Reconstructing Timelines: From NTFS Timestamps to File Histories

Jelle Bouma
Open University of the Netherlands
Heerlen, The Netherlands

Hugo Jonker
hugo.jonker@ou.nl
Open University of the Netherlands
Heerlen, The Netherlands

Vincent van der Meer
vincent.vandermeer@zuyd.nl
Zuyd University of Applied Sciences
Heerlen, The Netherlands

Eddy van den Aker
eddy.vandenaker@zuyd.nl
Zuyd University of Applied Sciences
Heerlen, The Netherlands

ABSTRACT

File history facilitates the creation of a timeline of attributed events, which is crucial in digital forensics. Timestamps play an important role for determining what happened to a file. Previous studies into leveraging timestamps to determine file history focused on identification of the last operation applied to a file. In contrast, in this paper, we determine all possible file histories given a file's current NTFS timestamps. That is, we infer all possible sequences of file system operations which culminate in the file's current NTFS timestamps. This results in a tree of timelines, with root node the current file state. Our method accounts for various forms of timestamp forgery. We provide an implementation of this method that depicts possible histories graphically.

CCS CONCEPTS

• **Applied computing → Evidence collection, storage and analysis**; **Investigation techniques**.

KEYWORDS

Digital forensics, Timestamps, File history, Timelines

ACM Reference Format:
Jelle Bouma, Hugo Jonker, Vincent van der Meer, and Eddy van den Aker. 2023. Reconstructing Timelines: From NTFS Timestamps to File Histories. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–1 September, 2023, Vienna, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A major goal of digital forensics is to construct a timeline of events. Specifically, placing digital evidence (incriminating as well as exculpatory) correctly on possible timelines is a key goal of digital forensics. For example, suppose a hacked computer is seeding a torrent file of illegally downloaded content. It is rather relevant whether the seeding started before or after the computer was hacked. In some cases, this can be determined via the timestamps of relevant files. File timestamps log, amongst others, the effect of file operations: when a file last was written to, was last read, or was last accessed.

ARES 2023, August 29–September 1, 2023, Benevento, Italy
2023. ACM ISBN 978-1-4503-9051-4/21/08…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Operations on a file have a deterministic effect upon the file's timestamps. That is: executing an operation on a file will cause the file's timestamps to be updated in a specific, predetermined way, based on the time the operation was executed and the initial timestamps of the file. Moreover, different operations can affect timestamps differently. Therefore, assuming (see Sec. 8) no tampering and a monotonically increasing clock, a given set of timestamps can only result from a limited set of file operations. In addition, each of these possible operations imposes certain requirements on the values of the timestamps prior to its execution. As such, it is possible to determine the set of possible operations that were last applied to a file, i.e., the operations that could have led to that set of timestamps. By applying this recursively, we can reconstruct all histories of a file possible for a given set of operations. Moreover, some timestamp forgery approaches have detectable, distinctive effects and can, thus, be included in this history. We consider our method sufficiently mature to be used by practitioners.

Contributions. The main contribution of this research is a method to determine all possible histories of a file, given a set of operations. This method is based on (1) determining, for each operation, its effect on timestamps, (2) inverting these effects and determining under which constraints this inverse may be applied, and (3) reasoning back from the file's current timestamps by matching these inverse effects. Secondly, we develop a proof-of-concept toolchain implementing this method for the NTFS file system. This allows us to visualise the histories (sequences of operations) possible given the file's current timestamps.

Availability. We provide two proof-of-concept tools that together implement the presented method. The tool TimestampAnalyzer implements the history-inference method; the tool TimestampVisualizer parses TimestampAnalyzer's output and presents it graphically. Both tools are publicly available on GitHub.[1] Our toolchain, which includes timestamp effects of an initial list of file operations, can support practitioners in executing the proposed method.

2 BACKGROUND

New Technology File System (NTFS). The NTFS file system has been the default on Windows systems since Windows 2000. It stores

[1] https://github.com/JelleBouma/TimestampAnalyser and https://github.com/eddyvdaker/NTFS-Timestamp-Visualizer.

information about files in a Master File Table (MFT). An MFT-entry contains metadata as well as disk allocation data for that file. Entries have multiple attributes for storing metadata; timestamps are stored as part of an entry's `$STANDARD_INFORMATION` (SI) and `$FILE_NAME` attributes (FN).

Timestamps in NTFS. A timestamp denotes when a certain event has taken place. The NTFS file system stores eight timestamps per file in the file's entry in the MFT. Operations on files cause changes to anywhere between zero and eight timestamps. We therefore treat these eight timestamps as separate data points. Four timestamps are stored in a file's SI attribute, and another four are stored in the file's FN attribute. Timestamps are stored in units of 100 nanoseconds (10^{-7} seconds) since 1601-01-01 00:00:00 UTC. As such, timestamp values are not affected by local time zone or daylight saving time.

3 RELATED WORK

The importance of timestamps in digital forensics has long been established in literature. For example, Buchholoz and Spafford [3] address importance of file system metadata (including timestamps), and discuss considerations and limitations that arise when trying to answer when and where a file came from, and who did that.

Effect of file operations on timestamps. Timestamps change when file operations are executed. This can be leveraged for forensic purposes. In 2007, Chow et al. [7] were one of the first studies to describe the effects of file operations on timestamps. They did this for both files and folders, using 3 SI timestamps from NTFS. The authors introduce timestamp rules to describe the effect of a file operation, warning to not apply these rules without considering timestamp-forgery. In their 2009 study, Bang et al. [1] expanded upon this work by adding the FN timestamps for NTFS and the FAT filesystem, and their expected values for a set of file operations and FAT timestamps. With these timestamp rules, the authors demonstrate how to identify manipulation of timestamps. In 2011, Bang et al. [2] expanded upon their previous study by analyzing file operations and their effects on timestamps for Windows from 2000 up to Windows 7. In addition, the authors observe different timestamp effects when modifying a file via Notepad vs. via MS Word. This suggests application-specific timestamp behaviour.

Timestamp manipulation detection. Timestamps can be manipulated. Various works have investigated detection of manipulated timestamps. Ding and Zou [8] proposed a cross-reference time-based approach to detect timestamp-manipulation. With the Windows Registry as their cross-reference source for a given set timestamps and derived timestamp rules, they demonstrate how certain timestamp manipulations can be detected. Cho [4] showed in 2013 how $LogFile can be used as a source for timestamp validation. Based upon the timestamp effects for seven file operations, he derives new rules that the timestamps must adhere to. When the timestamp rules are violated, the $LogFile can give conclusive evidence when the forgery occurred. In a subsequent study, Cho [5] described timestamp changing patterns based on file operations. He presented ten distinguishable patterns which can be conclusively attributed to specific file operations. To the best of our knowledge, this is the first work that aims to identify the last operation that was performed on a file. Jang et al. [10] presented a methodology to identify timestamp manipulation. Much like earlier work, they derive effects of operations on timestamps and build upon this. This resulted in equations of relations between $MFT and $LogFile, violation of which indicates timestamp manipulation. Palmbach and Breitinger [12] consider alternative sources of information for detecting timestamp manipulation. These include prefetch files, $USN journal (an NTFS log file), link files, and Windows event logs. None of the tested artifacts individually constitutes a reliable source of information, as each of them can be manipulated. However, the more sources of information included in a forensic analysis, the harder it is to manipulate all artifacts consistently.

Other impacts on timestamps. Other studies have considered timestamp effect beyond those of regular operations and timestamp manipulations. Schatz et al. [13] report on system clock behaviour, one of the factors that influence timestamp reliability. They characterize the behaviour of drifting clocks and describe ways to correlate timestamps to events by using other, more reliable source(s). Willassen [14] expanded upon their work, presenting a methodology for determining whether or not the system clock was altered using causality of timestamps. Galhuber and Luh [9] showed that, in addition to regular file operation and their effects on timestamp, many applications have specific timestamp effects not matching regular file operations. This gives rise to a potentially enormous set of operations with unique timestamp effects. While it may seem impossible to detect timestamp forgery in the face of this multitude of options, it turns out that most timestamp tools have telltale limitations (e.g., setting timestamps to full seconds), which allows their effects on timestamps to be distinguished from other operations. Lastly, Nordvid and Axelsson [11] examined whether different operating systems treated file systems equally. For the exFAT file system, they experimented with Windows, MacOS and Linux, and conclude that not all file system drivers implement the specification equivalently. Small differences exists in how each operating system stores exFAT timestamps.

4 METHODOLOGY: REASONING BACKWARDS

As previously stated, the goal is to arrive at the set of possible histories of a file. More specifically, each element in this set is an ordered list of operations that may have been applied to the file, with the last operation in the list culminating in the file's current state. Note that some sequences of operations are not possible. For example, creation must be the first operation; it cannot be preceded by other operations. To arrive at such a set of lists, we first determine how to denote a file's current state and make a selection of file operations to consider.

Next, for each file operation under consideration, we determine what effect it has on file state. Note that an operation can impose very specific effects on file state. This implies that the state of a file whose metadata lacks this telltale signature cannot be the direct result of that specific operation. For example, suppose an operation initialises all timestamps to the same value. A file whose timestamps are all equal may then be the direct result of this operation; however, a file whose timestamps vary, cannot.

Next, we determine for each operation its effect on file state. Then, we determine what the previous state of a file was, if the current state is the result of the operation under consideration. That

is, we determine the effect's inverse. As mentioned, some operations cannot possibly have resulted in a given state. Such operations must thus be excluded for this transition in a file's history.

Given the inverse state transition for each operation, and the requirements that state transition imposes on the resulting timestamp, we can then reason backwards to reconstruct the set of all possible lists of operations applied to the file, given a specific list of file operations and a specific state of a file.

5 EFFECT OF OPERATIONS ON TIMESTAMPS

In this section, we discuss the effect of operations on NTFS timestamps. To that end, we first establish notation of file state and a list of file operations to consider. File operations have different effects depending on "modifiers" – system settings under which (some) operations have a (slightly) different effect on timestamps. Last, we measure the impact of these modifiers separately.

Selecting file operations. There is no consensus in literature on a canonical list of file operations to include for timestamp research. We align with previous studies by including the basic CRUD operations (*create*, *access*, etc.), operations for altering file meta-data (*rename* and *attribute change*), and basic file system operations affecting two MFT entries (variants of *move* and *copy*). This leads to the following list:

- **Create:** Create new MFT entry
- **Access:** Access file contents
- **Update:** Modify file contents
- **Delete:** Mark MFT entry for deletion
- **Rename:** Change file name in MFT
- **Attribute change:** Change attribute(s) in MFT
- **Copy:** Create new MFT entry from source MFT entry, copy on-disk data
- **Overwriting copy:** Copy, destination is an existing MFT entry
- **Move within volume:** Updates the file path within the MFT entry
- **Move from another volume:** Create new MFT entry on destination volume, marks MFT entry for deletion on source volume
- **Overwriting move from other volume:** Destination is an existing MFT entry

5.1 Modifiers on file operations

File operation modifiers describe (system) environment changes that affect the workings of existing file operations. For example, 'last access updating' is a Windows system setting that can be active or not. Among others, the file operations 'copy' and 'update' are slightly different depending on this setting. In this research, we account for the following four file operation modifiers.

File tunneling. To prevent data loss during saving (e.g., in case of write errors), OS designers invented 'safe save'. 'Safe save' does not save changes to the original file, but to a new file instead. Once writing this file is finished successfully, the new file then replaces the original file by removing the original and renaming the new file. Normally, this file would appear to the user as a newly created file. To prevent that, Windows introduced file tunneling. Due to file tunneling, a create or rename operation is treated differently if the OS determines that file tunneling applies. More specifically, if a file is deleted or renamed, some of its metadata (incl. certain timestamps, long file name) is cached for a short time (default: 15 sec). If within this time frame a new file is created with the original name/path, or an existing file is renamed to the original name/path, that other file acquires the cached metadata. Note that, while file tunneling is intended to offer functionality for 'safe save', its implementation does not check for intent. That is, file tunneling occurs when its preconditions are met, irrespective of whether a 'safe save' operation was intended.

Last access updating. The last access timestamp value indicates the most recent event when the file was accessed. This feature is controlled by a registry setting, and was by default disabled for Windows Vista, 7, 8 and early builds of Windows 10. Since Windows 10 build 1607 (August 2016), last access updating is by default enabled for volumes smaller than 128 GB. The actual update of the last access timestamp itself is typically first cached in memory, not immediately written to disk. Writing the new timestamp to disk can be postponed by up to an hour.[2]

Transfer from FAT/exFAT. Occasionally, files are copied from a non-NTFS file system onto an NTFS file system. As the originating file system may differ in which timestamps are tracked, and to what resolution, the NTFS copy's timestamps may bear recognizable traces of the file's origin. In this paper, we investigate the effects of transferring files from FAT and exFAT file systems, two file systems widely used for portable storage media (memory cards, USB keys, portable hard disks).

The modified effects from file-operations originating from FAT or exFAT primarily come from limitations that these file systems have compared to NTFS.

FAT only tracks three timestamps per file (SI_1, SI_2 and SI_4). The SI_1 timestamp is stored in 10 milliseconds (or 0.01 seconds), and the SI_2 timestamp is stored with a resolution of two seconds.[2] The SI_4 timestamp is different: it is stored with an accuracy of one day for FAT. However, none of the operations under consideration transfer the SI_4 timestamp from FAT to NTFS systems. FAT also does not store any timezone information, whereas NTFS stores timestamps in UTC, making them 'resistant' to time zone changes or daylight savings time. exFAT (Extended File Allocation Table) solves some of the limitations FAT has. From a timestamp perspective, the same set of timestamps is available, i.e. SI_1, SI_2 and SI_4. The accuracy of both the SI_1 and SI_2 timestamps is now 10 milliseconds. SI_4 has a resolution of 2 seconds, but again, no operation under consideration transfers it to NTFS.

Directories. We follow Bang et al.'s approach [1] to measure timestamp effect of operations on directories separately from that on regular files. This makes sense, as directories are MFT entries like regular files, except that some operations (overwriting copy, overwriting move from another volume) cannot be applied to them.

[2] https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times

5.2 Measuring the effect of operations

For each combination of the considered file operations and file operation modifiers, we determine the effect on NTFS timestamps. As mentioned previously, the official documentation and existing literature is insufficient to determine these effects. Therefore, we perform experiments to measure the impact of each operation on timestamps. The process of performing the experiments consists of four steps:

(1) Set up the environment with the volumes and files to be tested.
(2) Read all the timestamps of the testfiles.
(3) Perform the file operation on all the testfiles.
(4) Read all the timestamps again and determine all the differences.

We perform tests for all Windows versions from Windows XP to Windows 11, for the following setup:

- Windows settings with last access updating enabled or disabled
- Files with filesizes ranging from 1 kb to 10 GB.
- Files with different extensions (i.e. .jpg, .exe, .txt)

Experimental setup. Since NTFS file timestamps are stored in the Master File Table, the effects of a file-operation can be observed in the MFT. We used RawCopy[3] to make an image of the test-volume before and after the file operations are performed, and used Mft2Csv[4] to extract the timestamps in a human-readable form from the MFT.

Table 1: Timestamp effect of file operations (no modifiers)

Operation	SI', FN'			
Create	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Access	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Update	$(SI_1,$	$op_{end},$	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Delete	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Rename	$(SI_1,$	$SI_2,$	$op_{start},$	$SI_4)$
	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$
Attribute change	$(SI_1,$	$SI_2,$	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Copy	$(op_{start},$	$src.SI_2,$	$op_{end},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwriting copy	$(SI_1,$	$src.SI_2,$	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Move within volume	$(SI_1,$	$SI_2,$	$op_{start},$	$SI_4)$
	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$
Mv. fr. other vol.	$(src.SI_1,$	$src.SI_2,$	$op_{end},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwr. mv. fr. other vol.	$(src.SI_1,$	$src.SI_2,$	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$

[3]https://github.com/jschicht/RawCopy
[4]https://github.com/jschicht/Mft2Csv

5.3 Experiment results

Notation. In this work, we consider file state as determined by the timestamps of an NTFS file. We denote the timestamps as $SI = (SI_1, SI_2, SI_3, SI_4)$ for the SI timestamps and the FN timestamps as $FN = (FN_1, FN_2, FN_3, FN_4)$. When considering the state transition resulting from a specific file operation, timestamps before the operation are denoted as SI and FN; timestamps post-operation are denoted as SI' and FN'. Finally, we denote the time when an operation starts as op_{start} and the time when it ends as op_{end}. Treating timestamps symbolically abstracts away from irrelevant details such as the speed of the storage device being used, and instead highlights the effect of the specific operation under investigation. We denote the effects of file operations under a modifier by showing the same table (with measurements for the non-modifier case) in gray, with any changes in regular black. Operations whose behaviour remains unchanged under a modifier are omitted from the table.

Table 1 shows the effects of the file operations on timestamps. These values were established without any modifiers enabled.

For example, for *copy* we see that the target file's timestamps are affected: all FN values are set to the starttime of the operation (op_{start}), the SI_2 timestamp is set to the source file's SI_2 timestamp ($src.SI_2$), SI_1 and SI_4 are set to operation starttime (op_{start}) and finally SI_3 is set to operation endtime (op_{end}). We found no difference in the effect of file-operations for the different versions of Windows, file size or file extension.

File Tunneling. Table 2 lists the operations whose behaviour is affected by file tunneling, and their behaviour under this operation. In short, in some cases, SI_1 and FN_1 take their value from the original (now removed) file's SI_1 and FN_1 timestamps, respectively.

Table 2: Modifiers: File Tunneling ("safe save").

Operation	SI', FN'			
Create	$(del.SI_1,$	$op_{start},$	$op_{start},$	$op_{start})$
	$(del.FN_1,$	$op_{start},$	$op_{start},$	$op_{start})$
Rename	$(del.SI_1,$	$SI_2,$	$op_{start},$	$SI_4)$
	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$
Copy	$(del.SI_1,$	$op_{start},$	$op_{start},$	$op_{start})$
	$(del.FN_1,$	$op_{start},$	$op_{start},$	$op_{start})$
Move within volume	$(del.SI_1,$	$SI_2,$	$op_{start},$	$SI_4)$
	$(SI_1,$	$SI_2,$	$SI_3,$	$SI_4)$

del: the file with the same exact path and name as this file, that was deleted prior to this operation (within the File Tunneling time window).

Last access updating. Effect on timestamps of file operations when last access updating is active is presented in Table 3. If the system crashes before last access updating is effectuated, the timestamps will not be updated beyond what was presented in Section 5.3. From the table, it is clear that enabling last access updating only affects timestamp SI_4.

Transfer from FAT / exFAT. The results for file operations regarding incoming files from FAT and exFAT are presented in Tables 4 and 5 respectively.

Table 3: Modifiers: Last access updating.

Operation	SI', FN'			
Access	$(SI_1,$	$SI_2,$	$SI_3,$	$op_{start})$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Update	$(SI_1,$	$op_{end},$	$op_{start},$	$op_{end})$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Copy	$(op_{start},$	$src.SI_2,$	$op_{end},$	$op_{end})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwriting copy	$(SI_1,$	$src.SI_2,$	$op_{start},$	$op_{start})$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Move from another volume	$(src.SI_1,$	$src.SI_2,$	$op_{end},$	$op_{end})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwr. move from other volume	$(src.SI_1,$	$src.SI_2,$	$op_{start},$	$op_{start})$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$

Table 4: Modifiers: Transfer from FAT

Operation	$SI',\ FN'$			
Copy	$(op_{start},$	valB,	$op_{end},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwriting copy	$(SI_1,$	valB,	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Move from FAT volume	(valA,	valB,	$op_{start},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwr. mv. from FAT vol.	(valA,	valB,	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$

valA: $src.SI_1$ rounded up to centiseconds (0.01 second) plus time zone difference (tzd);
valB: $src.SI_2$ rounded up to even seconds plus tzd.

Table 5: Modifiers: Transfer from exFAT

Operation	$SI',\ FN'$			
Copy	$(op_{start},$	valB,	$op_{end},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwriting copy	$(SI_1,$	valB,	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Move from exFAT volume	(valA,	valB,	$op_{end},$	$op_{start})$
	$(op_{start},$	$op_{start},$	$op_{start},$	$op_{start})$
Overwriting mv. from exFAT vol.	(valA,	valB,	$op_{start},$	$SI_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$

valA: $src.SI_1$ rounded up to centiseconds (0.01 second);
valB: $src.SI_2$ rounded up to centiseconds

Directories. We measured the effect of the considered file operations upon directories. The results of this are presented in Table 6; the only change from Table 1 is for the *update* operation, w.r.t. SI_4.

Table 6: Directories

Operation	SI', FN'			
Update	$(SI_1,$	$op_{end},$	$op_{start},$	$op_{end})$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Overwriting copy	operation not applicable to directories			
Overwriting mv. from another vol.	operation not applicable to directories			

5.4 Effects of timestamp forgery

Lastly, we illustrate the potential of our file histories method to detect timestamp forgery. To that end, we determine the effects of three timestamp forgery approaches: two basic Windows system calls and a popular timestamp forgery tool. The effects of each of these are presented in Table 7.

Approach 1: SetFileTime. Timestamps can be altered by the Windows system call *SetFileTime*. This system call allows to set the SI_1, SI_2, and SI_4 timestamps in whole seconds. Timestamp changing tools using this call are thus limited to changing only these three timestamps with conspicuous values. While in theory it is possible for these three timestamps to all three be exact whole seconds for most operations, in practice, this is a strong indication of tampering.

Approach 2: NtSetInformationFile. Another way of manipulating timestamps is through the undocumented *NtSetInformationFile* Windows system call. This system call can set the SI_1, SI_2 and SI_4 timestamps to any user specified values with full precision.

Approach 3: Timestomp. Lastly, we consider Timestomp, a timestamp manipulation tool. Timestomp uses the *NtSetInformationFile* system call [6]. Nevertheless, like other timestamp manipulation tools, it limits accuracy of altered timestamps to full seconds. These stand out from NTFS's default resolution of 100 nanoseconds. This is a strong indication of tampering, though most operations could theoretically result in such values.

Table 7: Effect of API / tool timestamp manipulation

Method	SI', FN			
`SetFileTime()`	(valA,	valB,	$op_{start},$	valD)
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
`NtSetInformationFile()`	$(any_1,$	$any_2,$	$any_3,$	$any_4)$
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$
Timestomp	(valA,	valB,	valC,	valD)
	$(FN_1,$	$FN_2,$	$FN_3,$	$FN_4)$

valA, valB, valC, valD: any value, rounded to whole seconds.
any_i: any value, up to full precision.

6 DEDUCING POSSIBLE FILE HISTORIES FROM NTFS TIMESTAMPS

With the effects of file operations on the *SI* and *FN* timestamps, we have laid the groundwork for reconstructing possible timelines. The concept will be introduced with a simplified running example. First, we can describe the state of a file along with its eight timestamps. When a file undergoes multiple file operations, its state and its timestamps change accordingly. For example, in Figure 1 a file is first created, then updated, and, lastly, renamed. After each state transition, its timestamps are updated (changes in bold red) according the timestamp rules described in the previous section.

We see that the *update* operation changes SI_2 and SI_3, leaving all other timestamps unchanged. Similarly, *rename* overwrites all *FN* timestamps with copies of the original's *SI* counterparts and changes SI_3, leaving the other three timestamps unchanged.

With the forwards evolution of timestamp established, we can now apply this reasoning backwards to determine previous allowed

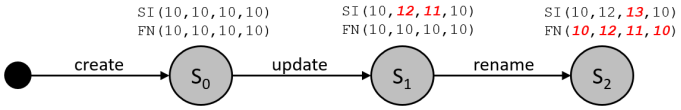

Figure 1: Example: forwards evolution of timestamps under file operations

states of the file. Assume that you find a file in a certain state with `SI(10, 12, 13, 10)` and `FN(10, 12, 11, 10)`. In Figure 1, this state was the result of the rename operation. If we now apply the inverse of the effect of the rename operation on the timestamp, we arrive at a state where we do not know any value of the *FN* timestamps, as they were overwritten. We do know what the *SI* timestamp should be in that previous state – namely, precisely the values of the *FN* timestamp of the current state. This is depicted graphically in Figure 2.

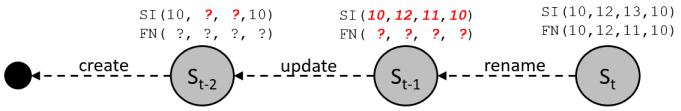

Figure 2: Example: backwards reasoning and information loss

The inverse effects of file operations on timestamps are described in Table 8. Some timestamps are overwritten by the operation; for these, no information of its previous state is available (denoted as '?' in the table). This table allows us to consider not one, but all operations at each turn. For each file state under consideration, we can try to apply the inverse of each file operations. Not all inverses will be possible, for example, for state S_t in Figure 2, only rename and copy (source) are possible. No other file operations could result in that state's timestamps. In general, most file operations can only result in certain values for timestamps. For example, the timestamps following a create operation are all equal. Therefore, any file state whose timestamps are not all equal, cannot be the result of a create operation. In other words, this imposes a constraint on (relations between) values for timestamps which can follow from a create operation.

Note that in Table 8, unlike in Table 1, we must distinguish between source and target for overwriting copies/moves. This is because for overwriting copy and overwriting move, two files must have existed previously: source and target. For all other operations, there is only one "ancestor" file.

For a state to result from a specific file operation, certain conditions have to be met, depending on the operation. These conditions are specified in Table 9. In addition to operation-specific constraints, there are constraints related to operation start/end time. We refer to these collectively as 'OPTIMING'. The first type of OPTIMING constraint is that operations cannot end before they start. Thus, all timestamps set to op_{start} must have values smaller than or equal to any timestamp set to op_{end}. Second, under a monotonically increasing system clock, operation start/end time must always be later than timestamps copied from the previous state. The last OPTIMING constraint is that if more than one timestamp is set to op_{start}, all such timestamps must be equal. This holds similarly for op_{end}.

Table 8: Previous state for file operations for $SI^t = (SI_1, SI_2, SI_3, SI_4)$, $FN^t = (FN_1, FN_2, FN_3, FN_4)$

Operation	SI^{t-1}, FN^{t-1}			
Create	N/A N/A			
Access	$(SI_1,$ $(FN_1,$	$SI_2,$ $FN_2,$	$SI_3,$ $FN_3,$	$SI_4)$ $FN_4)$
Update	$(SI_1,$ $(FN_1,$	?, $FN_2,$	?, $FN_3,$	$SI_4)$ $FN_4)$
Delete	$(SI_1,$ $(FN_1,$	$SI_2,$ $FN_2,$	$SI_3,$ $FN_3,$	$SI_4)$ $FN_4)$
Rename	$(FN_1,$ (?,	$FN_2,$?,	$FN_3,$?,	$FN_4)$?)
Attribute change	$(SI_1,$ $(FN_1,$	$SI_2,$ $FN_2,$	?, $FN_3,$	$SI_4)$ $FN_4)$
Copy (source)	(?, (?,	$SI_2,$?,	?, ?,	?) ?)
Overwriting copy				
– target	$(SI_1,$ $(FN_1,$	?, $FN_2,$	?, $FN_3,$	$SI_4)$ $FN_4)$
– source	(?, (?,	$SI_2,$?,	?, ?,	?) ?)
Move within volume	$(FN_1,$ (?,	$FN_2,$?,	$FN_3,$?,	$FN_4)$?)
Move from another volume (source)	$(SI_1,$ (?,	$SI_2,$?,	?, ?,	?) ?)
Overwriting move from another NTFS volume				
– target	(?, $(FN_1,$	?, $FN_2,$	?, $FN_3,$	$SI_4)$ $FN_4)$
– source	$(SI_1,$ (?,	$SI_2,$?,	?, ?,	?) ?)

For brevity, we abstract away from stating all such constraints explicitly and denote these as 'OPTIMING' in Table 9. The op_{start} and op_{end} columns indicate which timestamps are set to operation starttime and which for operation endtime, respectively. Lastly, note that, for both overwriting operations, the constraints apply equally to both source and target, and there is no further information on which to base any further constraints for either.

Table 9: Constraints operations impose on timestamps

Operation	Constraint	op_{start}	op_{end}
Create	$SI_i = FN_j$, for $i, j \in \{1, \ldots, 4\}$.	$SI_{1\ldots4}, FN_{1\ldots4}$	
Access	True.		
Update	OPTIMING.	SI_3	SI_2
Delete	True.		
Rename	OPTIMING and $FN_i = SI_i, i \in \{1, 2, 4\}$.	SI_3	
Attribute change	OPTIMING.	SI_3	
Copy	OPTIMING.	$SI_{1,4}, FN_{1\ldots4}$	SI_3
Overwriting copy	OPTIMING.	SI_3	
Move within volume	OPTIMING and $FN_i = SI_i, i \in \{1, 2, 4\}$.		SI_3
Move from another volume	OPTIMING	$SI_4, FN_{1\ldots4}$	SI_3
Overwriting move from other NTFS volume	OPTIMING.	SI_3	

Using these constraints, we can extend the example of Figure 2. A more extended, but still not complete, example of backwards reasoning is shown in Figure 3. We see branching, when a state could have been the result from more than one operation. We also see that information loss occurs, e.g., when timestamps are overwritten as by the rename operation. Lastly, we see that applying

the backwards reasoning process recursively, we can end up in a state where no information on timestamp values is known anymore (S'_{t-2} in the figure). There might still be more file history preceding this state, but nothing is known about this.

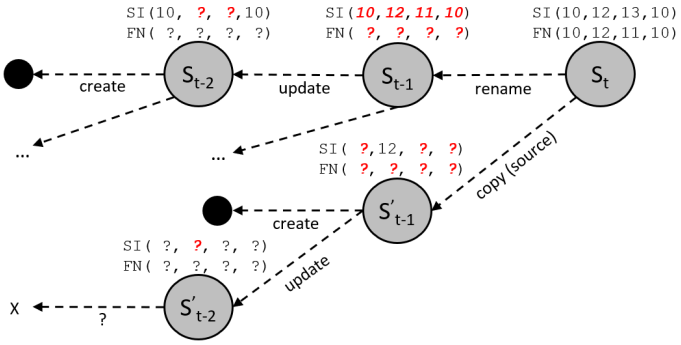

Figure 3: Example: backwards reasoning with branching and different branch-lengths

7 PROOF OF CONCEPT IMPLEMENTATION

We implemented and realized the presented methodology of reasoning backwards in two proof-of-concept tools: a timestamp analyzer and a visualisation tool.

7.1 Timestamp analyzer

This tool is configured with a list of file operations, where, for each operation, its effect on timestamps must be specified. We supply the list of operations and effects as discussed in the previous section. Operations that do not affect any timestamp are removed from consideration, as these can occur infinitely often at all points in a timeline. For our previously established list, this concerns first, the *access* operation when last-access-updating is not active, and second, the *delete* operation. Below, we describe three aspects (one concept and two algorithms) of the implementation.

Tracking information loss. To track loss of information in progressive steps of reasoning backwards, we use the concept of markings. A timestamp is marked when its value was changed due to the preceding file operation. This implies a loss of information: the value of a marked timestamp prior to the file operation that caused its marking is unknown and may not be used.

File operations can also *un*mark a timestamp. Unmarking is the reverse of marking: it denotes a gain of information, which occurs when a timestamp's new value originates from a known source. For example: the *rename* operation sets SI'_3 to op_{start}(which is thus marked), but also sets FN'_1 to SI_1, FN'_2 to SI_2, etc. Thus, we can still derive each of $SI_{1...4}$ from $FN'_{1...4}$.

Match-operation algorithm. The matching algorithm is responsible for identifying file operations compatible with the current state. As input, it is given the timestamps, markings, and a candidate file operation. If no constraint is violated, then the given state could have resulted from this operation and the algorithm returns `True`. In addition to the criteria presented in Table 9, the algorithm also takes constraints following from modifiers (Tables 3–6). This includes time-resolution constraints following from modifiers 'transfer from FAT/exFAT' (Tables 4, 5).

Timelines construction algorithm. The Timelines construction algorithm recursively builds all possible timelines of a given file state. Each timeline consists of a sequence of one or more file operations matching the state of the file at that point in the sequence (timestamps, markings). A sequence terminates when it encounters a *create* operation, or when no more operations can be matched to the file state. The latter can occur either due to complete loss of information, or when no file operation can match the available information.

Detecting timestamp forgery. Our proof-of-concept implementation considers timestamp forgery for timestamps. Our tooling adds a forgery branch only when (1) not all information has been lost yet (2) no regular operation can have caused the timestamps under consideration, and (3) the timestamps match the requirements induced by Table 7.

Implementation limitations. The proof-of-concept has only been tested on a limited number of MFTs and has not been optimised; quality attributes such as scalability and performance were not part of our testing process.

7.2 Visualisation tool implementation

This tool creates a visualisation of all possible timelines based on the output of the analysis tool. To illustrate the use of this visualisation tool, we show in Figure 4 a visualisation of the analysis of a file that underwent comparable file operations as our running example. The only difference is that these are real timestamps, instead of simple numbers. The visualisation is oriented like a timeline, with time increasing to the right. States further left are older, with the current (known) state depicted rightmost. The time at which an operation has taken place is noted in the black titlebar. When multiple operations lead to an identical state, they are grouped together for readability purposes. A file state that does not contain sufficient information to match any file operation is preceded by a question mark instead of another file state. This denotes that there is insufficient information to go back further in time.

8 DISCUSSION

System clock monotonicity. Our method relies on the key assumption that the system clock is monotonically increasing. This cannot be correct in all circumstances: switches from daylight savings time to regular time set the clock back. Similar problems may occur when system time is synchronised (e.g., via NTP) and the system clock drifts too far ahead, requiring the system clock to be wound back. Moreover, it is typically trivial to change the system clock.

Information loss. Our method for backwards reasoning inherently suffers from information loss. Therefore, each reconstructed timeline has a finite horizon of how far back our method can reconstruct possible file states. Possible file histories beyond a file's horizons cannot be reconstructed via this method.

Forensic limitations. Our approach supports reasoning about any file operation. However, our proof-of-concept analysis tool is only seeded with a list of basic file operations. Concretely, it lacks

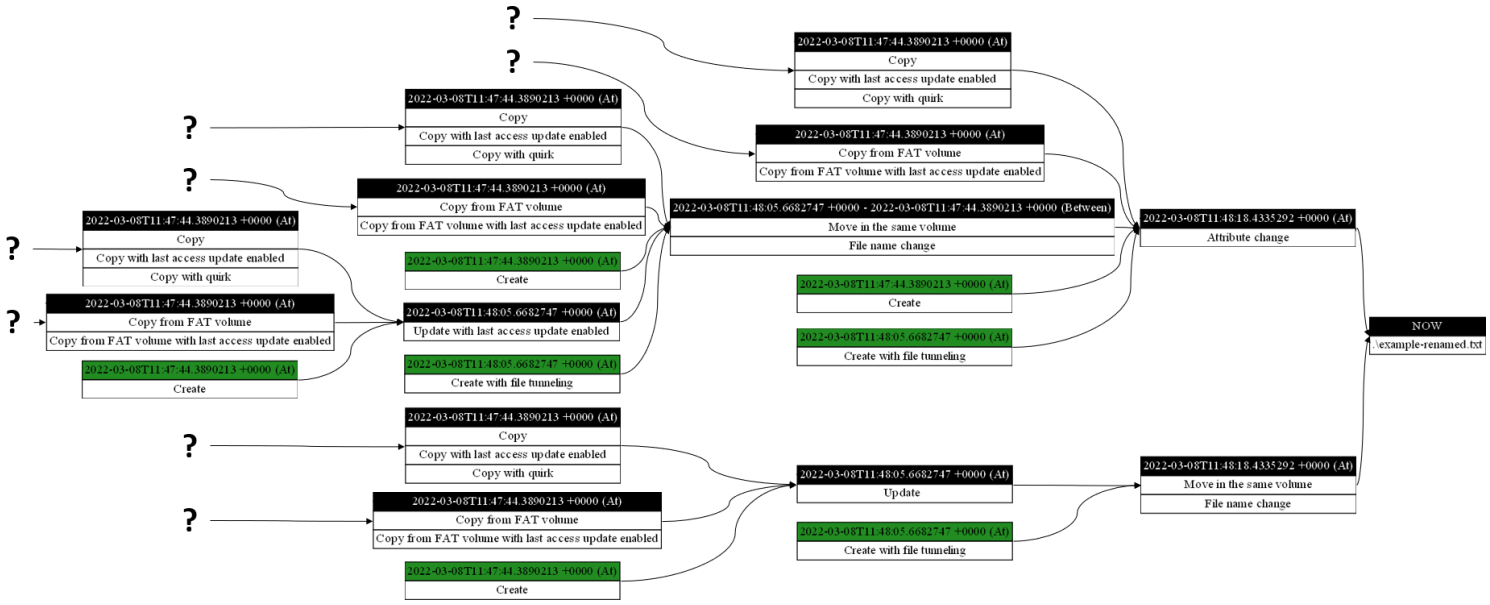

Figure 4: Full backwards reasoning running example

application-specific file operations. We recommend practitioners to follow our approach to determine the timestamp effects of file operations for any applications that they wish to incorporate into the reasoning framework. Expanding the list suffices to incorporate new file operations into the tooling.

9 CONCLUSION AND FUTURE WORK

In this paper we presented a novel method to reconstruct an overview of allowed histories of a file based on its timestamps. We discussed how to measure the timestamp effect of operations with and without modifiers (file tunneling and last access updating), and apply this to a set of basic file operations. The measurements reveal that operations can result in a file state where there are specific relations between some of the timestamps (Table 8). Our method leverages this to derive previous states: a certain file state cannot be the result of a given file operation, unless the state's timestamps all satisfy the relations required by the operation (Table 9). Thus, certain operations can be excluded as having caused the current state. This gives a set of possible previous file states. By recursively applying this process, our method can construct all allowed histories of a file, for the operations under consideration. This is fundamentally different from prior timestamp studies, which limit their approach to identifying only the last possible operation. In contrast, our method constructs a set of timelines of events. Lastly, we implemented our method in a proof-of-concept and showed viability of this approach to construct file histories.

The described method can be readily used by practitioners to reconstruct possible file histories. This will help them to substantiate or refute timeline-related hypotheses.

Future work. First, to improve practical use, the list of file operations can be expanded with application-specific file modifiers.

Second, the presented method of reconstructing possible file histories is based on the timestamp values stored in the MFT. Possible timeline reconstruction can be improved upon when sources of previous timestamp information can be included, such as log files, link files, or prefetch files [12]. With such additional points of information, information loss would be reduced and therefore the timeline horizon could be improved.

Last, the algorithms of the proof-of-concept tool are operation-agnostic. That is, they will accept any list of operations. Moreover, information loss ensures that our process for determining timelines always terminates for the operations from Table 1. It is possible that information loss of some other, not yet considered operations is insufficient to guarantee termination. That is, some operations could exist that jointly cause cyclical patterns in the reconstructed timeline. How to detect this and how to handle such occurrences are left for future work.

Acknowledgements. Van der Meer was supported by the Netherlands Organisation for Scientific Research (NWO) through Doctoral Grant for Teachers number 023.012.047.

REFERENCES

[1] Jewan Bang, Byeongyeong Yoo, Jongsung Kim, and Sangjin Lee. 2009. Analysis of Time Information for Digital Investigation. In *International Conference on Networked Computing and Advanced Information Management (NCM'09), Fifth International Joint Conference on INC, IMS and IDC: INC 2009: International Conference on Networked Computing, IMS 2009: International Conference on Advanced Information Management and Service, IDC 2009: International Conference on Digital Content, Multimedia Technology and its Applications, Seoul, Korea, August 25-27, 2009*. IEEE Computer Society, 1858–1864.

[2] 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.

[3] Florian Buchholz and Eugene Spafford. 2004. On the role of file system metadata in digital forensics. *Digital Investigation* 1, 4 (2004), 298–309.

[4] Gyu-Sang Cho. 2013. A computer forensic method for detecting timestamp forgery in NTFS. *Comput. Secur.* 34 (2013), 36–46.

[5] Gyu-Sang Cho. 2014. An Intuitive Computer Forensic Method by Timestamp Changing Patterns. In *Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'14)*. IEEE Computer Society, 542–548.

[6] Gyu-Sang Cho. 2016. Data Hiding in NTFS Timestamps for Anti-Forensics. *International Journal of Internet, Broadcasting and Communication* 8, 3 (2016), 31–40.

[7] Kam-Pui Chow, Michael Y. K. Kwan, Frank Y. W. Law, and Pierre K. Y. Lai. 2007. The Rules of Time on NTFS File System. In *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*. IEEE Computer Society, 71–85.

[8] Xiaoqin Ding and Hengming Zou. 2011. Time based data forensic and cross-reference analysis. In *Proceedings of the 2011 ACM Symposium on Applied Computing, Computer Forensics track (CF@SAC'11)*. ACM, 185–190.

[9] Michael Galhuber and Robert Luh. 2021. Time for Truth: Forensic Analysis of NTFS Timestamps. In *16th International Conference on Availability, Reliability and Security (ARES'21)*. ACM, 44:1–44:10.

[10] Dae-il Jang, Gail-Joon Ahn, Hyunuk Hwang, and Kibom Kim. 2016. Understanding Anti-forensic Techniques with Timestamp Manipulation (Invited Paper). In *17th IEEE International Conference on Information Reuse and Integration (IRI'16)*. IEEE Computer Society, 609–614.

[11] Rune Nordvik and Stefan Axelsson. 2022. It is about time–Do exFAT implementations handle timestamps correctly? *Forensic Science International: Digital Investigation* 42 (2022), 301476.

[12] 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.

[13] Bradley L. Schatz, George M. Mohay, and Andrew J. Clark. 2006. A correlation method for establishing provenance of timestamps in digital evidence. *Digital Investigations* 3, Supplement (2006), 98–107.

[14] Svein Yngvar Willassen. 2008. Hypothesis-Based Investigation of Digital Timestamps. In *4th Annual IFIP WG 11.9 Conference on Digital Forensics (Digital Forensics'08) (IFIP, Vol. 285)*. Springer, 75–86.