Analyzing Windows Subsystem for Linux Metadata to Detect Timestamp Forgery

Bhupendra Singh, Gaurav Gupta

Chapter 9

ANALYZING WINDOWS SUBSYSTEM FOR LINUX METADATA TO DETECT TIMESTAMP FORGERY

Bhupendra Singh and Gaurav Gupta

Abstract Timestamp patterns assist forensic analysts in detecting user activities, especially operations performed on files and folders. However, the Windows Subsystem for Linux feature in Windows 10 versions 1607 and later enables users to access and manipulate NTFS files using Linux command-line tools within the Bash shell. Therefore, forensic analysts should consider the timestamp patterns generated by file operations performed using Windows command-line utilities and Linux tools within the Bash shell.

This chapter describes the identification of timestamp patterns of various file operations in stand-alone NTFS and Ext4 filesystems as well as file interactions between the filesystems. Experiments are performed to analyze the anti-forensic capabilities of file timestamp changing utilities – called timestomping tools – on NTFS and Ext4 filesystems. The forensic implications of timestamp patterns and timestomping are also discussed.

Keywords: Anti-forensics, Windows Subsystem for Linux, timestamps, forgery

1. Introduction

Anti-forensic techniques and tools are increasingly used to circumvent digital forensic investigations. Several definitions of anti-forensics have been proposed over the years [2, 11, 12, 17]. According to Garfinkel [11], anti-forensics seeks to frustrate forensic tools, investigations and investigators. Conlan et al. [6] identify data hiding, encryption, data destruction, steganography and trail obfuscation as notable anti-forensic techniques. In their anti-forensics taxonomy, filesystem manipulation is a subcategory of data hiding. The modification of file timestamps

– often called "timestomping" – was pioneered by the `timestomp` utility [10]. Timestomping is the intentional alteration of created, modified or accessed timestamps of files or directories in the filesystem of a hard drive, USB stick, flash memory card or other storage device. Because timestamps are vital to event reconstruction and timeline creation, the authenticity and reliability of timestamps extracted from storage media are vital to forensic investigations [3].

The widespread use of anti-forensic tools, such as privacy cleaners (e.g., `CCleaner`) and timestomping utilities (e.g., `SetMACE`), has made the identification and analysis of suspicious files increasingly difficult. Moreover, advanced malware programs use anti-forensic techniques to persist and remain hidden in target systems [19]. These techniques include timestomping, data hiding and filesystem tunneling [5]. For example, Albano et al. [1] have presented an anti-forensic approach that leverages the Linux `touch` command to manipulate the last modified timestamp of an `mmssms.db` database in order to modify or delete evidence in an Android device.

Filesystem metadata analysis is an important component of digital forensic investigations. This chapter focuses on filesystem timestamp patterns that can be used to detect date and time forgery in stand-alone NTFS and Ext4 filesystems as well as in file interactions between the filesystems. The Windows Subsystem for Linux (WSL) feature in Windows 10 enables users to modify, access and delete files and folders in NTFS using Linux tools within a Windows Subsystem for Linux Bash shell. Moreover, users can launch Windows programs and store apps within the shell without leaving (or limiting) evidence in Prefetch files and other sources of program execution data; these traces would have been recorded if similar actions were performed using Windows Explorer. Malicious users can leverage the Windows Subsystem for Linux to manipulate file data and metadata using Linux commands such as `touch` and `shred`. Therefore, it is important to investigate the forensic implications of the Windows Subsystem for Linux feature with a focus on file metadata in hybrid filesystems. The research literature has just two works that discuss the forensic implications of the Windows Subsystem for Linux, one is a blog post [13] and the other is a research article [16].

2. Filesystem Timestamps

This section presents details about timestamps in the widely-used NTFS and Ext4 filesystems and their time resolutions. The time resolutions of other filesystems, including the newer APFS (Apple Filesys-

tem), are provided to understand anti-forensic techniques that may be used against them.

2.1 NTFS Timestamps and Time Resolutions

NTFS is a complex and robust filesystem used by default in computers running Windows NT 3.1 and later versions. NTFS timestamps are stored as 8-byte file time values that represent the number of 100-nanosecond intervals elapsed since 12:00 A.M. January 1, 1601. Consequently, NTFS timestamps have 100 nanosecond precision.

In addition to MAC (modified, accessed and changed) timestamps, NTFS, unlike other filesystems, also has a birth (i.e., file creation) timestamp. Thus, NTFS metadata contains four types of timestamps for each file on disk: (i) modified (file last modified); (ii) accessed (file last accessed); (iii) created (file created); and (iv) master file table (MFT) entry last modified. These timestamps are commonly referred to by their acronyms – MACE.

Several operating systems allow updates of the access time to be disabled. This means that the access time in a filesystem entry is not updated when the associated file is accessed. However, a user can change the default access time update. For example, the access time in Windows is controlled by the `HKLM\SYSTEM\CurrentControlSet\Control\FileSystem\NtfsDisableLastAccessUpdate` registry key, where a value of one disables access time updates.

NTFS has two locations in $MFT where timestamps are recorded – $STANDARD_INFORMATION (or $SI) and $FILE_NAME (or $FN). The $SI timestamps are collected by Windows Explorer and by `fls`, `mactime`, `timestomp`, `find` and other utilities that can display timestamps. The timestamp values in the $SI attribute can be modified by user processes.

NTFS maintains a duplicate set of timestamp values in the $FN attribute. For a given $MFT entry, multiple $FN attributes may exist. Because the $FN attribute can only be modified by the Windows kernel, the MACE values in $FN are updated in a different manner and are inconsistent with the MACE values in the $SI attribute. In fact, timestamp values in $FN are untouched by most timestamp manipulation tools.

It is important to note that the timestamp values in $SI and $FN update independently. For example, when a user accesses or modifies file content, then the temporal values in $SI are updated whereas the temporal values in $FN are updated when the user performs a move or copy operation. Analysis of the temporal values of these two attributes

can enable a forensic analyst to detect anomalies and timestomping. Additionally, the information can help an analyst create an accurate timeline of user activities in the system being investigated.

2.2 Ext4 Timestamps and Time Resolutions

Ext4, the successor to the standard Linux filesystem Ext3, is the default filesystem in most Linux distributions. The filesystem has introduced many new features, including the maximum file size (16 GB to 16 TB vs. 16 GB to 2 TB in Ext3), directory capacity (64,000 subdirectories vs. 32,000 in Ext3), journal checksum, journaling feature disabling option, delayed and multiple block allocations, large inodes, used inode count and fast `e2fsck` and fast extended attributes [7–9, 15]. These features improve the performance and reliability of Ext4 compared with the Ext3 filesystem.

In addition to these features, the inode structure in Ext4 is extended to return seconds and nanoseconds since the Unix epoch (1970-01-01 00:00:00 UTC). Five timestamps are stored in an Ext4 filesystem: (i) file last modification (m-time); (ii) file last access (a-time); (iii) inode metadata changed (c-time); (iv) file creation (cr-time); and (v) file deletion (d-time). The first four timestamps are commonly referred to MACB, where B denotes the birth (or creation) timestamp.

The larger inode structure size of 256 bytes in Ext4 provides additional space to support nanosecond timestamps and postpones the "year 2038 problem" to 2446-05-10 [20]. Mathur et al. [15] have shown that the 32 bits for the c-time, a-time, m-time and cr-time timestamps in Ext3 are extended to 64 bits in Ext4. However, d-time is not extended and remains a 32-bit timestamp in Ext4 with a precision of one second.

The Linux `stat` API enables users to access MAC timestamps up to nanosecond precision. The regular `stat` command does not report the cr-time and d-time timestamps. However, users can view all five timestamps with nanosecond precision using the TSK `istat` command or the `debugfs` version of `stat`. Note that the TSK `istat` command does not consider the extra epoch bits and, therefore, cannot resolve timestamps beyond the year 2038.

Table 1 shows the timestamp resolutions in various filesystems, along with their epoch dates and times.

3. Experiments and Results

The Windows 10 Anniversary Update (v1607) shipped with the beta version of the Windows Subsystem for Linux feature brings Windows and Linux platforms together. The feature was introduced to reduce

Table 1. Filesystem time resolutions and epoch dates and times.

Filesystem	**Timestamps**	**Size (Bytes)**	**Resolution**	**Epoch Date and Time**
Ext3	File modified	4	1 s	1970-01-01
	File accessed	4	1 s	00:00:00
	Inode metadata modified	4	1 s	
	File deleted	4	1 s	
FAT32	File modified	4	2 s	1980-01-01
	File accessed	2	1 day	00:00:00
	File created	4	2 s	
Ext4	File modified	8	1 ns	1970-01-01
	File accessed	8	1 ns	00:00:00
	Inode metadata modified	8	1 ns	
	File created	8	1 ns	
	File deleted	4	1 s	
NTFS	File modified	8	100 ns	1601-01-01
	File accessed	8	100 ns	00:00:00
	MFT entry modified	8	100 ns	
	File created	8	100 ns	
HFS+	File modified	4	1 s	1904-01-01
	File accessed	4	1 s	00:00:00 and
	Inode metadata modified	4	1 s	1970-01-01 00:00:00 since
	File created	4	1 s	Mac OS-X 10.7 (Lion)
APFS	File modified	8	1 ns	1970-01-01
	File accessed	8	1 ns	00:00:00
	Inode metadata	8	1 ns	
	modified	8	1 ns	
	File created	8	1 ns	

the "gaps" experienced when running Windows tools alongside Linux command-line tools and environments. Since its introduction, the Windows Subsystem for Linux has continuously improved Windows-Linux integration. The Windows 10 Creators Update (v1703) enables users to invoke Windows applications, store apps within a Linux Bash shell and use Linux mainstream developers tools within Windows. Microsoft has stated that it does not explicitly support X/GUI apps/desktops in the

Windows Subsystem for Linux because the intent is only to provide the needed command-line developers tools.

The Windows Subsystem for Linux feature in Windows 10 introduces exciting possibilities for digital forensics. In its Windows 10 Fall Creators Update (v1709), Microsoft announced that the Windows Subsystem for Linux would be a fully-supported operating system feature that would enable users to install multiple Linux distributions and to run them side-by-side simultaneously. Support for mounting USB storage devices is also provided to enable users to access files and folders from within the Linux Bash shell. Clearly, a deep understanding of the Windows Subsystem for Linux feature is required in order for a forensic analyst to correctly interpret timestamp values in filesystems of interest.

3.1 Stand-Alone NTFS Filesystems

The Windows Subsystem for Linux feature in Windows 10 (Anniversary Update and later versions) enables users to perform various file operations in NTFS using mainstream developer command-line tools within the Linux Bash shell. Therefore, an attempt was made to identify timestamp patterns in NTFS for various file operations performed using command-line tools within the Windows Subsystem for Linux. Windows command-line tools and Linux command-line tools were employed to manipulate files. Changes in the MACE timestamps in the $SI and $FN attributes of files were analyzed.

The experimental set-up involved the installation and configuration of Ubuntu and Windows Subsystem for Linux on a personal computer running Windows 10 Pro x64 with version 1709 (build 16299.371). The Ubuntu version was 16.04.03 LTS with core command-line tools (`ssh`, `scp`, `apt`, `grep`, `top`, `awk`, etc.) and mainstream developer tools (`emacs`, `vim`, `nano`, `gdb`, `git`, etc.). Following the configuration of Ubuntu, TSK (v4.2.0.3) was installed using `apt` within the Bash shell. FTK Imager was installed on the Windows system to create the `dd` image of the NTFS volume. The MACE timestamps in the $SI and $FN attributes corresponding to files of interest were extracted using the `istat` command.

The experiments focused on several file operations – creation, access, modification, renaming, copying, moving (same volume and across volumes), deletion, compression and decompression. Before performing file operations, several reference files were selected to record their MACE timestamps in $SI and $FN attributes using `istat` on the `dd` image of the NTFS volume. Note that `istat` requires the inode number ($MFT entry number for NTFS) of the file; this was obtained by parsing $MFT using `Mft2Csv` [18]. Following a file operation, the MACE timestamps

were collected for the reference files in the newly-created image of the NTFS volume. This process was repeated for every file operation considered in the timestamp pattern evaluations.

- **Timestamp Rules for File Creation:** These timestamp rules were determined by creating several files using `echo`, `copy` and `fsutil` within the Windows command-line, and `touch` within the Linux Bash shell. It was observed that, whenever a file was created, the MACE timestamps in the $SI and $FN attributes corresponded to the date and time that the file was created.

- **Timestamp Rules for File Access:** These timestamp rules were determined by reopening reference files using their default applications (called the standard GUI mechanism), and using the `cat` and `nano` commands within the Bash shell. It was observed that, when a file in an NTFS volume was accessed using the standard mechanism, then, by default, the em-time timestamp in the $SI attribute was updated to the date and time that the file was last accessed. However, no changes to the MACE timestamps in $FN were observed. Also, when files were accessed using `cat` and `nano`, no changes to the MACE timestamps in the $SI and $FN attributes were observed.

- **Timestamp Rules for File Modification:** These timestamp rules were determined by modifying the reference files using the standard GUI mechanism, and using `echo` and `powershell` within the Windows command-line and `nano` within the Bash shell. The `powershell` command-line shell, which was introduced in Windows 7, also provides a way to modify a file. In all the cases, the m-time and em-time timestamps in the $SI attribute were updated to correspond to the date and time when the file was modified. No changes to the MACE timestamps in $FN were observed.

- **Timestamp Rules for File Renaming:** These timestamp rules were determined by renaming the reference files via Windows Explorer (standard mechanism), `rename` within the Windows command line, and `rename` and `mv` within the Bash shell. It was observed that, in all four cases, em-time in $SI was updated to the date and time when the file was renamed. Moreover, em-time in $FN was changed to the last em-time in $SI. Thus, during normal NTFS operations, em-time in $SI is greater than or equal to em-time in $FN.

- **Timestamp Rules for File Copying:** These timestamp rules were determined by copying and pasting reference files via Win-

dows Explorer, and using `copy` within the Windows command-line and `cp` within the Bash shell. It was observed that, when a copy operation was performed using Windows Explorer or `copy`, the m-time and em-time timestamps in $SI were inherited from the original file. However, the a-time and c-time timestamps were changed to the date and time when the copied file was created on disk. Also, all four timestamps in $FN were updated in every case. When the `cp` command was used within the Bash shell, all the MACE timestamps in $SI and $FN attributes were changed.

- **Timestamp Rules for File Moving:** These timestamp rules were determined by moving reference files within the same NTFS volume and to a different NTFS volume using Windows Explorer, `move` within the Windows command-line and `mv` within the Bash shell. It was discovered that, when a file was moved using the standard mechanism within the same volume, only the $SI em-time was updated to the date and time when the file was moved. No changes were observed to the MACE timestamps in $FN when files were moved to the same volume. However, when a file was moved using `move` or `mv`, in addition to the $SI em-time being updated, the $FN em-time was updated to the last $SI em-time.

  In a second experiment, files were moved to a different NTFS volume using the same methods. It was discovered that the Windows `move` command and Bash shell `mv` command produced different timestamp patterns for a given file. Specifically, the Windows `move` preserved a-time and c-time in $SI whereas the Bash shell `mv` preserved only a-time. However, moving files across volumes changed all the MACE timestamps in the $FN attribute.

- **Timestamp Rules for File Deletion:** These rules were determined by deleting reference files using the Windows Explorer `SHIFT+DELETE`, Windows command-line `del` and Bash shell `rm`. It was discovered that, when a file in an NTFS volume was deleted, none of MACE timestamps in $SI and $FN were changed. In other words, it is very difficult to estimate the deletion dates and times of NTFS files from metadata alone.

- **Timestamp Rules for File Compression:** These rules were determined by compressing files and folders in several ways, such as using `WinZip`, a customized `VBScript`, `7-Zip` and Bash shell `tar`. It was discovered that all the file compression methods created a new file that recorded the MACE timestamps in $SI and $FN as the file compression date and time.

- **Timestamp Rules for File Decompression:** These rules were determined by decompressing files using `WinZip`, `7-Zip` and Bash shell `unzip`. It was discovered that different tools yielded different timestamp patterns. For example, when a compressed file was decompressed using `WinZip`, only the $SI em-time was changed to the file decompression date and time whereas the $SI c-time, m-time and a-time were unchanged from the c-time, m-time and a-time before compression. When a compressed file was decompressed using `7-Zip`, the $SI c-time, em-time and a-time were changed to the file decompression date and time whereas m-time was unchanged from the m-time before compression. If the compressed file was decompressed using the Bash shell `unzip`, then the $SI c-time and em-time were changed to the file decompression date and time whereas the m-time and a-time were unchanged from the m-time and a-time before compression, respectively. However, for all the decompression methods used in the experiments, the $FN MACE timestamps were changed to the file decompression date and time.

Tables 2 through 4 summarize the patterns observed for various operations on NTFS files.

3.2 Stand-Alone Ext4 Filesystems

Several experiments were performed to determine timestamp patterns for various file operations in an Ext4 filesystem.

- **Timestamp Rules for File Creation:** These rules were determined by creating several files using `touch`, `echo` and `cat` within a Ubuntu terminal. It was discovered that, when files were created, the inode MACB timestamps corresponded to the file creation dates and times.
- **Timestamp Rules for File Access:** These rules were determined by reopening reference files using the standard GUI mechanism, and the `cat` and `nano` commands within a Ubuntu terminal. No changes were observed to any of the timestamps.
- **Timestamp Rules for File Modification:** These rules were determined by modifying reference files using the standard GUI mechanism, and the `nano` and `vim` commands within a Ubuntu terminal. It was observed that, in all cases, m-time, a-time and c-time were updated to the dates and times when the files were modified. However, cr-time was unchanged during file modification.

Table 2. Timestamp patterns observed for operations on files in NTFS.

Operation	**Method**	**NTFS Location**	**Timestamps**			
			File Modified (m-time)	**File Accessed (a-time)**	**Entry Modified (em-time)**	**File Created (c-time)**
Creation	Standard GUI	$SI	Creation date, time	Creation date, time	Creation date, time	Creation date, time
		$FN	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	`echo`	$SI	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	command-line	$FN	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	`copy`	$SI	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	command-line	$FN	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	`fsutil`	$SI	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	command-line	$FN	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	`touch`	$SI	Creation date, time	Creation date, time	Creation date, time	Creation date, time
	Bash shell	$FN	Creation date, time	Creation date, time	Creation date, time	Creation date, time
Access	Standard GUI	$SI	Not changed	Not changed	Access date, time	Not changed
		$FN	Not changed	Not changed	Not changed	Not changed
	`cat`	$SI	Not changed	Not changed	Not changed	Not changed
	Bash shell	$FN	Not changed	Not changed	Not changed	Not changed
	`nano`	$SI	Not changed	Not changed	Not changed	Not changed
	Bash shell	$FN	Not changed	Not changed	Not changed	Not changed
Modification	Standard GUI	$SI	Modification date, time	Not changed	Modification date, time	Not changed
		$FN	Not changed	Not changed	Not changed	Not changed
	`echo`	$SI	Modification date, time	Not changed	Modification date, time	Not changed
	command-line	$FN	Not changed	Not changed	Not changed	Not changed
	`powershell`	$SI	Modification date, time	Not changed	Modification date, time	Not changed
	command-line	$FN	Not changed	Not changed	Not changed	Not changed
	`nano`	$SI	Modification date & time	Not changed	Modification date & time	Not changed
	Bash shell	$FN	Not changed	Not changed	Not changed	Not changed

Table 3. Timestamp patterns observed for operations on files in NTFS (continued).

Operation	**Method**	**NTFS Location**	**Timestamps**			
			File Modified (m-time)	**File Accessed (a-time)**	**Entry Modified (em-time)**	**File Created (c-time)**
Rename	Standard	$SI	Not changed	Not changed	Rename date, time	Not changed
	Windows Explorer	$FN	Not changed	Not changed	Not changed	Not changed
	`rename`	$SI	Not changed	Not changed	Rename date, time	Not changed
	command-line	$FN	Not changed	Not changed	Last $SI em-time	Not changed
	`rename`	$SI	Not changed	Not changed	Rename date, time	Not changed
	Bash shell	$FN	Not changed	Not changed	Last $SI em-time	Not changed
	`mv`	$SI	Not changed	Not changed	Rename date, time	Not changed
	Bash shell	$FN	Not changed	Not changed	Last $SI em-time	Not changed
Copy	Standard	$SI	Not changed	Copy date, time	Not changed	Copy date, time
	GUI	$FN	Copy date, time	Copy date, time	Copy date, time	Copy date, time
	`copy`	$SI	Not changed	Copy date, time	Not changed	Copy date, time
	command-line	$FN	Copy date, time	Copy date, time	Copy date, time	Copy date, time
	`cp`	$SI	Copy date, time	Copy date, time	Copy date, time	Copy date, time
	Bash shell	$FN	Copy date, time	Copy date, time	Copy date, time	Copy date, time
Move (Same Volume)	Standard	$SI	Not changed	Not changed	Move date, time	Not changed
	Windows Explorer	$FN	Not changed	Not changed	Last $SI em-time	Not changed
	`move`	$SI	Not changed	Not changed	Move date, time	Not changed
	command-line	$FN	Not changed	Not changed	Last $SI em-time	Not changed
	`mv`	$SI	Not changed	Not changed	Move date, time	Not changed
	Bash shell	$FN	Not changed	Not changed	Last $SI em-time	Not changed
Move (Across Volumes)	Standard	$SI	Not changed	Move date, time	Not changed	Move date, time
	Windows Explorer	$FN	Move date & time	Move date, time	Move date & time	Move date, time
	`move`	$SI	Not changed	Move date, time	Not changed	Move date, time
	command-line	$FN	Move date, time	Move date, time	Move date, time	Move date, time
	`mv`	$SI	Move date, time	Not changed	Move date & time	Move date, time
	Bash shell	$FN	Move date, time	Move date, time	Move date, time	Move date, time

Table 4. Timestamp patterns observed for operations on files in NTFS (continued).

Operation	**Method**	**NTFS Location**	**Timestamps**			
			File Modified (m-time)	**File Accessed (a-time)**	**Entry Modified (em-time)**	**File Created (c-time)**
Deletion	Standard	$SI	Not changed	Not changed	Not changed	Not changed
	Windows Explorer	$FN	Not changed	Not changed	Not changed	Not changed
	`SHIFT+DELETE`	$SI	Not changed	Not changed	Not changed	Not changed
		$FN	Not changed	Not changed	Not changed	Not changed
	`del`	$SI	Not changed	Not changed	Not changed	Not changed
	command-line	$FN	Not changed	Not changed	Not changed	Not changed
	`rm`	$SI	Not changed	Not changed	Not changed	Not changed
	Bash shell	$FN	Not changed	Not changed	Not changed	Not changed
Compression	`WinZip` and	$SI	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
	`7-Zip`	$FN	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
	`VBScript`	$SI	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
	command-line	$FN	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
	`tar`	$SI	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
	Bash shell	$FN	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time
Decompression	`WinZip`	$SI	File last m-time	File last a-time	Decomp. date, time	File last c-time
		$FN	Decomp. date, time	Decomp. date, time	Decomp. date, time	Decomp. date, time
	`7-Zip`	$SI	File last m-time	Decomp. date, time	Decomp. date, time	Decomp. date, time
		$FN	Decomp. date, time	Decomp. date, time	Decomp. date, time	Decomp. date, time
	`unzip`	$SI	File last m-time	File last a-time	Decomp. date, time	Decomp. date, time
	Bash shell	$FN	Decomp. date, time	Decomp. date, time	Decomp. date, time	Decomp. date, time

- **Timestamp Rules for File Renaming:** These rules were determined by renaming reference files using the standard GUI mechanism, and the `rename` and `mv` commands within a Ubuntu terminal. It was discovered that a-time and c-time were updated to the date and time when the renaming operation was performed whereas m-time and cr-time were unchanged.

- **Timestamp Rules for File Copying:** These rules were determined by copying reference files using the standard GUI mechanism and the `cp` command within a Ubuntu terminal. It was discovered that the m-time of a copied file was inherited from the original file whereas the a-time, c-time and cr-time were updated to the date and time of the copy operation. However, if the file was copied using the `cp` command, then all the MACB timestamps were changed to the date and time of the copy operation.

- **Timestamp Rules for File Moving:** These rules were determined by moving reference files to the same Ext4 volume as well as to another Ext4 volume using the standard GUI mechanism and the `mv` command within a Ubuntu terminal. In all cases, a-time and c-time were changed to the date and time of the move operation. However, m-time and cr-time were unchanged after all the move operations.

- **Timestamp Rules for File Deletion:** These rules were determined by deleting files using the standard `SHIFT+DELETE`, and the `rm` and `shred` commands within a Ubuntu terminal. It was discovered that, in addition to the m-time and c-time, the Ext4 filesystem recorded the d-time of the particular file, and all the timestamp values were updated to the date and time when the file was deleted. However, a-time and cr-time were unchanged after all the deletion operations.

- **Timestamp Rules for File Compression:** These rules were determined by compressing files and folders in an Ext4 volume using the standard GUI mechanism, and the `zip`, `tar` and `gzip` commands within a Linux terminal. It was discovered that, if a file or folder was compressed using the standard GUI mechanism or using `zip` or `tar`, then the inode MACB timestamps stored in the volume corresponded to the date and time when the file or folder was compressed. However, different timestamp patterns were observed for different compression algorithms. For example, when the `gzip` command-line tool within a Ubuntu Bash terminal was used, m-time and a-time were not updated. However, c-time and

cr-time were updated to the date and time when the compression command was executed.

- **Timestamp Rules for File Decompression:** These rules were determined by decompressing several files using the standard GUI mechanism, and the `unzip`, `tar` and `gzip` commands within a Ubuntu terminal. It was discovered that, when a compressed file was extracted or decompressed, the timestamp patterns depended on the decompression method used. For example, `unzip` and `tar` left m-time unchanged from the file last modified time (just before compression); the other three timestamps were updated to the date and time when the zipped file was decompressed. In the case of `gzip`, m-time and a-time were unchanged.

Tables 5 and 6 summarize the patterns observed for various operations on Ext4 files.

3.3 NTFS-Ext4 File Transfers

Experiments were conducted to identify the timestamp patterns of file transfers to and from Windows NTFS and Linux Ext4 volumes. In the experiments, a personal computer was set up to dual boot with Microsoft Windows 10 v1709 x64 and Ubuntu 16.04.03 LTS. The TSK tool was installed using `apt` within a Ubuntu Bash terminal. To enable file transfers between NTFS and Ext4, the Windows volumes were mounted in Ubuntu and files were transferred using methods such as the standard GUI mechanism, and `cp` and `mv` within a Ubuntu Bash terminal. The `istat` tool was used to collect timestamps before and after each file was transferred.

In the case of file transfers from NTFS to Ext4 using the standard GUI, and `cp` and `mv` within a Ubuntu Bash terminal, it was discovered that, for all the cases shown in Table 7, at least a-time, c-time and cr-time were changed to the file transfer date and time. However, file transfers using `cp` changed all the inode MACB timestamps. Also, file transfers from NTFS to Ext4 using the standard GUI and `mv` within a Ubuntu Bash terminal caused the inode m-time to be inherited from $SI m-time.

When a file was transferred from Ext4 to NTFS using the standard GUI, and `cp` and `mv` within a Ubuntu Bash terminal, all the MACE timestamps in $FN were assigned the date and time when the file was transferred. Also, a-time, em-time and c-time in $SI were updated to the file transfer date and time. However, except for the file transfer using `cp`, m-time in $SI was not updated. In fact, it inherited the inode m-time of

Table 5. Timestamp patterns observed for operations on files in Ext4.

Operation	**Method**	**Timestamps**				
		File Modified (m-time)	**File Accessed (a-time)**	**Inode Modified (c-time)**	**File Created (cr-time)**	**File Deleted (d-time)**
Creation	`touch`	Creation date, time	Creation date, time	Creation date, time	Creation date, time	NA
	`cat`	Creation date, time	Creation date, time	Creation date, time	Creation date, time	NA
	`echo`	Creation date, time	Creation date, time	Creation date, time	Creation date, time	NA
Access	GUI	Not changed	Not changed	Not changed	Not changed	NA
	`cat`	Not changed	Not changed	Not changed	Not changed	NA
	`nano`	Not changed	Not changed	Not changed	Not changed	NA
Modification	GUI	Modification date, time	Modification date, time	Modification date, time	Not changed	NA
	`nano`	Modification date, time	Modification date, time	Modification date, time	Not changed	NA
	`vim`	Modification date, time	Modification date, time	Modification date, time	Not changed	NA
Rename	GUI	Not changed	Rename date, time	Rename date, time	Not changed	NA
	`rename`	Not changed	Rename date, time	Rename date, time	Not changed	NA
	`mv`	Not changed	Rename date, time	Rename date, time	Not changed	NA
Copy	GUI	Not changed	Copy date, time	Copy date, time	Copy date, time	NA
	`cp`	Copy date & time	Copy date, time	Copy date, time	Copy date, time	NA
Move	GUI	Not changed	Move date, time	Move date, time	Not changed	NA
	Move To Trash	Not changed	Move date, time	Move date, time	Not changed	NA
	`mv`	Not changed	Move date, time	Move date, time	Not changed	NA

Table 6. Timestamp patterns observed for operations on files in Ext4.

Operation	**Method**	**Timestamps**				
		File Modified (m-time)	**File Accessed (a-time)**	**Inode Modified (c-time)**	**File Created (cr-time)**	**File Deleted (d-time)**
Deletion	Move to Trash and Delete	Del. date, time	Not changed	Del. date, time	Not changed	Del. date, time
	`SHIFT+DELETE`	Del. date, time	Not changed	Del. date, time	Not changed	Del. date, time
	`rm`	Del. date, time	Not changed	Del. date, time	Not changed	Del. date, time
	`shred`	Del. date, time	Not changed	Del. date, time	Not changed	Del. date, time
Compression	GUI	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time	NA
	`tar`	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time	NA
	`zip`	Comp. date, time	Comp. date, time	Comp. date, time	Comp. date, time	NA
	`gzip`	Not changed	Not changed	Comp. date, time	Comp. date, time	NA
Decompression	GUI	File last m-time	Decomp. date, time	Decomp. date, time	Decomp. date, time	NA
	`unzip`	File last m-time	Decomp. date, time	Decomp. date, time	Decomp. date, time	NA
	`tar`	File last m-time	Decomp. date, time	Decomp. date, time	Decomp. date, time	NA
	`gzip`	Not changed	Not changed	Decomp. date, time	Decomp. date, time	NA

Table 7. Timestamp patterns observed during NTFS to Ext4 file transfers.

Operation	Transfer Method	Timestamps			
		File Modified (m-time)	**File Accessed (a-time)**	**Inode Modified (c-time)**	**File Created (cr-time)**
Copy	File copy via GUI	Not changed	Copy date and time	Copy date and time	Copy date and time
Copy	File copy via `cp` in Bash shell	Copy date and time	Copy date and time	Copy date and time	Copy date and time
Move	File move via GUI	Not changed	Move date and time	Move date and time	Move date and time
Move	File move via `mv` in Bash shell	Not changed	Move date and time	Move date and time	Move date and time

the file in the Ext4 volume. Table 8 summarizes the timestamp patterns observed during the Ext4 to NTFS file transfers.

3.4 Timestomping Tool Capabilities

Experiments were conducted with several timestomping utilities and command-line tools to imvestigate evidence tampering in NTFS and Ext4 filesystems.

Six utilities, `BulkFileChanger`, `Attribute Changer`, `SKTimeStamp`, `FS Touch`, `SetMACE` and `AttributeMagic`, were used to alter the MACE timestamps in an NTFS volume. The utilities were installed on a Windows 10 system and various (newly created and existing) files in a disk volume were considered as reference files.

Before any timestamps were modified, the MACE timestamps corresponding to $SI and $FN were collected from the reference files in an NTFS volume image (i.e., `dd` image created using FTK Imager). The timestamps were extracted using the TSK `istat` command within a Bash shell. The inode number (MFT entry number for NTFS) of a file required by `istat` was obtained by parsing $MFT using `Mft2Csv`.

Each of the six anti-forensic utilities was executed to change the timestamps of the reference files to future times. A `dd` image of the NTFS volume was then created and `istat` was executed to extract MACE timestamps from the $SI and $FN attributes of the reference files. It

Table 8. Timestamp patterns observed during Ext4 to NTFS file transfers.

Operation	Transfer Method	NTFS Location	Timestamps			
			File Modified (m-time)	File Accessed (a-time)	Entry Modified (em-time)	File Created (c-time)
Copy	File copy via GUI	$SI	Not changed	Copy date and time	Copy date and time	Copy date and time
		$FN	Copy date and time	Copy date and time	Copy date and time	Copy date and time
Copy	File copy via `cp` in Bash shell	$SI	Copy date and time	Copy date and time	Copy date and time	Copy date and time
		$FN	Copy date and time	Copy date and time	Copy date and time	Copy date and time
Move	File move via GUI	$SI	Not changed	Move date and time	Move date and time	Move date and time
		$FN	Move date and time	Move date and time	Move date and time	Copy date and time
Move	File move via `cp` in Bash shell	$SI	Note changed	Move date and time	Move date and time	Move date and time
		$FN	Move date and time	Move date and time	Move date and time	Move date and time

is believed that is the best methodology for comparing the anti-forensic capabilities of timestomping utilities.

By and large, the six timestomping tools could not set the $FN MACE timestamps. In fact, only the `SetMACE` command-line tool could alter the $FN MACE timestamps. Additionally, whereas all the tools could alter the $SI MAC timestamps, no tool – except for `SetMACE` – could alter the $SI em-time. However, `BulkFileChanger`, `Attribute Changer`, `SKTimeStamp`, `FS Touch` and `AttributeMagic` set the em-time to the date and time when the tool was executed. These five tools were unable to set the nanoseconds portion of the date and time; instead, they set all nine digits after the seconds part to zeroes. `FS Touch` could only change the $SI MAC timestamps up to the milliseconds part (three digits after the seconds part).

Only the `SetMACE` command-line tool was able to successfully alter all the MACE timestamps in $SI and $FN with nanosecond precision. This is because, unlike the other tools, `SetMACE` performs direct disk accesses to manipulate the MACE timestamps in $SI and $FN, as well as in the `$INDEX_ROOT` and `$INDEX_ALLOCATION` attributes. `SetMACE` accomplishes this using a driver that bypasses the filesystem and writes directly

Table 9. Timestomping capabilities of six anti-forensic tools on NTFS.

Tool	Timestamps	File Modified	File Accessed	Entry Modified	File Created
`BulkFileChanger` v1.51	$SI	✓	✓	●	✓
	$FN	✗	✗	✗	✗
`Attribute Changer v9.0a`	$SI	✓	✓	●	✓
	$FN	✗	✗	✗	✗
`SKTimeStamp` v1.3.5	$SI	✓	✓	●	✓
	$FN	✗	✗	✗	✗
`FS Touch v7.3`	$SI	✓	✓	●	✓
	$FN	✗	✗	✗	✗
`SetMACE` v1.0.0.14	$SI	✔	✔	✔	✔
	$FN	✔	✔	✔	✔
`AttributeMagic` v2.4	$SI	✓	✓	●	✓
	$FN	✗	✗	✗	✗

to the disk without leaving any traces in NTFS metadata (`$LogFile` and `$UsnJrnl`), provided that the adversary has elevated disk access privileges. That means that `SetMACE` resolves the filesystem internally and writes the timestamps directly to the physical disk, bypassing filesystem and operating system control mechanisms. As a result, detecting traces of `SetMACE` execution is extremely difficult.

Table 9 compares the timestomping capabilities of the six anti-forensic tools on NTFS. The ✓ symbol denotes that the timestamp was changed, but the nanoseconds part was zeroed. The ● symbol indicates that the timestamp was changed to the date and time when the utility was executed. The ✓ symbol denotes that the timestamp was changed, but only up to the milliseconds part. The ✔ symbol signifies that the timestamp was changed, including the nanoseconds part. Finally, the ✗ symbol indicates that the timestamp was not changed.

Five utilities, `chmod`, `chattr`, `touch`, `SetMACB` and `BulkFileChanger`, were used to alter the MACB timestamps in an Ext4 filesystem. TSK was installed on a computer running Ubuntu 16.04 LTS and several (newly created and existing) files in the Ext4 volume were considered as reference files. `BulkFileChanger` was executed on the Ubuntu 16.04 LTS system using the `Wine` package, which is capable of running Windows applications on several POSIX-compliant operating systems. Before any timestamps were modified, the MACB timestamps corresponding to the reference files in the Ext4 volume were extracted using `stat` (when the

file inode number was known) and `istat` (otherwise). Following this, each tool was executed to manipulate the MACB timestamps of the reference files. The new timestamps then were recorded.

The analysis revealed that `chmod` updated only the accessed (a-time) and inode modified (c-time) timestamps whereas `chattr` only updated c-time. The `touch` command (with default options) updated a-time and c-time to the current date and time. The `touch` command was also able to alter the accessed and file modified timestamps to a specific date and time using the `-a` option for a-time and the `-m` option for m-time. For example, `touch -a -m -t 201612061104.45 test.txt` changed the m-time and a-time of `test.txt` to `2016-12-06 11:04:45.000000000`, but c-time was updated to the date and time when the command was executed. If the system date and time were correct, then c-time would have the correct date and time; however, if the system time was manipulated, then c-time would have the incorrect date and time.

The `touch` command also changed m-time and a-time up to the nanoseconds part using the `-d` option. For example, `touch -d "2016-12-06 11:04:45.123456789" test.txt` changed the m-time and a-time of the file `test.txt` to `2016-12-06 11:04:45.123456789`, but c-time was updated to the date and time when the command was executed.

The `touch` command also copied the timestamps of a file to a target file using option `-r`. For example, `touch test.txt -r sample.txt` set the m-time of `test.txt` file to the m-time of `sample.txt`. Also, the a-time and c-time of the `test.txt` file were updated to the date and time when the command was executed; however, the created timestamp (cr-time) was untouched.

The `BulkFileChanger` utility was unable to manipulate cr-time; it only changed the date and time of m-time and left the nanoseconds part to be all zeroes.

However, it is possible to manipulate all the MACB timestamps of a file in an Ext4 filesystem. A workaround procedure, referred to as `SetMACB` in Table 10, was created to successfully manipulate all four MACB timestamps with nanosecond precision. The `SetMACB` procedure involved the following steps:

- Alter the system time to the desired date and time using the `date` command-line tool.
- Create a new file with the same content or copy-paste an existing file whose timestamps need to be manipulated.
- Update the a-time and m-time of the file using the `touch` command-line tool.

Table 10. Timestomping capabilities of five anti-forensic tools on Ext4 filesystems.

Tool	Options	File Modified	File Accessed	Inode Modified	File Created
`chmod`	`775`	✗	✗	●	✗
`chattr`	`+a`	✗	✗	●	✗
`touch`	`-a -t`	✗	✓	●	✗
	`-a -m -t`	✓	✓	●	✗
	`-d`	✔	✔	●	✗
	`-r`	✔	✔	●	✗
`SetMACB`	See text	✔	✔	✔	✔
`BulkFileChanger`	Using `Wine`	✓	●	●	✗

- Reset the system time to the current date and time using the `date` command-line tool.

Table 10 compares the timestomping capabilities of the five anti-forensic tools on the Ext4 filesystem. The ✓ symbol denotes that the timestamp was changed, but the nanoseconds part was zeroed. The ● symbol indicates that the timestamp was changed to the date and time when the utility was executed. The ✔ symbol signifies that the timestamp was changed, including the nanoseconds part. Finally, the ✗ symbol indicates that the timestamp was not changed.

4. Discussion

Analyzing timestamps in a filesystem to their full precision is important to detect timestamp forgery. The experiments reveal that, by and large, the evaluated timestomping tools could set the file created, modified and accessed timestamps to specified dates and times with precisions of seconds, leaving the nanoseconds parts as zeroes. Thus, if the nanoseconds part of any NTFS MACE or Ext4 MACB timestamp (except for inode d-time in Ext4) contains all zeroes, then timestamp forgery is indicated. However, this is not always true because, when a file is copied or moved from a USB device (FAT32 filesystem) to an NTFS or Ext4 volume, it inherits the m-time (in $SI only for NTFS), but the nanoseconds part has all zeroes because the time resolution of the last modified time in a FAT32 filesystem only has a precision of seconds. In the case of an NTFS file, because of the 100 ns interval, there is one chance out of $4,782,969$ ($= 9^7$) that the nanoseconds part of a timestamp would be

all zeroes by default. In the case of an Ext4 file, the chance is only one out of $387,420,489$ $(= 9^9)$.

The experiments also reveal that the `touch` timestamp manipulation tool behaves differently in the NTFS and Ext4 filesystems. In the case of NTFS, `touch` (by default), updates a-time, m-time and em-time to the system date and time in the $SI attribute whereas all four timestamps in the $FN attribute are not touched. In the case of Ext4, `touch` updates a-time, m-time and c-time to the system date and time. However, `touch` does not manipulate file creation timestamps in both the filesystems.

5. Conclusions

Timestamp patterns assist forensic analysts in detecting user activities in filesystems, especially operations performed on files. However, anti-forensic techniques such as timestomping can alter file created, modified and accessed timestamps in the filesystems of hard drives, USB sticks, flash memory cards and other storage devices. Because timestamps are vital to event reconstruction and timeline creation, the determination of the authenticity and reliability of timestamps extracted from storage media are vital in forensic investigations

The filesystem timestamp patterns specified in this chapter enable forensic analysts to detect date and time forgeries in stand-alone NTFS and Ext4 filesystems as well as forgeries related to file transfers between the two filesystems. The analysis of well-known file timestamp changing utilities (timestomping tools) on NTFS and Ext4 filesystems provides valuable insights into their anti-forensic capabilities. Timestamp anomalies can be detected by leveraging timestamp patterns and analyzing timestamps to their full precision.

The research described in this chapter has focused on the Windows Subsystem for Linux feature in Windows 10 systems. Since this feature is still evolving, it is expected that the timestamp patterns would vary in future versions of Windows 10. Nevertheless, this research has demonstrated how timestamp patterns and the capabilities of anti-forensic tools can be systematically investigated to detect timestamp forgeries.

Future research will investigate the forensic implications of the Windows Subsystem for Linux feature with regard to the recovery of deleted files using Linux tools such as `rm`, `shred` and `srm`. Also, research will attempt to identify the sources and locations of execution artifacts created when Windows programs and apps are launched within a Bash shell using the Windows Subsystem for Linux feature.

References

[1] P. Albano, A. Castiglione, G. Cattaneo and A. De Santis, A novel anti-forensic technique for the Android OS, *Proceedings of the International Conference on Broadband and Wireless Computing, Communications and Applications*, pp. 380–385, 2011.

[2] I. Baggili, A. BaAbdallah, D. Al-Safi and A. Marrington, Research trends in digital forensic science: An empirical analysis of published research, *Proceedings of the Fourth International Conference on Digital Forensics and Cyber Crime*, pp. 144–157, 2012.

[3] F. Buchholz and E. Spafford, On the role of filesystem metadata in digital forensics, *Digital Investigation*, vol. 1(4), pp. 298–309, 2004.

[4] B. Carrier, *File System Forensic Analysis*, Pearson Education, Upper Saddle River, New Jersey, 2005.

[5] E. Casey, Digital stratigraphy: Contextual analysis of filesystem traces in forensic science, *Journal of Forensic Sciences*, vol. 63(5), pp. 1383–1391, 2018.

[6] K. Conlan, I. Baggili and F. Breitinger, Anti-forensics: Furthering digital forensic science through a new, extended, granular taxonomy, *Digital Investigation*, vol. 18(S), pp. S66–S75, 2016.

[7] A. Dewald and S. Seufert, AFEIC: Advanced forensic Ext4 inode carving, *Digital Investigation*, vol. 20(S), pp. S83–S91, 2017.

[8] K. Fairbanks, An analysis of Ext4 for digital forensics, *Digital Investigation*, vol. 9(S), pp. S118–S130, 2012.

[9] K. Fairbanks, C. Lee and H. Owen III, Forensic implications of Ext4, *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, article no. 22, 2010.

[10] J. Foster and V. Liu, Catch me, if you can, presented at *Black Hat Japan*, 2005.

[11] S. Garfinkel, Anti-forensics: Techniques, detection and countermeasures, *Proceedings of the Second International Conference on i-Warfare and Security*, pp. 77–84, 2007.

[12] R. Harris, Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem, *Digital Investigation*, vol. 3(S), pp. 44–49, 2006.

[13] A. Harrison, Further Forensicating of Windows Subsystem for Linux, *1234n6 Blog* (`www.blog.1234n6.com/2017/10/further-forensicating-of-windows.html`), October 17, 2017.

[14] S. Ho, D. Kao and W. Wu, Following the breadcrumbs: Timestamp pattern identification for cloud forensics, *Digital Investigation*, vol. 24, pp. 79–94, 2018.

[15] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas and L. Vivier, The new Ext4 filesystem: Current status and future plans, *Proceedings of the Linux Symposium*, vol. 2, pp. 21–34, 2007.

[16] L. Nathan, A. Case, A. Ali-Gombe and G. Richard III, Memory forensics and the Windows Subsystem for Linux, *Digital Investigation*, vol. 26(S), pp. S3–S11, 2018.

[17] M. Rogers, Anti-forensics: The coming wave in digital forensics, poster presentation at the *Seventh Annual CERIAS Information Security Symposium*, 2006.

[18] J. Schicht, `Mft2Csv`, GitHub (`www.github.com/jschicht/Mft2Csv/wiki/Mft2Csv`), May 20, 2017.

[19] B. Singh and U. Singh, Program execution analysis in Windows: A study of data sources, their formats and comparison of forensic capability, *Computers and Security*, vol. 74, pp. 94–114, 2018.

[20] D. Wong, Ext4 Disk Layout (`www.ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout`), February 18, 2019.