Forensic Science International: Digital Investigation 32 (2020) 300920

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 EU – Proceedings of the Seventh Annual DFRWS Europe

Artifacts for Detecting Timestamp Manipulation in NTFS on Windows and Their Reliability

David Palmbach [a], Frank Breitinger [a, b, *]

[a] Cyber Forensics Research and Education Group (UNHcFREG), Tagliatela College of Engineering, ECECS, University of New Haven, 300 Boston Post Rd., West Haven, CT, 06516, USA

[b] Hilti Chair for Data and Application Security, Institute of Information Systems, University of Liechtenstein, Fürst-Franz-Josef-Strasse, 9490, Vaduz, Liechtenstein

ARTICLE INFO

Article history:

Keywords:
Timestamp manipulation
Forgery
$Logfile
$USNJrnl
SetMACE
nTimestomp
Timestomping
Anti-forensics

ABSTRACT

Timestamps have proven to be an expedient source of evidence for examiners in the reconstruction of computer crimes. Consequently, active adversaries and malware have implemented timestomping techniques (i.e., mechanisms to alter timestamps) to hide their traces. Previous research on detecting timestamp manipulation primarily focused on two artifacts: the $MFT as well as the records in the $LogFile. In this paper, we present a new use of four existing windows artifacts – the $USNjrnl, link files, prefetch files, and Windows event logs – that can provide valuable information during investigations and diversify the artifacts available to examiners. These artifacts contain either information about executed programs or additional timestamps which, when inconsistencies occur, can be used to prove timestamp forgery. Furthermore, we examine the reliability of artifacts being used to detect timestamp manipulation, i.e., testing their ability to retain information against users actively trying to alter or delete them. Based on our findings we conclude that none of the artifacts analyzed can withstand active exploitation.

1. Introduction

As computer crimes become more prevalent and sophisticated, forensic examiners rely heavily on meta-data such as timestamps during their investigations (Buchholz and Spafford, 2004; Koen and Olivier, 2008). Due to their importance, and the fact that it is relatively easy to alter timestamps with current (open source) tools, the reliability of this evidence has been repeatedly called into question in the court of law (Hannon, 2018). The integrity of timestamps has become an increasingly important issue, while on the other hand detecting timestamp manipulation is not trivial. Sophisticated groups and state actors commonly add timestomping capabilities into their malware which is well-documented by MITRE (b). Hacking groups such as the Lazarus group for North Korea (Novetta, 2016), Fancy Bear for Russia (Alperovitch, 2016), Threat Group 3390 for China (Works, 2015), Copy Kittens for Iran (Sky, 2017) and many more, have been documented using it.

Problem description. While practitioners are aware of the possibility of timestamp forgery, there is a limited amount of peer-reviewed literature on detecting timestamp manipulation as well as the reliability of the artifacts being used to accomplish that. If the artifact that is being used to identify timestamp manipulation can be manipulated itself (or deleted), then it becomes less valuable to an examiner. For instance, Jang et al. (2016) found that it is not possible to detect timestamp manipulation by only analyzing the values in the $MFT and subsequent methods were needed. However, the authors limited their research to one additional source (the $LogFile) to detect timestamp forgery. Thus, at the time of writing this paper, the two primary sources of information used for identifying timestamp manipulation are the $MFT and $LogFile. This lack of diversity is a problem for practitioners as it is possible for an active adversary to hide/obfuscate evidence on a system and avoid detection. Lastly, there is also only a confined amount of research on the timestomping tools currently available and methods for detecting their use on a system.

Research Questions. These current challenges led us to the following three research questions:

* Corresponding author. Cyber Forensics Research and Education Group (UNHcFREG), Tagliatela College of Engineering, ECECS, University of New Haven, 300 Boston Post Rd., West Haven, CT, 06516, USA.

E-mail addresses: DPalmbach@gmail.com (D. Palmbach), Frank.Breitinger@uni.li (F. Breitinger).

URL: http://www.FBreitinger.de

https://doi.org/10.1016/j.fsidi.2020.300920

RQ1. What is the range of artifacts that can be used by examiners to identify timestamp manipulation in NTFS on Windows?
RQ2. How reliable are the artifacts (i.e., resilient to obfuscation or deletion) that are being used for timestamp manipulation detection?
RQ3. Besides identifying timestamp manipulation directly, can we detect the execution/presence of timestamp forgery tools?

Contribution. This paper has two major contributions. First, we expand the list of known artifacts that can be used for timestamp forgery detection. While existing literature mostly focused on the `$MFT` and the `$LogFile`, we present a new use of four existing windows artifacts/methods for detecting timestamp manipulation in NTFS. Secondly, we analyze the reliability of those four artifacts as well as the `$LogFile`, i.e., the possibilities to delete or obfuscate them anyway. For testing, we ran several experiments using three prominent timestomping tools: Timestomp, SetMACE, and nTimestomp. Lastly, as a minor contribution, we highlight some peculiarities for these tools as well as present methods to identify their presence on a system.

Overview. The remainder of this paper is organized as follows: The Background section provides basic information about NTFS, timestamps, and timestomping tools. and is followed by the Previous work section. The apparatus and methodology of our experiment will be discussed in Sec. 4. The core of this article is Sec. 5 which presents our findings organized by artifact. The last two sections include the discussion and our conclusion.

2. Background

Before covering the various methods that can be used to identify timestamp manipulation, there are several key concepts and terms that the reader should become familiar with. In the following we provide a brief summary of NTFS (New Technology File System), for more details refer to Carrier (2005). A list of commonly used acronyms for this article can be seen in Table 1. Readers familiar with NTFS (i.e., terminology and files related to timestamps) may skip this section.

NTFS relies on the `$MFT` which is a database containing a comprehensive list of all files and folders on the volume. It reserves the first 16 entries for Windows system files which can be identified by the $ at the beginning of their names. They are protected and hidden from the user by default (they are essential to the operating system and are not intended to be altered by a user). Two of these protected files, the `$LogFile` and the `$USNjrnl`, keep track of changes made to files on the volume and can be valuable sources of information for an examiner trying to detect timestamp manipulation.

Table 1
Summary of acronyms used throughout the article.

Acronym	Actual file
SIA	`$STANDARD_INFORMATION` attribute
FNA	`$FILE_NAME` attribute
`$MFT`	Master file table
LNK	Link file
PF	Prefetch file
`$LogFile`	Log file
`$USNjrnl`	`$UsnJrnl:$J`
MACE	Modified, Accessed, Created, MFT Changed

Table 2
File system timestamps (MACE).

Timestamp	Information
Last **M**odified	When the file's contents were modified.
Last **A**ccessed	When the file was last accessed.
File **C**reation	When the file was created, copied, or changed directories.
MFT **E**ntry Changed	When the file's `$MFT` record was updated.

Each record in the `$MFT` contains several attributes that are used to organize a file's content and meta-data. With respect to timestamps, there are two particular attributes that keep track of that information for every file. The first is the `$STANDARD_INFORMATION` attribute (SIA) which contains four unique timestamps that we collectively refer to as MACE[1] and are described in Table 2. We will use an abbreviated writing to refer particular changes, e.g., SIA-M represents the 'last modified timestamp in the SIA. The second is the `$FILE_NAME` attribute (FNA) which contains the name of the file, the name of the parent directory, and has its own set of the MACE timestamps. The timestamps in the FNA only update when one of the other attributes stored in the FNA is changed as well, such as the file name or its location on the drive. Consequently, the `$MFT` stores eight different timestamps for each file on the volume; four in the SIA and four in the FNA. It is important to note that NTFS stores each timestamp as a 64-bit value representing the number of nanoseconds that have passed since January 1, 1601 UTC (Windows, 2018).

2.1. Modification of SIA/FNA

The findings on SIA/FNA modification presented in the following are based on analyzing the three tools – Timestomp, SetMACE, and nTimestomp. We are currently not aware of any other open source tools that can alter timestamps in NTFS. Note, we do not cover external manipulation techniques (i.e., manipulating a hard drive of a turned-off computer) or altering files not located on the system drive.

To alter the four SIA timestamps, tools use the `NtSetInformationFile` (API) which can access and write to all four of them (Schicht, 2018; Lim, 2019; Microsoft, 2018c; Minnaard et al., 2014). Specifically, the API allows a user to set any of the SIA MACE values to a 64-bit value of their choice. For our experiment, we assume that the user making the alterations to the timestamps has a good understanding of NTFS timestamp rules[2] and thus there is no possibility to detect timestamp manipulation by only looking at the timestomped values in SIA or FNA (e.g., setting the SIA-C to 01/01/1910 at 14:15 EST; or setting milliseconds to all 0).

The timestamps in the FNA are set to mirror the SIA timestamps when the file is created and cannot be altered directly. However, FNA values are updated to mirror the SIA values whenever the file is renamed or changes location on the drive. Thus, Schicht (2014), the author of SetMACE, uses a combination of SIA changes and moving as depicted in Fig. 1 to get FNA updated: First the SIA timestamps are altered, the file is moved to a different directory causing the timestamps in the FNA to update and mirror the changes made to the SIA. The file has to be timestomped again before being relocated back to its original location because the FNA values will be updated again. Finally, the files SIA values are timestomped one last time to ensure they are set exactly how the user wants them. This method allowed for the timestamps stored in the SIA and the FNA to be modified with nanosecond precision making the detection of timestamp manipulation in NTFS difficult.

[1] Note, sometimes also referred to as MACb where b stands for birth.
[2] Here a rule is a logical behavior of timestamps, e.g., "When the SIA-M time is equal to the SIA-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." (Chow et al., 2007).

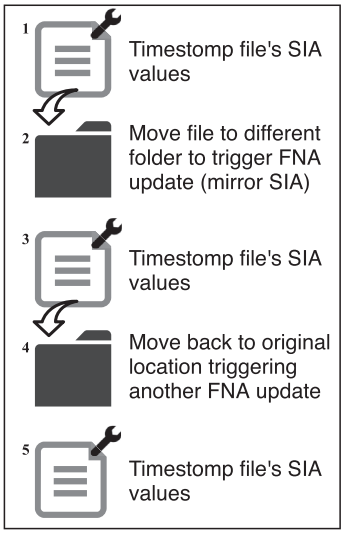

Fig. 1. Process to modify FNA.

Later, Schicht (2014) realized that this method may leave additional evidence behind on the system and thus rewrote SetMACE to write directly to the system drive for versions 1.0.0.6 and newer. However, Microsoft (2018a) patched this and currently direct access to the system drive is no longer allowed. Hence, we went back to SetMace 1.0.0.5 which uses the API allowing it to alter files on the system drive.

Another method for altering timestamps is the `GetFileTime` and `SetFileTime` commands. The `GetFileTime` command can be used to retrieve the MACE timestamps of any file the malicious user wants and then the `SetFileTime` command can be used to copy them to any file. This method requires write access to the file just as the `NtSetInformationFile` command does and it also cannot alter the FNA timestamps (Carvey, 2014). The previously discussed method for altering FNA values by changing the file's directory should also work with these commands although we did not test it.

2.2. Peculiarities of timestomping tools

The tools considered in this paper are: Timestomp, SetMACE, and nTimestomp. The first two are very prominent and have been used by various researchers/practitioners; nTimestomp was chosen as it was only recently released on January 10, 2019 (Lim, 2019).

Timestomp: As mentioned, Windows stores timestamps in 64-bit values which allow for nanosecond precision. Gungor (2014) discovered that the tool Timestomp truncates the nanoseconds value for the timestamps it alters. The last 7 digits in the timestamp used to describe the nanoseconds are set to zero by the tool. Given that a series of 7 zeros is extremely unlikely to happen, this can be used as an indicator for timestamp manipulation. SetMACE and nTimestomp both write to nanosecond precision and thus are harder to detect.

SetMACE: Up until version 1.0.0.6, SetMACE used the Windows API to alter the SIA timestamps and the previously described method in Fig. 1 to update the FNA timestamps. The tool is capable of changing the targeted timestamp to a 64-bit user defined value. The newer versions stopped using the API and instead write directly to the hard drive which was patched by Windows (discussed in Sec. 2.1). Consequently, the latest version[3] will not work on the newest Windows.

[3] At the time of writing this paper, the latest version is 1.0.0.16.

3. Previous work

Timestamps and their value for investigations have been discussed in literature, e.g., by Schatz et al. (2006). However, timestamp manipulation and its detection are not as well explored and only few peer-research articles exist where most of the current work focuses its efforts on the Master File Table (`$MFT`) and the `$LogFile`.

3.1. Detecting manipulation with timestamp rules

A first approach was timestamp rules which may be used for manipulation detection and can assist the reconstruction of events. Chow et al. (2007) was one of the first to create a comprehensive list of rules that could be used by examiners to identify specific behavior in NTFS. For example, "when a large number of files with 'close' A times are found inside the hard drive, those files are likely to be scanned by some tool, e.g. anti-virus software." While their research was not specifically aimed at identifying timestamp manipulation, it set the foundation for future research. Bang et al. (2009) furthered this research by additionally analyzing the SIA-E times as well as utilizing FNA timestamps. The authors were able to use these timestamp rules to identify malicious user behaviors, i.e., hiding information by replacing a folder with another folder that had the same name. Bang et al. (2011) also researched the timestamp behavior of files and folders within various versions of operating systems as well as the adding more timestamp rules. For instance, a list of actions where the existing MACE times are maintained and a list of actions where they are changed to the time of the operation. While previous research had focused on using timestamp rules to identify specific user actions, Ding and Zou (2010) was the first to use these rules to identify timestamp manipulation using a set of conditions such as: SIA-M should be less than or equal to SIA-E, SIA-C should be less than or equal to FNA-C, or SIA-C should be less than or equal to SIA-A. They were able to prove timestamp manipulation in an example case by comparing the values in the FNA to the values in the SIA and identifying inconsistencies.

3.2. Detecting manipulation with the $LogFile

Due to the fact that timestomping tools and techniques are capable of altering all eight timestamps in the `$MFT` with nanosecond precision, none of the aforementioned rules can be utilized to identify timestamp manipulation (as long as an attacker followed the rules) (Jang et al., 2016). However, timestamps before and after any change are also stored in the `$LogFile` which can support an analysis. To the best of our knowledge, Cho (2013) was the first to utilize the `$LogFile` to detect timestamp manipulation. The author discussed the `$LogFile` as a supplemental technique if timestamp rules failed to detect forgery. In their example they concluded that since the `$LogFile` did not contain any records for the file's FNA, those timestamps were never updated. Consequently, the relationship between the FNA times and SIA times now did not satisfy basic timestamp rules and they were able to identify the manipulation.

While that research was a major advancement, an even newer and more comprehensive study was published by Jang et al. (2016). The authors used several different timestomping tools to make random time alterations to a large number of text files. Next, they applied their rules (similar to the ones we discussed previously) to identify timestamp manipulation but combined it with evidence

from the $LogFile. One limitation to their research was the fact that they used randomized timestamps for their alterations which often did not obey basic NTFS timestamp rules.

3.3. Other methods for detecting manipulation

Minnaard et al. (2014) discovered that the directory indices stored timestamps did not update when a timestamp was altered using direct disk access. They concluded that while there were discrepancies initially, there was an error correction method in Windows that later updated the value and fix the discrepancy. This paper also discussed different methods used for timestamp manipulation such as direct disk access and using the Windows API.

Freiling and Hösch (2018) conducted an experiment to show how altering digital evidence without leaving traces is a difficult task. While their research was not focused on the same artifacts we used in our research, some of their conclusions can be directly applied. For instance, they found that the removal of evidence can complicate the analysis of a system greatly. Rather than altering the artifacts, we found it easier to delete them or wipe their contents. Additionally, the notion that there are many different ways to identify if digital evidence was tampered with directly relates to our conclusion that the more methods examiners have for identifying timestamp manipulation, the greater the chance they will succeed.

Willassen (2008) created a tool that was able to detect timestamp manipulation by comparing $MFT record numbers. Since new $MFT records are placed in the first one hole on the table just looking at record numbers wouldn't provide an accurate chronological ordering for when files were added. However, by using generational markers the author's tool was able to detect timestamp manipulation. Unfortunately, we were not able to locate the author's tool and there were not enough details in the paper to accurately reproduce their results for our experiment. The four files we tested in this experiment were in the following order based on their $MFT record numbers T4, T1, T3, T2. Considering that T1 was our test document and T2, T3, and T4 were all timestamped to the same dates and times in that order, there is no logical correlation between their order and when or if they were timestamped. While we were not able to replicate their results this should still be considered an additional method that could potentially be used to identify timestamp manipulation.

Aside from analyzing the actual timestamps, there has also been research and suggestions that the tools themselves leave valuable artifacts behind as well. For instance, Geiger (2005) researched six different anti-forensics tools such as CCLeaner. They concluded that all of them failed to completely erase or hide evidence which may be due to how complex operating systems have become. Hence, it may be possible to find a program's presence in unallocated space or other artifacts. Cowen (2013) suggested that evidence relating to the tools being used could be found in PF files, LNKs, shell bags, jump lists, and Windows registry most recently used. Furthermore, they suggested comparing the creation times of LNKs to the creation times found in the $MFT but they failed to provide any reasoning, rules, or examples. Gungor (2014) found flaws with the timestomping tools themselves. For instance, Timestomp does not modify the entire 64-bit time value used in NTFS and thus the nanoseconds are all set to zero making the changes easy to identify. Singh and Singh (2018) identified 9 different artifacts in NTFS that can be used to identify program execution: PF files, LNKs, jump lists, userassist, amcache.hve, iconcache.db, appcompatflags, appcompatcache, runMRU, and muicacheandSRUDB.dat. They further went on to test anti-forensic tools that delete system data such as CCleaner and were able to find evidence that all of the tools were run on the system.

4. Apparatus and methodology

Before discussing the procedure of our experiments and how we were able to extract evidence, we list all software products including their versions that were utilized:

- Autopsy 4.7.0
- Oracle VM VirtualBox 5.2.16
- Windows 10 Pro version 1803
- UsnJrnl2Csv 1.0.0.22
- NTFS LogFile Parser 2.0.0.46
- Event Log Explorer 5.0.1.4018
- FTK Imager 4.2.0.13
- Tableau 2018.2.0
- SetMace 1.0.0.5
- nTimestomp (x64) v1.1
- Timestomp

All images were captured with FTK Imager and processed in Autopsy for further analysis (to extract specific files/artifacts). Certain files needed to be extracted and parsed with other open source tools such as UsnJrnl2Csv for the $USNjrnl or NTFS LogFile Parser for the $LogFile. The resulting CSVs were then loaded into Tableau which helped visualize the information.

We realize that there are a variety of tools that could be used to parse specific artifacts from NTFS but we chose these ones based on availability, ease of use, and their good reputation. Another tool worth mentioning that could be used to parse much of the same data is Plaso's Log2Timeline (Metz, 2019). While we did not use this tool in our experiment we believe it would be able to carve much of the same information from a forensic image.

Test image. We started by creating a virtual machine in VirtualBox having a 40 GB fixed virtual hard drive. The operating system used for our tests was Windows 10 (default installation). There were no additional partitions set up for this test, only the system volume. Four test files were created on the system named: f_default.txt, f_Timestomp.txt, f_SetMACE.txt, and f_nTimestomp.txt.

Procedure. Our experiment was done using the following five basic steps; results are presented in the Experimental results section:

I. **Preparation:** The aforementioned test image was booted, and all three timestamp manipulation tools were installed. Additionally, we enabled the $USNjrnl running the

Table 3
Alterations made to the test files for the experiment.

File name	Original SIA-C	Altered SIA-MACE	Time tool was run	FNA altered?
f_default.txt	03/01/2019 at 14:15 EST	N/A	N/A	N/A
f_Timestomp.txt	03/01/2019 at 14:16 EST	03/01/2019 at 18:31:58 EST	03/02/2019 at 14:58:23	No
f_SetMACE.txt	03/01/2019 at 14:17 EST	03/01/2019 at 18:31:58:1234567 EST	03/02/2019 at 14:59:05	No
f_nTimestomp.txt	03/01/2019 at 14:18 EST	03/01/2019 at 18:31:58:1234567 EST	03/02/2019 at 15:00:13	Yes

following command: `fsutil USN createjournal m = 1000 a = 100 c:.`

II. **Timestamp manipulation:** Each tool was executed to perform a timestamp manipulation on its corresponding test file. An overview is provided in Table 3 where we list the test files with their original SIA timestamps along with their altered ones.
III. **Evidence identification and extraction:** After the files had been manipulated, we imaged the virtual machine for further analysis. We extracted the `$LogFile`, the `$USNjrnl`, the Windows Event Logs, and any Prefetch files for the timestomping tools for further analysis (the Link files were examined in Autopsy). The extracted artifacts were further analyzed using open source tools, which we will discuss in Sec. 5.
IV. **Evidence reliability testing:** After further analysis the test machine was booted up again and we attempted to delete all of the evidence of timestamp manipulation on the device. The objective was to test each of the artifacts we analyzed in the previous step and test their reliability, i.e., can a sophisticated attacker/malware tamper with or delete them.
V. **Verification:** To test these results and because deleting evidence could produce new evidence, we repeated step 3 and analyzed the system again.

Various snapshots were created throughout the experiment which allowed us to analyze the impact of each step individually.

5. Experimental results

An essential aspect of our work was to find artifacts that have not previously been used to identify timestamp manipulation in NTFS. In the following we provide a summary of artifacts we used in our experiment along with the results from our tests. Overall, we probed the following five artifacts:

1. `$LogFile`
2. Prefetch Files
3. `$USNjrnl`
4. LNKs (Link files)
5. Windows event logs

While the above artifacts may be well-known within the forensic community, to the best of our knowledge, there are no proposed methods for using them to identify potential timestamp forgery with the exception of the `$LogFile`. The upcoming sections discuss each of these artifacts where each section will first explain the artifact itself followed by our experimental findings. The last paragraph of each section outlines the artifacts reliability and our methods for deleting or otherwise obfuscating the evidence it contained.

Note, as will be shown in the corresponding *data reliability* paragraphs, an active adversary may be able to falsify information. However, we argue that all digital evidence can be manipulated and thus our results are still helpful for practitioners.

5.1. $LogFile

The `$LogFile` has previously been used for detecting timestamp manipulation which is summarized in Sec. 3.

The `$LogFile` is a Windows protected file and is the 3rd entry in the `$MFT` (Schwarz, 2007). It keeps track of changes that are made to files on the volume and was created to help the system recover from an unexpected crash. The records are stored sequentially where each entry is given a unique Log Sequence Number (LSN) that is also stored in the related file's `$MFT` record. Before any meta-data changes for a file are made, a record is created in the `$LogFile` that details the upcoming modification and stores a copy of the original data. Thus, if the system crashes, it can be reverted to a valid state. Otherwise, the `$LogFile` record gets flagged as the change was successful (Carrier, 2005). In order for this journaling method to be successful, it needs to record every change that is made to a file which in turn creates a large number of records. It is important to note that the `$LogFile` is circular, meaning the oldest entries are overwritten by newer ones when the file reaches its capacity (Polakovic, 2016). Its size varies based on the size of the system volume but is typically 64 MB or less. That means, it holds approximately between two to 3 h' worth of information with normal computer usage (Oh, 2013).

Table 4
Prefetch files for timestomping tools and their run times.

File	Time of program run
Timestomp.exe-C3A5003F.pf	03/02/2019 at 14:58:23
SetMace.exe-9AD6A728.pf	03/02/2019 at 14:59:06
nTimestomp.exe-295C9CCD.pf	03/02/2019 at 15:00:14
nTimestomp.exe-295C9CCD.pf	03/02/2019 at 15:00:41
nTimestomp.exe-295C9CCD.pf	03/02/2019 at 15:01:08

Findings. We extracted and analyzed the `$LogFile` from the snapshots using autopsy and further parsed the file with the LogFileParser.[4] Each `$LogFile` extracted was exactly 56,639,488 bytes (56 MB) which coincides with the earlier statement that the file is $\leq$ 64 MB. The `$LogFile` (after running the programs) contained records for each of the timestomping tools PF files, i.e., a timestamp of when each tool was executed. We were also able to recover a record for `f_nTimestomp.txt` that contained the file's timestamps from before and after they were altered. However, there were no records for the `f_Timestomp.txt` or the `f_SetMACE.txt` in the `$LogFile` in our test, likely due to the small file size.

Data reliability. The `$LogFile` is protected, always enabled, and records every meta-data change on the system. While we did not find any way to alter it directly, we were able to exploit the small capacity and its circular storage behavior. For testing, we ran a python script to flood the `$LogFile` with irrelevant information. The script makes a temporary directory in a user-specified location and then creates, modifies, reads, and deletes a file. In our experiment 1000 iterations were sufficient to produce an overflow (in fact the first 202 were overwritten themselves). While this requires using an additional script, it is an effective way to clear the contents of the `$LogFile`. Alternatively, one can wait until the file is naturally overwritten (see Sec. 5).

5.2. Prefetch files

PF files are created by the system whenever a program is executed for the first time and are used to speed up future executions; every time a program is re-run, Windows attempts to locate the associated prefetch file. The naming convention for PF files is simple as it starts with the name of the associated program (including its extension) which is then followed by a hyphen and 8 characters representing a hash of the file path from where the application was executed (McQuaid, 2014b). Most relevant, these PF files store timestamps for the eight most recent executions and remain on the system even if the software is deleted.

Findings. To analyze the PF files on the system, we mounted a

[4] https://github.com/jschicht/LogFileParser (last accessed 2019-04-25).

copy of our image and pointed a tool called WinPrefetchView[5] at the test systems prefetch folder, usually `C:/Windows/Prefetch`. As expected, all tools executed during our experiment created PF files. Naturally, these PF files have their own `$MFT` record and thus their own set of timestamps which can indicate the first time the program was run. Additionally, PF files contained timestamps for each execution as shown in Table 4. Besides timestamps, we also know that each program remained in the same location/directory due to the identical hashes. It should be noted that the file path being hashed also includes the name of the program being run, thus different programs (or renamed) run from the same directory will not have matching hashes.

As discussed in Sec. 2.1, modifying the timestamp values in the FNA requires the user to trigger a natural update from the file system. To achieve this, the timestomping tool has to be run a minimum of three times to overwrite the SIA values while moving the target file around. Hence, PF files can be used to detect this action by analyzing the timestamps for the previous eight times the program was run. For instance, Table 4 shows that nTimestomp was run three times in less than 60 s. This can serve as an indication that the FNA values may have been manipulated as well. Of course, if a timestomping tool automates this process (and does not need to be run three times) then this method is ineffective. In general, PF files are a good source of evidence when trying to identify malicious programs running on the system and should be utilized by examiners.

Data reliability. PF files are regular Windows files and therefore deleting them can be done by navigating to the prefetch folder. Due to the fact that these files can be deleted individually, and the other PF files are left unaltered, it is difficult for an examiner to identify when information is missing in this folder. However, it should be noted that when we processed the final image in Autopsy, we were able to recover some of the data for the SetMACE PF file. The PF file's SIA and FNA timestamps were carved from unallocated space but the file content was not. This means that we were not able to analyze the execution times that are typically stored in the file. While none of the other PF files were recovered in this study, it is certainly possible to carve entire files from unallocated space in the memory.

5.3. $USN journal

The `$USNjrnl`, sometimes referred to as the change journal, is log file in NTFS that keeps track of any when changes are made to files. Change journals can be created or deleted using the followings commands:

- `fsutil USN createjournal m = 1000 a = 100 c:` creates a new journal and enables the file system to keep track of changes, where `m` is the max size, `a` is the allocation delta, and `c:` is the drive the `$USNjrnl` will be used on (Microsoft, 2017).
- `fsutil USN deletejournal/d c:` deletes the journal which essentially dumps its contents into unallocated space.

The `$USNjrnl` is also a Windows protected file which makes it more difficult to alter or to delete. The file is located in the `$Extend` folder which is the 11th entry in the `$MFT` (Carrier, 2005, p. 202). Within the `$Extend` folder, there is a file named `$USNJrnl:$Max` which contains basic information about the journal itself and a file named `$USNJrnl:$J` which has the actual journal entries. For this paper, we will focus on the latter of the two and use the term `$USNjrnl` as a synonym.

Table 5
`BASIC_INFO_CHANGE` record creation times compared to SIA-E times for our test files.

File	Time of meta-data change
f_Timestomp.txt (SIA-E)	03/01/2019 at 18:31:58 EST
f_Timestomp.txt (USNJ Record)	03/02/2019 at 14:58:23 EST
f_SetMACE.txt (SIA-E)	03/01/2019 at 18:31:58 EST
f_SetMACE.txt (USNJ Record)	03/02/2019 at 14:59:06 EST
f_nTimestomp.txt (SIA-E)	03/01/2019 at 18:31:58 EST
f_nTimestomp.txt (USNJ Record)	03/02/2019 at 15:00:14 EST

Table 6
`BASIC_INFO_CHANGE` records from the `$USNjrnl`.

File	Time of Change
f_Timestomp.txt	03/02/2019 at 14:58:23 EST
Timestomp.exe-C3A5003F.pf	03/02/2019 at 14:58:23 EST
f_SetMACE.txt	03/02/2019 at 14:59:06 EST
SetMace.exe-9AD6A728.pf	03/02/2019 at 14:59:06 EST
f_nTimestomp.txt	03/02/2019 at 15:00:14 EST
nTimestomp.exe-295C9CCD.pf	03/02/2019 at 15:00:14 EST

When enabled, it can be extracted using Autopsy before being parsed with UsnJrnl2Csv.[6] The journal maintains several events but most importantly it has a record for every file change on the volume (event category: `BASIC_INFO_CHANGE`). While the journal functions similar to the `$LogFile`, it should be noted that it does not record the original data or what changes were made. Instead, it records the time a change occurred, the associated file name, and the category of the change that occurred. For this research, we primarily focused on records that fell under the `BASIC_INFO_CHANGE` category which included any changes related to file meta-data and timestamps (Microsoft, 2018d). Additionally, each record in the `$USNjrnl` contains a unique identifier called an Update Sequence Number (USN) that can also be found in any file's SIA.

Findings. Upon parsing the `$USNjrnl`, we found `BASIC_INFO_CHANGE` records for each of the three files we had modified. These records provided us with the last time the test file had its meta-data altered. Consequently, if the `$USNjrnl` recorded a change to the files meta-data, then the file's SIA-E should also show these changes. As seen in Table 5 (where we compare the file's SIA-E times to the creation times for the `$USNjrnl` `BASIC_INFO_CHANGE` records), there are discrepancies. All of the test files in our experiment had `BASIC_INFO_CHANGE` records created on 03/02/2019 but their SIA-E times were last updated on 03/01/2019. This is a strong indicator that the timestamps for these files were altered.

Additionally, the `$USNjrnl` stored records on the PF files, i.e., whenever a program was executed (and the PF file changed) a `BASIC_INFO_CHANGE` record was found. Consequently, we looked for programs or files that had `BASIC_INFO_CHANGE` records at or around the same time as the test files. These comparisons can be seen in Table 6 and show that each of the test files had a meta-data change at the same time a PF file was updated. This correlation can help examiners to potentially identify what program(s) were run to modify the timestamps. Additionally, if the `$USNjrnl` is not available this same correlation can be made using the eight most recent execution times stored within a PF file.

The `$USNjrnl` also creates a `USN_REASON_FILE_CREATE`[7]

[5] https://www.nirsoft.net/utils/win_prefetch_view.html (last accessed 2019-04-25).

[6] https://github.com/jschicht/UsnJrnl2Csv (last accessed 2019-04-25).

[7] Another category instead of the BASIC_INFO_CHANGE.

Table 7
Test file's SIA timestamps compared to their LNKs SIA timestamps (extracted directly from Autopsy).

File	SIA-M	SIA-E	SIA-A	SIA-C
f_Deafult.lnk	2019-03-01 14:15:17 EST	2019-03-01 14:15:17 EST	2019-03-01 14:15:17 EST	2019-03-01 14:15:17 EST
f_Deafult.txt	2019-03-01 14:15:27 EST	2019-03-01 14:15:27 EST	2019-03-01 14:15:30 EST	2019-03-01 14:14:59 EST
f_Timestomp.lnk	2019-03-01 14:16:14 EST	2019-03-01 14:16:14 EST	2019-03-01 14:16:14 EST	2019-03-01 14:16:14 EST
f_Timestomp.txt	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST
f_SetMACE.lnk	2019-03-01 14:17:29 EST	2019-03-01 14:17:29 EST	2019-03-01 14:17:29 EST	2019-03-01 14:17:29 EST
f_SetMACE.txt	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST
f_nTimestomp.lnk	2019-03-01 14:18:10 EST	2019-03-01 14:18:10 EST	2019-03-01 14:18:10 EST	2019-03-01 14:17:17 EST
f_nTimestomp.txt	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST	2019-03-01 18:31:58 EST

record whenever a file is added to the system (Microsoft, 2018d). This record can be used to identify when the timestomping tool first arrived on the system which could be a crucial piece of evidence in an investigation. Furthermore, we also recovered a file creation record for the PF files associated with the timestomping tools that allowed us to identify when the tools were first run on the system.

Data reliability. An attacker could move the journal into unallocated space running the `fsutil usn deletejournal/d c:` command and create a new journal in the same location running the `fsutil USN createjournal m=1000 a=100 c:` command. While wiping `$USNjrnl` is straightforward, these commands do trigger Windows to record an event in its application event log: Windows event ID 3079 is created when the `$USNjrnl` is deleted. This can be a good indication for an examiner that further data manipulation may have occurred on the system. Based on our findings we conclude that since it is easy to delete it should not be considered a reliable source of information for examiners.

5.4. Link files

LNKs are essentially shortcuts to local files that are created manually by the user or automatically by the file system. The more common of the two options is the latter case: whenever a local file is created or opened for the first time, a LNK is created in `C:/Users/NAME/AppData/Roaming/Microsoft/Windows/Recent` (McQuaid, 2014a). All of the LNKs used in our analysis were recovered from this location and automatically created. Additionally, LNKs contain the file path for their associated file as well as additional information about the file's storage location including the volume name and potentially the MAC address of the remote device the file is located on. Most importantly, each LNK has its own `$MFT` entry that can be analyzed.

Findings. During our experiment we tested and confirmed that anytime a file is opened on the local machine (either by clicking directly on it or locating it in the file explorer), the associated LNK's SIA-A and SIA-E times are updated as well. Thus, the SIA-A and SIA-E timestamps for the LNK should approximately match to those timestamps for their associated file. This rule excludes the file's SIA-M and SIA-C timestamps because they can vary between a file and its corresponding LNK. An example is provided in Table 7 which shows several discrepancies when comparing the two file's SIA timestamps. The default file that we created during the experiment portrays the expected relationship: the difference between both timestamps is in the range of seconds. On the other hand, there are significant differences for the manipulated files as they were never accessed by the user directly.

While this can be an indicator for timestamp manipulation, it is important to note that the SIA-A time could be updated by other programs that would not update the LNK's timestamps such as an anti-virus scanner (Chow et al., 2007). For instance, running a scan with Windows Defender confirmed that the SIA-A timestamps for all of the test files were updated (the other three timestamps were not affected). If there was an anti-virus scan that updated the SIA-A time, then the SIA-E time could still be used.

Side note: A LNK cannot be created for a file that does not exist, i.e., the file's SIA-C time is not allowed to be in the future. The LNK's SIA-C time should be equal to or after the SIA-C time listed for its associated file. This discrepancy can also be seen in Table 7 where the SIA-C times for all three of the manipulated test files are 4 h after their associated LNK's SIA-C times. Similarly, if a LNK's SIA-A time is updated, the corresponding file's SIA-A time should be updated as well. A LNK's SIA-A time should be equal to or before the actual file's SIA-A time.

Data reliability. Similar to the PF files, LNKs are regular Windows files that can be deleted easily. Singh and Singh (2016) extensively researched LNKs and found that while they could be deleted, it was possible to recover some of or all of their information from unallocated space. They also found that if a LNK was modified in any way, the changes would be reverted back as soon as the file was accessed again. In our experiment, we were able to carve a LNK for each of our test documents. However, other than the names of the files, we were not able to recover any information. It should also be noted that a new LNK will not be created until the file has been accessed again.

Table 8
Active user times according to windows system events.

Date	Login time	Logoff time
03/01/2019	13:19:09 EST	13:28:09 EST
03/01/2019	13:28:11 EST	14:52:03 EST
03/02/2019	14:37:03 EST	15:02:17 EST
03/02/2019	16:58:13 EST	17:13:22 EST

5.5. Windows event logs

Windows keeps logs of events that happen on the system which are organized into several categories and separate log files: the Application log, the System log, and the Security log (Microsoft, 2018b). While these logs have a diverse collection of data, we focused on the System log which maintains events that allow us to track user activity. Specifically, an event with the ID number 7001 is created when a user logs in; logging off creates a 7002 event. Hence, it was possible to create a timeline of when the user was active which can be cross-referenced with other timestamps on the system (timestamps outside of active user sessions are suspicious).

Findings. Autopsy was able to extract the artifact and we opened it locally with a program called Event Log Explorer.[8] An example of user activity for our experiment is depicted in Table 8. If either the SIA-C or the SIA-M times are outside an active session, it is an

[8] https://eventlogxp.com/download.html(last accessed 2019-04-25).

indication that timestamp manipulation has occurred. The SIA-A and SIA-E times were not included due to the fact that they may have been updated by anti-virus software or some other program running as a background service. For our experiment files were created outside an active user session (compare Table 7 vs. Table 8).

Data reliability. Both the Application log and System log can be cleared by right-clicking them in the Windows event viewer and selecting the clear log option. However, this action triggers a new event (104) in the System log which includes the time it was cleared and the user who cleared it. While Windows event logs were the most persistent source of information, they are unable to retain any evidence that could be used to identify timestamp manipulation.

6. Discussion

The findings from previous work and our experiment showed that timestamps can be a valuable source of evidence, but a sophisticated adversary could manipulate them. In the following, we respond to the three research questions proposed in the introduction.

[RQ1] What is the range of artifacts that can be used by examiners to identify timestamp manipulation in NTFS on Windows? While existing literature mostly focused on the `$MFT` (SIA/FNA timestamps) and `$LogFile` to detect timestamp forgery, we found four new artifacts: `$USNjrnl`, PF files, LNKs, and Windows event logs. These artifacts increase the diversity of artifacts available to examiners and therefore complement prior work. Each of these artifacts has its own unique value and data that relates to timestamp manipulation. Compared to previous work, the new findings give examiners more 'long-term' artifacts such as PF and LNK files as compared to the `$LogFile` and `$USNjrnl` whose data can be naturally overwritten rather quickly.

In general, we realized that there is a lot of (meta-)data available, and that we may have missed other artifacts during our experiments. For instance, Schicht (2014), in their description of the SetMACE tool, suggested that evidence of time forgery could also potentially be recovered from shadow volumes. The importance of the shadow volume data was addressed by Leschke and Nicholas (2013) who created a tool for visualizing its changes, i.e., a file's timestamps being changed. This concept may be applied on an even larger scale by running comparisons against entire backups attempting to identify suspicious behavior. However, further research is needed to conclude the feasibility for detecting timestamp forgery.

[RQ2] How reliable are the artifacts (i.e., resilient to obfuscation or deletion) that are being used for timestamp manipulation detection? Our experiments showed that none of the five artifacts we tested in this paper were a consistently reliable source for identifying timestamp forgery as they were not enabled by default (`$USNjrnl`) or they could be deleted/exploited. A positive aspect was that tampering with artifacts occasionally left behind other (more permanent) artifacts. For instance, clearing the Windows event log resulted in a new Windows event. However, these artifacts are merely suspicious and do not prove timestamp forgery.

As pointed out by Yoo et al. (2010), there is the possibility to carve deleted data from unallocated space. During our experiments we were only able to carve PF files and LNKs using Autopsy, but there may be tools that have the capability to recover deleted records from the `$LogFile` and the `$USNjrnl`. Future work could look into specialized techniques that are able to recover these artifacts.

On the other hand, Microsoft has imposed restrictions in NFTS which made it harder to tamper with some artifacts (e.g., no direct disk writes to the system volume, no updating FNA timestamps, and Windows protected files cannot be altered at all). One potential solution could be identifying the misuse of time setting commands in the Windows API. While these commands are used in a lot of benign programs, they should not allow user input but instead rely on the internal clock in the computer.

For future research, we recommend that in addition to finding novel artifacts, one should also discuss their reliability. On the other hand, the operating system should make it harder to tamper with meta-data by putting more restrictions on their API or develop methods for detecting changes, e.g., a 'local Blockchain' that captures information and does not allow changes (similar to Sutton and Samavi (2017)).

[RQ3] Besides identifying timestamp manipulation directly, can we detect the execution/presence of timestamp forgery tools? During our experiments, we were able to detect the presence and execution of timestomping tools utilizing the PF files. Additionally, an examiner could perform string searches on the medium at hand to find evidence. Note, if an adversary masqueraded the name of the tool before execution (e.g., `Windows/System32/svchost.exe`), this procedure will not be successful (MITRE, a).

Similar to how malware can be detected by unique signatures or its behavior, one could create signatures for popular timestomping tools (Idika and Mathur, 2007; Conlan et al., 2016). For instance, in our scenario Timestomp was flagged by Windows Defender and Magnet AXIOM version 1.2.2.7502 as an anti-forensic tool (SetMACE and nTimestomp were not flagged). If the security tool's logging is enabled, Timestomp may show up in another artifact (application specific logs). Lastly, these signatures could be used to proactively stop the timestomping tools from altering timestamps.

7. Conclusion

Timestamp forgery has become popular and many hacking groups and malicious programs utilize these techniques to hide evidence and hinder investigations. Thus, it is essential for researchers and practitioners to have numerous artifacts and methods for detecting timestamp forgery. The more artifacts examiners know for identifying timestamp manipulation, the harder it becomes for a malicious user to obfuscate all of the traces.

In this article, we proposed four new methods that have not previously been used in peer-reviewed literature to detect timestamp manipulation: `$USNjrnl`, PF files, LNKs, and Windows event logs. These artifacts can contain evidence of timestamp forgery by providing additional, unaltered, timestamps as well as evidence of suspicious software activity. Based on our findings, we propose the following five rules that can be used to detect timestamp inconsistencies in NTFS on Windows:

1. The SIA-E for a file should be similar to the most recent `BASIC_INFO_CHANGE` record for the file in the `$USNjrnl`.
2. The SIA-E for a LNK file should be identical with the SIA-E of the associated file.
3. The SIA-C for a LNK file should be equal to or newer than the SIA-C of its associated file.
4. The SIA-A for a LNK file should be equal to or before the SIA-A of its associated file.
5. The SIA-C and SIA-M times for a file cannot be at a time when there is not a user logged in.

On the other hand, this article also explored the reliability of different artifacts used for detecting timestamp forgery. We concluded that none of the tested artifacts were a reliable source of information as they could be exploited by a sophisticated attacker or malicious software. Furthermore, to the best of our knowledge, there are no current methods that can consistently prove

timestamp manipulation as the evidence can always be deleted or altered to avoid detection. Consequently, we recommended using a combination of the previously mentioned methods such as timestamp rules and `$LogFile` analysis along with other system artifacts, such as LNKs and PF files, to increase the odds of detecting timestomping tools and techniques. Furthermore, the evidence stored in the `$LogFile` and `$USNjrnl` becomes obsolete after a certain amount of time which creates another set of problems for investigators as analysis on the targeted machine may not be done for an extended period of time. While this research was a solid first step at a more broad and holistic view of timestamp manipulation detection, this domain needs more research.

References

Alperovitch, D., 2016. Bears in the midst: intrusion into the democratic national committee. CrowdStrike Blog 15.

Bang, J., Yoo, B., Kim, J., Lee, S., 2009. Analysis of time information for digital investigation. In: 2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE, pp. 1858–1864.

Bang, J., Yoo, B., Lee, S., 2011. Analysis of changes in file time attributes with file manipulation. Digit. Invest. 7, 135–144.

Buchholz, F., Spafford, E., 2004. On the role of file system metadata in digital forensics. Digit. Invest. 1, 298–309.

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional.

Carvey, H., 2014. Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 8. Elsevier.

Cho, G.-S., 2013. A computer forensic method for detecting timestamp forgery in NTFS. Comput. Secur. 34, 36–46.

Chow, K.-P., Law, F.Y., Kwan, M.Y., Lai, P.K., 2007. The rules of time on NTFS file system. In: Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07). IEEE, pp. 71–85.

Conlan, K., Baggili, I., Breitinger, F., 2016. Anti-forensics: furthering digital forensic science through a new extended, granular taxonomy. Digit. Invest. 18, 66–75.

Cowen, D., 2013. Daily blog #130: detecting fraud sunday funday 10/27/13 part 3 – setmace. http://www.learndfir.com/2013/10/31/daily-blog-130-detecting-fraud-sunday-funday-102713-part-3-setmace/.

Ding, X., Zou, H., 2010. Reliable Time Based Forensics in NTFS. School of Software, Shanghai Jiao Tong University, pp. 1–2.

Freiling, F., Hösch, L., 2018. Controlled experiments in digital evidence tampering. Digit. Invest. 24, S83–S92.

Geiger, M., 2005. Evaluating Commercial Counter-forensic Tools. DFRWS.

Gungor, A., 2014. Date forgery analysis and timestamp resolution. https://www.meridiandiscovery.com/articles/date-forgery-analysis-timestamp-resolution/.

Hannon, M.J., 2018. Metadata in Civil and Criminal Discovery—Part ii, vol 35. The Computer & Internet Lawyer.

Idika, N., Mathur, A.P., 2007. A Survey of Malware Detection Techniques. Purdue University, p. 48.

Jang, D.-I., Hwang, G.-J.A.H., Kim, K., 2016. Understanding anti-forensic techniques with timestamp manipulation. In: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI). IEEE, pp. 609–614.

Koen, R., Olivier, M.S., 2008. The Use of File Timestamps in Digital Forensics. ISSA, Citeseer, pp. 1–16.

Leschke, T.R., Nicholas, C., 2013. Change-link 2.0: a digital forensic tool for visualizing changes to shadow volume data. In: Proceedings of the Tenth Workshop on Visualization for Cyber Security. ACM, pp. 17–24.

Lim, B., 2019. ntimetools. https://github.com/limbenjamin/nTimetools.

McQuaid, J., 2014a. Forensic analysis of lnk files. https://www.magnetforensics.com/blog/forensic-analysis-of-lnk-files/.

McQuaid, J., 2014b. Forensic analysis of prefetch files in windows. https://www.magnetforensics.com/blog/forensic-analysis-of-prefetch-files-in-windows/.

Metz, J., 2019. Plaso log2timeline. https://github.com/log2timeline/plaso.

Microsoft, 2017. Fsutil usn. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/fsutil-usn.

Microsoft, 2018a. Changes to the file system and to the storage stack to restrict direct disk access and direct volume access in windows vista and in windows server 2008. https://support.microsoft.com/en-us/help/942448/changes-to-the-file-system-and-to-the-storage-stack-to-restrict-direct.

Microsoft, 2018b. Event logging. https://docs.microsoft.com/en-us/windows/desktop/eventlog/event-logging.

Microsoft, 2018c. NtSetInformationFile function. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/ntifs/nf-ntifs-ntsetinformationfile.

Microsoft, 2018d. Read_usn_journal_data_v0 structure. https://docs.microsoft.com/en-us/windows/desktop/api/WinIoCtl/ns-winioctl-read_usn_journal_data_v0.

Minnaard, W., de Laat, C., van Loosen MSc, M., 2014. Timestomping ntfs. https://delaat.net/rp/2013-2014/p48/report.pdf.

MITRE (a). Masquerading. https://attack.mitre.org/techniques/T1036/.

MITRE (b). Timestomp. https://attack.mitre.org/techniques/T1099/.

Novetta, 2016. Loaders Installers and Uninstallers Report. Operation Blockbuster.

Oh, J., 2013. NTFS log tracker. http://forensicinsight.org/wp-content/uploads/2013/06/F-INSIGHT-NTFS-Log-TrackerEnglish.pdf.

Polakovic, P., 2016. NTFS LogFile parser. https://www.codeproject.com/Tips/1072219/NTFS-LogFile-Parser.

Schatz, B., Mohay, G., Clark, A., 2006. A correlation method for establishing provenance of timestamps in digital evidence. Digit. Invest. 3, 98–107.

Schicht, J., 2014. SetMACE. https://github.com/jschicht/SetMace.

Schicht, J., 2018. mft2csv - setmace.wiki. https://code.google.com/archive/p/mft2csv/wikis/SetMACE.wiki.

Schwarz, T., 2007. Ntfs Architecture for x86-Based Systems.

Singh, B., Singh, U., 2016. A forensic insight into windows 10 jump lists. Digit. Invest. 17, 1–13.

Singh, B., Singh, U., 2018. Program execution analysis in windows: a study of data sources, their format and comparison of forensic capability. Comput. Secur. 74, 94–114.

Sky, C., 2017. Operation wilted tulip: exposing a cyber espionage apparatus. https://www.clearskysec.com/wp-content/uploads/2017/07/Operation_Wilted_Tulip.pdf.

Sutton, A., Samavi, R., 2017. Blockchain enabled privacy audit logs. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (Eds.), The Semantic Web – ISWC 2017. Springer International Publishing, Cham, pp. 645–660.

Willassen, S.Y., 2008. Finding evidence of antedating in digital investigations. In: 2008 Third International Conference on Availability, Reliability and Security, pp. 26–32.

Windows, 2018. File times. https://docs.microsoft.com/en-us/windows/desktop/sysinfo/file-times.

Works, S., 2015. Threat group-3390 targets organizations for cyberespionage. https://www.secureworks.com/research/threat-group-3390-targets-organizations-for-cyberespionage.

Yoo, B., Park, J., Bang, J., Lee, S., 2010. A study on a carving method for deleted ntfs compressed files. In: 2010 3rd International Conference on Human-Centric Computing. IEEE, pp. 1–6.