern to create a rule ('fingerprint') for that particular scenario.

File operations executed in our experiments include file creation, access, modification, renaming, copying, moving locally and across volumes, as well as file deletion [18]. Numerous combinations of applications and file operation triggers were tested. Each file operation resulted in a specific timestamp pattern, from which rules can be derived. We defined the scope of the experiment to include some of the most common user file types, namely plain text (TXT), Microsoft Office documents (DOCX, XLSX, PPTX), portable document format files (PDF), and image files (PNG, GIF, JPG).

The file operations themselves are triggered via PowerShell, through a client application, or via the Windows Explorer shell and were repeated at least 5 times at random intervals for each case (e.g., for resident and non-resident files). This allows us to compare the system's behavior in different scenarios and to check whether time stamp behavior is deterministic in identical conditions. To establish rules for timestamp changes after a file was created we define and track the following parameters:

- **Timestamp before file operation $\mathbf{t_i}$**,
- **Timestamp after file operation $\mathbf{t_f}$**,
- **Operation time $\mathbf{t_{op}}$** in UTC format,
- **Timestamp values**: Individual values for birth (B), modification (M), MFT entry change (C) and access (A),
- **Processing time** Δ required to complete an operation.

If a timestamp remains unchanged, it is specifically denoted (**U**). For example, the rule:

$$\$SI(U^{\mathrm{B}}, U^{\mathrm{M}}, {t_{\mathrm{op}}}^{\mathrm{C}}, {t_{\mathrm{op}}}^{\mathrm{A}} + \Delta) \wedge \$FN(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}})$$

...which represents a PDF file access operation via Microsoft Edge and tells the analyst that \$SI.B, \$SI.M, and all of the \$FN attribute remain unchanged. However, \$SI.C is set to time t_{op} the operation was launched, while \$SI.A additionally adds the required processing time Δ to the launch time value, something that does not happen when e.g., opening text files in Notepad.

4.2 Results

This section summarizes our experimental results for all permissible file operations in NTFS. Please refer to Table 3 for a summary of patterns. Note that we – for brevity's sake – consider different timestamps to be identical if the deviation is less than 2 milliseconds (see 'File Creation' for an example of a simplified pattern). Refer to the link in the Appendix to download full-precision patterns.

In the following, we focus on explaining our test cases as well as the most important (majority) patterns derived from our experiments.

4.2.1 File Creation. The following cases were investigated for file creation (i.e., born) operation:

(B1) Text files via PowerShell script. (a) Empty files (resulting in \$DATA-resident files), and (b) files with a size >1024 bytes (non-resident).
(B2) Text files via Notepad, 'File > Save as' dialog. (a) Resident and (b) non-resident files.
(B3) Microsoft Office (a) Word, (b) Excel, and (c) PowerPoint files via 'File > Save as'. Non-resident files only.
(B4) PDF files via (a) Microsoft Print to PDF in Microsoft Word and Notepad, as well as (b) 'File > Save as > PDF' in Word. Only non-resident files with the exception of the Notepad vector.
(B5) PNG, JPG, and GIF files via Microsoft Paint 'File > Save'. Non-resident files only.
(B6) Microsoft Office (a) Word, (b) Excel, (c) PowerPoint files, as well as (d) Text files via Windows Explorer 'Context menu > New > File type'

Timestamp analysis for case (B1)(a) shows equal timestamps for all MACB values, which results in the below timestamp pattern:

(B1)(a): $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.A = \$SI.B = \$SI.C = \$SI.M$

This is in line with the information provided by SANS DFIR [18]. Things get a bit more complex if content is written to the files. In that case, all \$FN timestamps are set to the file's creation time as stored in \$SI.B. In 4 out of 10 tested cases \$SI.C and \$SI.M hold that same value. For six of the files the two timestamps were equal but clocked in around 1 millisecond later than \$SI.B. The value for file access \$SI.A was always set last.

(B1)(b), 40% of cases: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B = \$SI.C = \$SI.M < \$SI.A$

(B1)(b), 60% of cases: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B < \$SI.C = \$SI.M < \$SI.A$

The deviation between \$SI.B and \$SI.C can occur if a temporary file is created before the file content is written. The resulting gap then correlates to the time required for writing the file content to disk. This process is comparable to a multiple-step operation including file creation, file renaming, and file modification.

Since the time difference between \$SI.C/\$SI.M and \$SI.B is zero or at least very small, we decided to harmonise both patterns into the following representation by introducing a 2-millisecond tolerance. This value corresponds to the average deviation of the disk/filesystem operations observed in our experiments and helps limit the number of corner cases:

(B1)(b) simplified: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \cong (\$SI.C = \$SI.M) < \$SI.A$

Case (B2) resulted in 5 different timestamp patterns, two of which only occurred in files with non-resident \$DATA attributes. All evaluated timestamp patterns have the same time values for MACB of \$FN and \$SI.B. In half the cases \$SI.M holds the same data as well. However the sequence of \$SI.M, \$SI.A and \$SI.C can vary. Simplifying the resulting rules, we arrive at:

(B2) simplified: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \cong \$SI.M < (\$SI.A, \$SI.C)$

Case (B3) resulted in a total of 7 patterns. Interestingly, most of them can be correlated to the use of a specific Office application. For example, 3 patterns are unique to PowerPoint, 2 only occur for Excel files, and one case is limited to Word documents. Only one pattern occurred in both Word and Excel files. This gives rise to the assumption that timestamp analysis can provide an alternative means of identifying file types independent from their magic number.

A more detailed look at the patterns shows that the two most recently adapted timestamps are always $SI.A and $SI.C in a different order or at the same time. When providing a file name during the saving operation, the time value for $SI.C is adjusted. Since the file is still opened, its content is accessed at the same time, influencing the value for $SI.A. Depending on the processing time of the test steps performed – saving the file and closing the application – the two time values and their sequence vary.

If the two timestamps for $SI.A and $SI.C are disregarded and only the first six time values from the patterns are considered, we can reduce the high number of cases to three simplified patterns:

(B3)(a,b) simplified: $\$FN.B = \$SI.B < \$FN.A = \$FN.M = \$SI.M < \$FN.C < (\$SI.A, \$SI.C)$

(B3)(b) simplified: $\$FN.B = \$SI.B < \$FN.A = \$FN.C = \$FN.M = \$SI.M < (\$SI.A, \$SI.C)$

(B3)(c) simplified: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B < \$SI.M < (\$SI.A, \$SI.C)$

For case (B4), 5 different patterns were identified. It was accurately possible to distinguish the 'print to PDF' and 'save as PDF' approach, but not the application the saving operation was triggered from. The most common pattern, which resulted from the use of the PDF printer, is identical to pattern (B1)(b) associated to non-resident text files. The runner-up pattern, which is unique to the 'save as PDF' function from within Microsoft Word, is identical to the one for creating XLSX files.

Case (B5), which includes the creation of picture files, yielded 3 distinct patterns that are not directly associated to one of the file types but rather represent the application used (Paint) and how it handles the saving process.

(B5) simplified: $\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \leq \$SI.M < (\$SI.A, \$SI.C)$

Creating files via the Windows Explorer context menu (case (B6)) returned a total of 5 patterns, 3 of which are unique to Excel files. The most common pattern resulted from text, Word, and PowerPoint files, where all patterns, witch the exception of $SI.C, hold the same value. Using this information, we can create two simplified patterns:

(B6)(a,c,d) simplified: $\$FN.A = \$FN.B = \$FN.M = \$SI.A = \$SI.B = \$SI.M \cong \$FN.C < \$SI.C$

(B6)(b) simplified: $\$FN.B = \$SI.B < \$FN.A = \$FN.M = \$SI.M < \$FN.C < \$SI.C \wedge \$FN.B = \$SI.B < \$FN.A = \$FN.M \leq \$SI.A$

4.2.2 File Access. For this part of the experiment, files were opened with their associated default application: (A1) Notepad and (A2) PowerShell's `get-content` function for text files, the 3 aforementioned (A3) Office applications, (A4) Microsoft Photos for pictures, and (A5) Microsoft Edge for PDF documents. The only method that did not alter any timestamp when reading file content was PowerShell.

With the exception of the PowerShell vector, the overall result slightly deviates from the information provided by SANS. In addition, when accessing a file in Notepad, the $SI.C timestamp was set to the time of the operation instead of $SI.A. Furthermore, opening a PDF file with Microsoft Edge caused the access time to be written to $SI.C, while $SI.A was altered only after closing the file.

From these findings, the following rules for the adaptation of timestamps can be derived:

(A1): $\$SI(U^{\mathrm{B}}, U^{\mathrm{M}}, {t_{\mathrm{op}}}^{\mathrm{C}}, U^{\mathrm{A}}) \wedge \$FN(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}})$

(A2): $\$SI(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}}) \wedge \$FN(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}})$

(A3,A4): $\$SI(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, {t_{\mathrm{op}}}^{\mathrm{A}}) \wedge \$FN(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}})$

(A5): $\$SI(U^{\mathrm{B}}, U^{\mathrm{M}}, {t_{\mathrm{op}}}^{\mathrm{C}}, {t_{\mathrm{op}}}^{\mathrm{A}} + \Delta) \wedge \$FN(U^{\mathrm{B}}, U^{\mathrm{M}}, U^{\mathrm{C}}, U^{\mathrm{A}})$

4.2.3 File Modification. In this part, the effects on file times due to changes in file content are examined. For this experiment, PDF files were not considered. In summary, the $SI.M, A, and C times of all examined files change if the file content is modified. Furthermore, Microsoft Office applications also trigger an update for the MAC time values in $FN. In conflict with the information provided by SANS, $SI.A is in fact changed upon modifying the file.

The 4 resulting rules, which can be found in Table 3, can be used to distinguish text files (both Notepad and PowerShell's `Add-Content` function), DOCX/XLSX, PPTX, and image files/Microsoft Paint.

4.2.4 File Rename. In this experiment, files of all aforementioned types were renamed using two methods: the `Rename-Item` function of PowerShell, and the file context menu in Windows Explorer. With each method, a set of three test files per extension was processed.

For text files that were previously created via PowerShell only $SI.C is updated upon renaming. This is in line with SANS DFIR. However, Cho [8] discovered that all timestamps from $SI will be copied to $FN before the renaming happens and that the time value for $SI.C will be updated to reflect the time of operation. To verify this behaviour, an additional experiment was performed. Here, the test files were modified after creation to obtain a varied pattern of initial timestamps. After changing the file names, the results show a clearer picture of what is happening in this process: It could be identified that, except for $SI.A, all other time values of $SI will indeed be copied to $FN before the file is renamed. Nevertheless, $FN.A is set to the initial value of $SI.C. Refer to Table 3 for the rule derived from this behavior.

For Office files, we have seen marked differences between renaming via PowerShell command and through Windows Explorer. While the timestamp update behaviour via PowerShell is very similar to text files, the changing of file names in Windows Explorer is handled differently. Specifically, the value for $SI.A will be changed to the starting time of the renaming process [3], while $SI.C is set to the point in time when the renaming has been completed. There is no well-defined behavior for the update of $FN.A; analysis shows that it will be set to a certain time before the resulting $SI.A, respectively t_{op}. The experiments showed a time gap between $FN.A and $SI.A ranging from 8 milliseconds up to 3 minutes.

The comparison of timestamp update behaviour for different image files and Microsoft Office documents was largely identical.

For PDFs, all files resulted in the same timestamp pattern regardless of the renaming method used. The initial values for $SI.C were copied to $FN.C, while $SI.C was set to the renaming operation time. The timestamps for last access remained unchanged.

4.2.5 File Copy. For this experiment, we conducted a number of copy operations within the same volume (from one folder to another) as well as across volumes using the following 4 approaches for each of the file extensions:

(C1) PowerShell `Copy-Item` function
(C2) Windows Explorer context menu
(C3) CTRL+C, CTRL+V keyboard shortcuts
(C4) Windows Explorer drag and drop

Regardless of the copy method and destination volume, all newly created destination files show the same pattern of timestamps. The MACB values of $FN and the $SI.B timestamp will be set to the operation time of the copy process. The timestamp $SI.A is either set to the same value, or to a value taking into account the duration of the copy process. The time values for $SI.M and $SI.C will be inherited from the source file.

Dst. file: $\$SI({t_{\text{op}}}^{\text{B}}, {t_{\text{i}}}^{\text{M}}, {t_{\text{i}}}^{\text{C}}, {t_{\text{op}}}^{\text{A}}+\Delta) \wedge \$FN({t_{\text{op}}}^{\text{B}}, {t_{\text{op}}}^{\text{M}}, {t_{\text{op}}}^{\text{C}}, {t_{\text{op}}}^{\text{A}})$

For timestamp changes to the source file, we were able to identify two distinct cases: (a) non-TXT files are being copied, and (b) text files are copied using the copy/paste keyboard shortcuts. The following source file rules can be derived from these findings:

(C1,C2,C4)(b): $\$SI(U^{\text{B}}, U^{\text{M}}, U^{\text{C}}, U^{\text{A}}) \wedge \$FN(U^{\text{B}}, U^{\text{M}}, U^{\text{C}}, U^{\text{A}})$
(C1-4)(a),(C3)(b): $\$SI(U^{\text{B}}, U^{\text{M}}, U^{\text{C}}, \cong {t_{\text{op}}}^{\text{A}}) \wedge \$FN(U^{\text{B}}, U^{\text{M}}, U^{\text{C}}, U^{\text{A}})$

Comparing our destination file results to the SANS DFIR reference shows us a deviation in $SI.C. According to SANS, this time value is always set to the operation time. However, this could not be corroborated with any of the test files during our experiments.

Source file timestamps are not considered by SANS at all.

4.2.6 File Move. Moving files used the same 4 methods as copying: PowerShell (`Move-Item` function), context menu, shortcut (CTRL+X, CTRL+V), and drag and drop. When moving files locally, the MFT record number of the file itself does not change. Only the reference to its parent directory will be modified to point to the value representing the destination directory. The timestamp for $SI.C will be set to the operation time. Since local moving causes changes in the $FN attribute, all timestamps will be overwritten by the original time values of $SI, which is behavior comparable to rename operations.

According to SANS, only the timestamp for $SI.C changes when files are moved on the local volume. All other timestamps remain unchanged. This statement only holds true if all timestamps in the source file have the same value. Refer to Table 3 for the resulting pattern.

Moving files across volumes is behaviorally different. The process of moving creates a completely new file resulting in a new MFT entry on the destination volume. At the same time, the MFT record on the source volume is marked as deleted. This results in very unique timestamp characteristics: All timestamps of the $FN attribute are equated to the operation time. The value of $SI.A is set to that same point in time. All other times of $SI will be inherited from the original file. This timestamp updating behaviour was observed for all investigated methods except for moving a file via PowerShell command. Here, the value of $SI.B will be set to the operation time of the moving process instead. This behavior is consistent with SANS. See Table 3 for specific patterns/rules.

4.2.7 File Deletion. In this experiment we sought to determine if timestamps change upon file deletion. Three approaches were investigated: (D1) PowerShell's `Remove-Item` function, (D2) the Explorer context menu, and (D3) the DEL keyboard button. Prior to our experiments, we disabled the recycle bin and deletion dialog.

Corresponding to SANS' information, no timestamps were altered upon deletion; the MFT entry itself will merely be marked with the 'delete' flag.

4.3 Discussion

From our experiments we can derive a number of key takeaways: Firstly, widely used forensic sources are in fact outdated and need to be brought up to speed to be of help in Windows 10 investigations. Since application-specific deviations are largely unexplored, it can be assumed that the SANS reference in particular was created through scripted events (e.g. PowerShell) using text files and did not attempt to replicate operations with user-side applications.

Secondly, We show that the choice of application – ranging from the Explorer shell to a particular PDF printer – does impact timestamp update patterns and might even be used to distinguish between certain applications. If correlated with information about an app's file handle manipulation, this insight could open up numerous additional artifacts helpful in forensic investigations.

Lastly, our experiments have shown that certain properties have less impact than initially assumed. For example, it typically matters little whether a file is resident or not. On the other hand, it was confirmed that access timestamps need to be regarded with caution, as they may be manipulated by various applications even if the registry value for `NtfsDisableLastAccessUpdate` is configured to omit file access tracking.

5 TIMESTAMP FORGERY

This second experiment focuses on timestamp forgery. Here, we determine the limitations of the tools listed in Table 1 and assess whether these tools can be used to forge patterns that feign valid file operations.

5.1 Procedure

We decided to focus on resident and non-resident text files created via PowerShell; initial experimentation did not reveal any impact on the forgery process that could be ascribed to the file type. The timestamps of these files were altered with each of the third-party tools listed, in addition to a custom PowerShell script.

5.2 Results

5.2.1 Basic forgery tools. Tools in this class are able to alter the timestamps for $SI.B, $SI.M, and $SI.A. Specifically, the tool *NewFileTime* can change the values for date and the time separately. When both date and time are modified, timestamp accuracy is limited to full seconds (case 1 in Table 2). If only the date of a timestamp is changed, the millisecond value is retained and all subsequent digits

Class	Application	Version	Usage	Developer	Release	Source
Basic	NewFileTime	4.61	GUI	SoftwareOK	Oct. 18, 2020	[14]
	SKTimeStamp	1.3.5	GUI	Stefan Küng	Nov. 19, 2016	[17]
	BulkFileChanger	1.71	GUI	NirSoft	Apr. 4, 2020	[30]
	eXpress TimeStamp Toucher (XTST)	1.1.0	GUI	Irnis Haliullin	Dec. 14, 2004	[13]
	TimeStampForger (custom PowerShell script)	1.0	CLI	Paper author	Oct. 24, 2020	-
Intermediate	Timestomp	-	CLI, GUI	James C. Foster, Vincent Liu	2005	[21]
	nTimestomp	1.1	CLI	Benjamin Lim	Oct. 2, 2019	[19]
Advanced	SetMace	1.0.0.16	CLI	Joakim Schicht	Nov. 10, 2014	[29]

Table 1: Examined timestamp forgery tools

are set to zero (case 2). Furthermore, the tool makes it possible to increase or decrease the values of the timestamps by a certain amount of time through its 'be older/be younger' function. In this case, all decimal places are retained from the original value (case 3).

Very similar functionality and behaviour can be observed when using *BulkFileChanger*. Here, the timestamp accuracy is limited to full seconds when date and time are both changed (1). *BulkFileChanger* can add a specified period to the existing time value. When this feature is used or if only the date of a timestamp is altered, the resulting values adopt the milliseconds of the original time while the remaining digits are zeroed (2).

XTST also shows comparable results. If only the date is changed, the milliseconds are retained and all remaining digits are deleted. If the time is changed as well, accuracy is again reduced to milliseconds. However, the first three decimal places are set to an unspecified value, which is not derived from the original time.

SKTimeStamp is implemented as a separate register in the file properties of Windows Explorer, thereby causing additional changes to the operating system. Date and time can only be changed together, which results in a full-second accuracy.

Lastly, we developed a PowerShell script motivated by the work of Brinkmann [4], which allows setting the file times for $SI.B, $SI.M and $SI.A with a default accuracy in the hundreds of nanoseconds. This custom timestamp manipulation tool was dubbed *TimeStampForger*. It successfully eliminated the limitations of the aforementioned basic tools, as is evident in Table 2.

5.2.2 Intermediate forgery tools. This class of tools provides the ability to additionally change $SI.C, which is usually hidden from the user. In their experiment, Gungor [12] determined that the tool *Timestomp* is incompatible with the high-resolution timestamps in NTFS. This information has been corroborated by our findings: the timestamps manipulated by *Timestomp* are truncated to the full second. This weakness has been addressed in *nTimestomp* [19], which has the ability to alter all $SI timestamps with a hundred nanosecond precision.

5.2.3 Advanced forgery tools. Currently, only *SetMace* offers $FN modification capabilities. This is achieved by writing time values directly into the attributes of the MFT, bypassing the Windows API altogether. If file times are falsified to plausible values, it is not possible to prove manipulation by solely looking at the timestamps and their patterns. However, the execution of the tool requires administrative privileges and is thwarted by Windows security features such as 'PatchGuard'. Specifically, modern NT-based systems block direct write access within volume space, which either mandates unmounting the volume prior to the forgery operation (disqualifying it for use on live system disks), needs a signed disk driver, or requires a crack bypassing PatchGuard [29].

5.3 Discussion

Determining the authenticity of timestamps is largely reliant on a tool's ability to manipulate the digits after the full second. It is never sufficient to only consider the file properties dialog when investigating timestamps – they need to be extracted directly from the MFT.

Since many timestamp forgery tools truncate decimal places and thus reduce the accuracy of timestamps, their manipulation can be detected fairly easily. Nevertheless, analysts should consider that there is a false-positive probability of 0.00001% in respect to full-second timestamps. For tools that round to full milliseconds, this probability increases to 0.01%. Cloned digits, where the source file provides the original value, require additional scrutiny. In general, caution is advised when files are assessed only for their timestamp accuracy, since older file systems such as FAT do not support the same precision as NTFS [16]. If files were copied or moved from e.g., a FAT volume, their timestamps will have their decimal places zeroed, adding to the risk of a false positive.

Only the advanced forgery tool *SetMace* is able to access both $SI and $FN timestamps. Through these, it is possible to replicate any valid NTFS operation. The tool comes with a number of prerequisites, however, which limits its use in productive environments.

6 RELATED WORK

Initial research by Chow et al. [10] highlighted the significance of a credible timeline to reconstruct the sequence of criminal activities on a computer system. At the same time, the authors emphasized the risk of relying on existing file timestamps because of possible forgery. Their experiments, which were limited to traditional MAC time values (translating to $SI.M, $SI.A, and $SI.B), were based on seven hypothetical rules that describe the characteristics of timestamps that resulted from different operations. Each of these rules was verified through a total of 10 experiments conducted on a Microsoft Windows XP machine. Bang et al. [2] expanded on this work by considering $FN attributes and full MACB timestamps, as well as directories in addition to files.

Bang et al. [3] analyzed timestamp changes for different versions of Microsoft Windows, ranging from Windows 2000 up to Windows 7 Professional. In addition to what was investigated in previous work, they considered resident and non-resident $DATA

Forgery tool	Scenario	Time	$SI.B	$SI.M	$SI.C	$SI.A
NewFileTime	Case (1)	t_i	2020-10-26 11:58:30.3001650	2020-10-26 11:58:30.3001650	2020-10-26 11:58:30.3001650	2020-10-26 11:58:30.5792567
		t_f	2020-10-02 17:20:36.0000000	2020-10-03 08:27:18.0000000	2020-10-26 12:06:39.2050896	2020-10-04 09:01:16.0000000
	Case (2)	t_i	2020-10-26 11:59:57.4002406	2020-10-26 11:59:57.4015944	2020-10-26 11:59:57.4015944	2020-10-26 11:59:57.6813132
		t_f	2019-01-07 11:59:57.4000000	2019-01-14 11:59:57.4010000	2020-10-26 12:08:30.0020255	2019-10-18 11:59:57.6810000
	Case (3)	t_i	2020-10-26 12:00:28.4841608	2020-10-26 12:00:28.4841608	2020-10-26 12:00:28.4841608	2020-10-26 12:00:28.7611742
		t_f	2021-02-03 14:00:28.4841608	2021-02-03 14:00:28.4841608	2020-10-26 12:09:34.3303623	2021-02-03 14:00:28.7611742
SKTimeStamp	Case (1)	t_i	2020-10-25 20:03:32.0390209	2020-10-25 20:03:32.0390209	2020-10-25 20:03:32.0390209	2020-10-25 20:03:32.0390209
		t_f	2020-10-08 10:03:32.0000000	2020-10-16 15:03:32.0000000	2020-10-25 21:01:57.9953407	2020-10-05 01:01:48.0000000
BulkFileChanger	Case (1)	t_i	2020-10-25 20:08:43.1979316	2020-10-25 20:08:43.1979316	2020-10-25 20:08:43.1979316	2020-10-25 20:08:43.1979316
		t_f	2020-10-30 07:08:03.0000000	2020-10-12 08:07:06.0000000	2020-10-25 21:17:09.2613599	2020-10-08 07:08:06.0000000
	Case (2)	t_i	2020-10-26 12:26:55.9725132	2020-10-26 12:26:55.9725132	2020-10-26 12:26:55.9725132	2020-10-26 12:26:56.1947129
		t_f	2021-06-23 12:26:55.9720000	2020-10-26 14:26:55.9720000	2020-10-26 13:07:56.3810327	2020-10-26 12:27:56.1940000
XTST	Case (1)	t_i	2020-10-26 13:00:01.7684886	2020-10-26 13:00:01.7695362	2020-10-26 13:00:01.7695362	2020-10-26 13:00:02.0359502
		t_f	2025-03-01 13:00:01.7680000	2025-03-02 06:48:16.9460000	2020-10-26 13:09:49.5140233	2025-03-05 17:25:57.9460000
TimeStampForger	Case (1)	t_i	2020-10-26 22:08:17.8029475	2020-10-26 22:08:17.8029475	2020-10-26 22:08:17.8029475	2020-10-26 22:08:18.0771853
		t_f	2022-04-01 00:00:00.1234567	1998-03-15 14:21:22.0102034	2020-10-26 22:28:18.7508309	2020-11-11 11:11:11.1111111

Table 2: Timestamp changes caused by basic forgery tools

attributes in their experiments. They observed that the timestamp update process may be dependent on certain applications, thereby laying the groundwork for this study. Microsoft Word in particular was identified as culprit due to its use of temporary and backup files. Furthermore their research highlighted the differences when updating $FN values in different Windows versions.

Ding and Zou [11] endeavored to detect file time manipulation through cross-reference analysis. They used a self-developed template containing MACB time rules and correlated that with temporal information from the Windows registry. This enabled them to find evidence for illegal file access or manipulation as well as timestamp forgery. In their study, they discussed the issue of missing countermeasures for anti-forensic tools stemming from a lack of documentation for NTFS time rules. The influence of the application was confirmed for a limited number of cases, prompting them to develop three kinds of updating rules for portable executable (PE) files, documents, and directories.

In his 2013 work, Cho [7] emphasized the difficulty of determining the authenticity of timestamps. He determined a number of permissible timestamp patterns generated by seven different file operations. Cho limited his research to document file types with the extension TXT, DOCX, and PDF to develop timestamp patterns that should allow distinguishing between real and tampered timestamps. To find traces for file time manipulation, he additionally investigated the journal file *$LogFile* which, like the MFT, is one of the core metadata files in NTFS.

Minnaard et al. [25] focused on identifying filesystem tampering by regarding the parent directory indices contained in each MFT entry. These directories, which in turn hold information about their child entries, can be used to identify inconsistencies caused by manipulation tools that take advantage of direct disk access. Based on this information, Minnaard et al. created a fingerprint for unaltered executable files.

In 2016, Jang et al. [16] defined a set of 14 regular timestamp patterns resulting from normal file operations. To identify possible manipulation, they used the NTFS journal file for context and considered information from deleted files. The authors first identified the truncation of timestamps to be an indicator for forgery and tested their hypothesis on 3 different falsification tools.

All of the above studies were conducted using versions of Microsoft Windows that are no longer supported. As one of the most recent publications, SANS DFIR [18] published a first overview for Windows 10 time rules, but omitted information about methodology or specific experiments. Furthermore, our own experiments have shown that there are deviations from the rules presented by SANS in a number of situations, which should be considered when doing a forensic investigation.

7 CONCLUSION

The structured experiments discussed in this paper have highlighted a number of time rules that have changed in modern versions of Microsoft Windows, providing forensic analysts with an overdue update for their established reference material. Copy operations in particular have been shown to exhibit hitherto unknown behavior in regards to their $SI.C value. Furthermore, several previously disregarded cases have been identified, mostly concerning operation times that are factored into timestamps, as well as the heavy influence of user-side applications on the operating system's updating rules. Significantly expanding earlier experiments, we have identified many applications that are responsible for unique new patterns, for which we have created a set of rules summarized in Table 3. Our research lays the groundwork for larger-scale application fingerprinting through timestamp analysis on the local system.

For timestamp forgery, we could show the differences between, and limitations of, 3 classes of manipulation tools. Only one out of 8 evaluated tools was able to freely falsify the full range of NTFS timestamps, thereby eluding all attempts at immediate detection. For such low-level changes, artifacts pointing at the presence of the forgery tool itself as well as its prerequisite software (e.g. drivers, cracks) will have to be investigated in addition to MFT metadata. A custom solution for basic timestamp forgery via PowerShell, which eliminates all drawbacks of comparable tools, was presented as well.

Timestamp forgery remains an issue for forensic analysts when dealing with skilled opponents. However, our results have shown that the avenue of application-specific timestamp analysis is both viable and promising in modern Windows environments.

REFERENCES

[1] Mamoun Alazab, Sitalakshmi Venkatraman, and Paul Watters. 2009. Effective digital forensic analysis of the NTFS disk image. *Ubiquitous Computing and Communication Journal* 4, 1 (2009), 551–558.

[2] Jewan Bang, Byeongyeong Yoo, Jongsung Kim, and Sangjin Lee. 2009. Analysis of time information for digital investigation. (2009), 1858–1864.

[3] Jewan Bang, Byeongyeong Yoo, and Sangjin Lee. 2011. Analysis of changes in file time attributes with file manipulation. *digital investigation* 7, 3-4 (2011), 135–144.

[4] Martin Brinkmann. [n. d.]. How to edit timestamps with Windows PowerShell - gHacks Tech News. https://www.ghacks.net/2017/10/09/how-to-edit-timestamps-with-windows-powershell/ Accessed Nov. 11, 2020.

[5] Brian Carrier. [n. d.]. The Sleuth Kit. https://sleuthkit.org/sleuthkit/ Accessed Nov. 11, 2020.

[6] Brian Carrier. 2010. File System Forensic Analysis. Addison-Wesley Professional, 273–396.

[7] Gyu-Sang Cho. 2013. A computer forensic method for detecting timestamp forgery in NTFS. *Computers & Security* 34 (2013), 36–46.

[8] G. S. Cho. 2014. An Intuitive Computer Forensic Method by Timestamp Changing Patterns. In *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. 542–548.

[9] Gyu-Sang Cho. 2019. A Digital Forensic Analysis of Timestamp Change Tools for Windows NTFS. *Journal of the Korea Society of Computer and Information* 24, 9 (2019), 51–58.

[10] Kam-Pui Chow, Frank YW Law, Michael YK Kwan, and Pierre KY Lai. 2007. The rules of time on NTFS file system. In *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*. IEEE, 71–85.

[11] Xiaoqin Ding and Hengming Zou. 2011. Time Based Data Forensic and Cross-reference Analysis. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11)*. ACM, New York, NY, USA, 185–190. https://doi.org/10.1145/1982185.1982227

[12] Arman Gungor. [n. d.]. Date Forgery Analysis and Timestamp Resolution. https://www.meridiandiscovery.com/articles/date-forgery-analysis-timestamp-resolution Accessed Nov. 11, 2020.

[13] Irnis Haliullin. [n. d.]. eXpress TimeStamp Toucher. http://www.irnis.net/soft/xtst/ Accessed Nov. 11, 2020.

[14] Nenad Hrg. [n. d.]. NewFileTime 4.61 Corrections and manipulation of timestamp. http://www.softwareok.com/?seite=Microsoft/NewFileTime Accessed Nov. 11, 2020.

[15] Hamid Jahankhani, Gianluigi Me, David Watson, and Frank Leonhardt. 2010. Handbook of Electronic Security and Digital Forensics. 417. https://doi.org/10.1142/7110

[16] D. Jang, G. A. H. Hwang, and K. Kim. 2016. Understanding Anti-forensic Techniques with Timestamp Manipulation (Invited Paper). In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. 609–614. https://doi.org/10.1109/IRI.2016.94

[17] Stefan Küng. [n. d.]. SKTimeStamp - Stefans Tools. https://tools.stefankueng.com/SKTimeStamp.html Accessed Nov. 11, 2020.

[18] Rob Lee. 2019. Cyber Security Resources | SANS Institute. https://www.sans.org/security-resources/posters/windows-forensic-analysis/170/download Accessed Nov. 11, 2020.

[19] Benjamin Lim. [n. d.]. GitHub - limbenjamin/nTimetools: Timestomper and Timestamp checker with nanosecond accuracy for NTFS volumes. https://github.com/limbenjamin/nTimetools Accessed Nov. 11, 2020.

[20] Xiaodong Lin. 2018. *Timeline Analysis*. Springer International Publishing, Cham, 257–269. https://doi.org/10.1007/978-3-030-00581-8_12

[21] Joachim Metz. [n. d.]. Timestomp - Forensics Wiki. https://forensicswiki.xyz/wiki/index.php?title=Timestomp Accessed Nov. 11, 2020.

[22] Microsoft. 2009. NTFS Technical Reference. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc758691(v=ws.10) Accessed Feb. 02, 2021.

[23] Microsoft. 2018. File Times. https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times Accessed Nov. 11, 2020.

[24] Microsoft. 2018. SetFileTime function (fileapi.h). https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime Accessed Nov. 11, 2020.

[25] Wicher Minnaard, CTAM de Laat, and M van Loosen MSc. 2014. Timestomping NTFS. (2014). https://delaat.net/rp/2013-2014/p48/report.pdf

[26] Sebastian Neuner, Artemios G Voyiatzis, Martin Schmiedecker, Stefan Brunthaler, Stefan Katzenbeisser, and Edgar R Weippl. 2016. Time is on my side: Steganography in filesystem metadata. *Digital Investigation* 18 (2016), 76–86.

[27] David Palmbach and Frank Breitinger. 2020. Artifacts for Detecting Timestamp Manipulation in NTFS on Windows and Their Reliability. *Forensic Science International: Digital Investigation* 32 (2020), 300920.

[28] Joakim Schicht. [n. d.]. GitHub - jschicht/Mft2Csv: Extract $MFT record info and log it to a csv file. https://github.com/jschicht/Mft2Csv Accessed Nov. 11, 2020.

[29] Joakim Schicht. 2019. GitHub - jschicht/SetMace: Manipulate timestamps on NTFS. https://github.com/jschicht/SetMace Accessed Nov. 11, 2020.

[30] Nir Sofer. [n. d.]. BulkFileChanger: Change date/time/attributes of multiple files. http://www.nirsoft.net/utils/bulk_file_changer.html Accessed Nov. 11, 2020.

A ONLINE RESOURCES

Additional detail tables as well as all scripts referenced in this paper are available for download here:
https://1drv.ms/u/s!Av5ytJNNiT_I6kZzmCCqqXhk-0jG?e=KexNxU

File operation	Timestamp update rule[a]	Simplified pattern[b]	File type							
			TXT	DOCX	XLSX	PPTX	PNG	GIF	JPG	PDF
Creation	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.A = \$SI.B = \$SI.C = \$SI.M$	X							X
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \cong \$SI.M < (\$SI.A, \$SI.C)$	X							
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \cong (\$SI.C = \$SI.M) < \$SI.A$	X							
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B < \$SI.A = \$SI.C = \$SI.M$								X
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B < \$SI.C = \$SI.M < \$SI.A$								X
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B < \$SI.M < (\$SI.A, \$SI.C)$				X				
	n/a	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.B \leq \$SI.M < (\$SI.A, \$SI.C)$					X	X	X	
	n/a	$\$FN.A = \$FN.B = \$FN.M = \$SI.A = \$SI.B = \$SI.M \cong \$FN.C < \$SI.C$	X	X		X				
	n/a	$\$FN.B = \$SI.B < \$FN.A = \$FN.C = \$FN.M = \$SI.C = \$SI.M < \$SI.A$								X
	n/a	$\$FN.B = \$SI.B < \$FN.A = \$FN.C = \$FN.M = \$SI.M < \$SI.C < \$SI.A$								X
	n/a	$\$FN.B = \$SI.B < \$FN.A = \$FN.C = \$FN.M = \$SI.M < (\$SI.A, \$SI.C)$			X					
	n/a	$\$FN.B = \$SI.B < \$FN.A = \$FN.M = \$SI.M < \$FN.C < \$SI.C$ $\$FN.B = \$SI.B < \$FN.A = \$FN.M \leq \$SI.A$			X					
	n/a	$\$FN.B = \$SI.B < \$FN.A = \$FN.M = \$SI.M < \$FN.C < (\$SI.A, \$SI.C)$		X	X					
Access	$\$SI(U^B, U^M, t_{op}{}^C, t_{op}{}^A + \Delta)$ $\$FN(U^B, U^M, U^C, U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B, \$SI.M) < \$SI.C < \$SI.A$								X
	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(U^B, U^M, U^C, U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C$	X							
	$\$SI(U^B, U^M, U^C, t_{op}{}^A)$ $\$FN(U^B, U^M, U^C, U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B, \$SI.C, \$SI.M) < \$SI.A$		X	X	X	X	X	X	
	$\$SI(U^B, U^M, U^C, U^A)$ $\$FN(U^B, U^M, U^C, U^A)$	$\$FN.A = \$FN.B = \$FN.C = \$FN.M = \$SI.A = \$SI.B = \$SI.C = \$SI.M$	X							
Modification	$\$SI(U^B, t_{op}{}^M, t_{op}{}^C, t_{op}{}^A + \Delta)$ $\$FN(U^B, U^M, U^C, U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B) < \$SI.C = \$SI.M < \$SI.A$	X							
	$\$SI(U^B, t_{op}{}^M, t_{op}{}^C + \Delta, t_{op}{}^A)$ $\$FN(U^B, t_{op}{}^M, t_{op}{}^C + \Delta, t_{op}{}^A)$	$(\$FN.B, \$SI.B) < \$FN.A = \$FN.M = \$SI.A = \$SI.M < (\$FN.C, \$SI.C)$				X				
	$\$SI(U^B, t_{op}{}^M, t_{op}{}^C + \Delta, t_{op}{}^A + \Delta)$ $\$FN(U^B, t_{op}{}^M, t_{op}{}^C + \Delta, t_{op}{}^A)$	$(\$FN.B, \$SI.B) < \$FN.A = \$FN.M = \$SI.M) < (\$FN.C, \$SI.A, \$SI.C)$		X	X					
	$\$SI(U^B, t_{op}{}^M, t_{op}{}^C + \Delta, t_{op}{}^A + \Delta)$ $\$FN(U^B, U^M, U^C, U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B) < \$SI.M < (\$SI.A, \$SI.C)$					X	X	X	
Renaming	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], t_f{}^A \approx \$SI[t_i{}^C])$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C)$					X	X	X	
	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], t_f{}^A \cong \$SI[t_i{}^A \wedge t_i{}^C])$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C)$		X	X	X				
	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], t_f{}^A = \$SI[t_i{}^C])$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C)$	X							
	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], U^A)$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C)$								X
	$\$SI(U^B, U^M, t_{op}{}^C + \Delta, t_{op}{}^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], \approx t_{op}{}^A)$	$(\$FN.B, \$FN.C, \$FN.M, \$SI.B, \$SI.M) < (\$FN.A, \$SI.A, \$SI.C)$		X	X	X	X	X	X	
Copying	$\$SI(t_{op}{}^B, t_i{}^M, t_i{}^C, t_{op}{}^A + \Delta^c)$ $\$FN(t_{op}{}^B, t_{op}{}^M, t_{op}{}^C, t_{op}{}^A)$	$(\$SI.C, \$SI.M) < (\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B) \leq \$SI.A$	X	X	X	X	X	X	X	X
Moving (local)	$\$SI(U^B, U^M, t_{op}{}^C, U^A)$ $\$FN(\$SI[t_i{}^B, t_i{}^M, t_i{}^C], t_f{}^A \cong \$SI[t_i{}^C])$	$(\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.A, \$SI.B, \$SI.M) < \$SI.C)$	X	X	X	X	X	X	X	X
Moving (volume)	$\$SI(t_i{}^B, t_i{}^M, t_i{}^C, t_{op}{}^A + \Delta^d)$ $\$FN(t_{op}{}^B, t_{op}{}^M, t_{op}{}^C, t_{op}{}^A)$	$(\$SI.B, \$SI.C, \$SI.M) < (\$FN.A, \$FN.B, \$FN.C, \$FN.M) \leq \$SI.A$	X	X	X	X	X	X	X	X
	$\$SI(t_{op}{}^B, t_i{}^M, t_i{}^C, t_{op}{}^A + \Delta)$ $\$FN(t_{op}{}^B, t_{op}{}^M, t_{op}{}^C, t_{op}{}^A)$	$(\$SI.C, \$SI.M) < (\$FN.A, \$FN.B, \$FN.C, \$FN.M, \$SI.B) \leq \$SI.A$	X	X	X	X	X	X	X	X
Deletion	$\$SI(U^B, U^M, U^C, U^A)$ $\$FN(U^B, U^M, U^C, U^A)$	n/a[e]	X	X	X	X	X	X	X	X

[a] Timestamp update rules cannot be applied to file creation since the initial pattern is created during this operation.
[b] The simplified pattern for file access, modification, renaming, copying and moving is derived from the determined timestamp update rule.
[c] $\Delta \geq 0$
[d] $\Delta \geq 0$
[e] The derivation of a simplified pattern for file deletion is not applicable since the timestamps are not changed during deletion.

Table 3: Overview of permissible timestamp patterns and updating rules