IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 April 2024, accepted 17 April 2024, date of publication 1 May 2024, date of current version 30 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3395644

RESEARCH ARTICLE

Forensic Detection of Timestamp Manipulation for Digital Forensic Investigation

JUNGHOON OH[1,2], SANGJIN LEE[1], AND HYUNUK HWANG[2], (Member, IEEE)
[1]School of Cybersecurity, Korea University, Seongbuk-gu, Seoul 02841, South Korea
[2]The Affiliated Institute of ETRI, Yuseong-gu, Daejeon 34044, South Korea

Corresponding author: Sangjin Lee (sangjin@korea.ac.kr)

ABSTRACT File system forensics is one of the most important areas of digital forensic investigations. To date, various file system forensic methods have been studied, of which anti-forensic countermeasures include deleted file recovery, metadata recovery, and metadata manipulation detection. In particular, manipulation detection of timestamps, which are important file metadata, is one of the key techniques in digital forensic investigations. Existing detection methods for file timestamp manipulation in the New Technology File System (NTFS) have been studied based on various file system and operating system artifacts. This paper compares and analyzes the features and limitations of various existing detection methods and confirms that the NTFS journal-based detection method is the most effectively way to detect timestamp manipulation. However, previous NTFS journal-based detection methods have limitations such as incorrectly identifying normal events as manipulation or detecting manipulation only in limited cases. Therefore, we propose a new detection algorithm that can overcome these limitations. The proposed detection algorithm was implemented as a tool and verified through performance comparison experiments with existing detection methods. The results of experiment showed that the proposed detection algorithm has significantly improved performance by detecting timestamp manipulations that were not detected by previous detection methods and identifying normal events that were misidentified by existing detection methods. Finally, we introduce a case in which existing detection methods and the proposed detection algorithm are applied to malware that performs file timestamp manipulation in real-world advanced persistent threat attacks. The results of which confirm the superiority of the proposed detection algorithm.

INDEX TERMS File system, forensics, anti-forensic countermeasures, timestamp manipulation, forensic detection.

I. INTRODUCTION

File system forensics is a branch of digital forensics that analyses and investigates the structure and contents of file systems [1]. Since most of the evidence collected during a digital forensic investigation is at the file level, file system forensics is the most basic and important element for forensic investigators. To date, various file system forensic methods have been studied, including tree structure analysis [1], [2], [3], [4], metadata analysis [5], [6], [7], [8], [9], [10], and anti-forensic countermeasures. Among these file system forensic methods, anti-forensic countermeasures include deleted file recovery [11], [12], metadata recovery [13], [14], [15], [16], [17], [18], [19], [20], and metadata manipulation detection [21], [22], [23], [24], [25], [26], [27]. In particular, manipulation detection of timestamps, which are important file metadata, is one of the key techniques in digital forensics investigations because it can reveal files that malicious users or attackers are trying to hide.

File timestamp manipulation is when a malicious user or attacker manipulates the timestamp of a file to avoid detection by a timeline analysis [28]. Timeline analysis is a chronological analysis of events extracted from the file system and artifacts during a digital forensic investigation. Once a specific trace is found, additional traces can be found by analyzing events that occurred before and after the trace [29]. For example, if an investigator finds in a timeline analysis that a specific malware execution event

The associate editor coordinating the review of this manuscript and approving it for publication was Donato Impedovo.

occurred, followed by a specific file creation event, the investigator may suspect that the file was created by malware. In the above case, if the attacker had manipulated the creation time of the generated file into the distant past, the investigator would not be able to detect the file in the timeline analysis. This file timestamp manipulation is mainly performed on files that remain on the system, such as backdoors. In fact, timestamp manipulation is second only to file deletion when it comes to erasing traces of an attack [30], meaning that timestamp manipulation is the most commonly performed operation to hide traces of undeleted files on the system.

Previous detection methods for file timestamp manipulation in the New Technology File System (NTFS), one of the most widely used file systems today, have been studied based on various file system and operating system artifacts. However, existing detection studies have been conducted for each artifact individually, and there has been no integrated analysis of the different detection methods. This paper compares and analyzes the features and limitations of these various existing detection methods and confirms that the NTFS journal-based detection method, which can directly detect timestamp manipulation behavior, can most effectively detect timestamp manipulation. However, previous NTFS journal-based detection methods have limitations such as detecting normal events as manipulation or detecting manipulation only in limited cases, which make them difficult to use in actual digital forensic investigations.

Therefore, we propose a new detection algorithm that can overcome the limitations of previous NTFS journal-based detection methods. To this end, the timestamp manipulation behavior of various timestamp manipulation tools and malware were studied, as well as a method for identifying file system tunneling [31], which is the main cause of false positives in existing detection methods. The proposed detection algorithm was implemented as a tool and verified through performance comparison experiments with existing detection methods. The results of experiment showed that the proposed detection algorithm has significantly improved performance by detecting timestamp manipulations that were not detected by previous detection methods, and not generating any false positives due to file system tunneling. Finally, we introduce a case in which existing detection methods and the proposed detection algorithm are applied to detect the manipulation of a malware that performs file timestamp manipulation in real-world advanced persistent threat (APT) attacks. In this case, the proposed detection algorithm not only detected additional detection factors that were not detected by existing detection methods, but also detected timestamp manipulation that was not detected by existing detection methods at all.

The main contributions of this paper were as follows:

- We propose a new detection algorithm that overcomes the limitations of existing NTFS journal-based detection methods. The proposed detection algorithm detects timestamp manipulations that are not detected by existing detection methods and does not generate false positives due to file system tunneling.

- We analyze the behavior of file timestamp manipulation of a malware used in a real APT attack and provide the results of applying existing detection methods and the proposed detection algorithm. The results confirm that the proposed detection algorithm is more useful in real-world digital forensic investigations.

- The implemented tools are released to contribute to the digital forensic community.

- We provided the results of a comparative analysis regarding the characteristics and limitations of different artifact-based detection methods. The results of this analysis can be used to effectively detect timestamp manipulation in various system environments and situations.

TABLE 1. NTFS timestamps.

Timestamp	Acronym	Detail
Last Modified Time	M	When the file was last modified.
Last Accessed Time	A	When the file was last accessed.
File Creation Time	C	When the file was created.
MFT Entry Modified Time	E	When the $MFT entry of the file was updated.

The remainder of this paper is organized as follows. Section II describes the background knowledge needed to understand the detection of file timestamp manipulation in NTFS. Section III introduces previous detection studies of timestamp manipulation in NTFS, and Section IV presents a comparative analysis of the features and limitations of the existing detection studies discussed in Section III. Section V describes a detection algorithm that improves existing NTFS journal-based detection methods. Section VI introduces the tool as implemented and describes the results of performance evaluation between the detection algorithm proposed in this paper and previous detection methods. Section VII introduces the results of applying previous detection methods and the proposed detection algorithm to malware that manipulates file timestamps used in real-world APT attacks. Section VIII discuss the limitations of the proposed detection algorithm and the process of determining timestamp manipulation in an integrated manner. Finally, Section IX summarizes the conclusions of this paper.

II. BACKGROUND KNOWLEDGE

A. NTFS TIMESTAMP & $MFT

NTFS, which is the main file system of the Windows operating system, stores the timestamp of each file as a 64bit value in 100-nanoseconds since January 1, 1601 [32]. There are four types of timestamp: Last Modified Time, Last Accessed Time, File Creation Time, and MFT Entry Modified Time. Each timestamp can be expressed in M, A, C, E abbreviation format [26]. In later papers, each timestamp is expressed as an abbreviation, as shown in Table 1.

The master file table ($MFT) is a file that stores the metadata of all files and folders in NTFS. It consists of

entry units, and the metadata of each file and folder are stored in one or more entries. Each entry consists of attribute units, and the four timestamps corresponding to MACE are stored in the $STANDARD_INFORMATION attribute and the $FILE_NAME attribute respectively. The timestamp of the $STANDARD_INFORMATION attribute is the time information of the file and folder that can be checked in Windows Explorer, and the timestamp in the $FILE_NAME attribute is updated when the file name or location in the volume is changed [1]. Therefore, each file and folder in NTFS has a total of eight timestamps.

In a later paper, the $STANDARD_INFORMATION attribute was renamed by reducing it to $SI, the $FILE_NAME attribute to $FN, and the two were expressed in combination with the MACE timestamp. For example, $SI-E means the MFT Entry Modified Time of the $STANDARD_INFORMATION attribute, and $SI-MC means the Last Modified Time and File Creation Time of the $STANDARD_INFORMATION attribute.

B. $LOGFILE

$LogFile, which is a metafile of the NTFS, is a log file that records data from such file system transactions as file creations, deletions, data changes, and name changes. Transaction data are stored in record units, where each record consists of redo data, which are the data updated to the $MFT entry, and undo data, which are the data before the update. If the file system is corrupted due to a system error, the operating system restores the file system to its normal state by using the undo data [1]. Therefore, $LogFile is an important artifact that provides information about all transaction operations performed in the file system during a specific period. The transaction data stored in the $LogFile file reflects how much data have been updated in which location in which attribute of which entry in the $MFT. Therefore, various studies have been conducted to extract human-recognizable file-level events (creation, deletion, renaming, movement) from these transaction data [6], [7], [8], [9], [10].

C. $USNJRNL

$UsnJrnl is a log file that stores NTFS change logs in record units and is used by applications to determine whether a particular file has changed. The format of the change log data includes the time when the change event occurred, the file or directory name in which the event occurred, and the event type [9]. Therefore, the $UsnJrnl is an important artifact to know all the change events that occurred in the file system during a specific period. Because $UsnJrnl records store data in a fixed format, there are many tools analyzing the data of $UsnJrnl [6], [7], [8], [33].

D. METHODS OF TIMESTAMP MANIPULATION

The methods for manipulating the timestamp of a file on a Windows system are as follows.

The first method is to use the SetFileTime() API [34]. The corresponding API can change the MAC among the timestamp of $SI. For $SI-E, this is changed to the time when the SetFileTime() API is used. The SetFileTime() API is mainly used in GUI-type timestamp manipulation tools [35], [36], [37], [38].

The second method is to use Powershell's Get-Item Cmdlet [39]. The corresponding method is executed by importing an object of a specific file through Get-Item Cmdlet and then setting a time value for the creationtime, last-writetime, and lastaccesstime attributes of the file object [40]. This method can be performed in the Powershell command window or by writing a Powershell script.

The third method is to use the NtSetInformationFile() API [41]. This API can change all timestamps (MACE) of $SI. The NtSetInformationFile() API is mainly used in CLI-type timestamp manipulation tools [42], [43], [44].

The fourth method is to change the timestamp of $FN. This method uses the feature that when a file is moved within the same volume, the timestamp of $SI is copied to the timestamp of $FN as is. The reason for using this method is that there is no API that can directly change the timestamp of $FN in Windows systems. For example, after changing the $SI-MACE of a specific file to a desired time using the NtSetInformationFile() API, and then moving the file to another location within the same volume, the changed $SI-MACE is copied to $FN-MACE. Finally, if $SI-E changed by the file movement is changed back to the desired time, all timestamps of the file can be changed to the desired time [26]. A tool that uses this method is SetMACE (v1.0.0.4) [44].

In addition, there is a way to change the timestamp by accessing the physical disk directly, but this method is currently unavailable when a Windows system is running because Microsoft patched it to prevent direct access to the system drive [26]. Therefore, this paper excludes the detection of timestamp manipulation by this method.

III. RELATED WORKS

Previous studies for detecting file timestamp manipulation in NTFS can largely be divided into methods that use file system artifacts and those that use operating system artifacts. Table 2 summarizes the classification and detailed methods of each type of detection method. In later papers, each detection method is represented as an alias, as shown in Table 2.

A. FILE SYSTEM ARTIFACTS-BASED METHODS

Existing detection methods using file system artifacts include using $MFT, which stores metadata for all files and directories in NTFS, and $LogFile and $UsnJrnl, which are NTFS journal files.

1) $MFT

The detection methods using $MFT are the following. The first method ($MFT-1), proposed by Ding and Zou [21] and Jang et al. [22], checks for normal rules for $SI/$FN

TABLE 2. Overview of existing detection methods for timestamp manipulation in NTFS.

Category	Artifact	Method Alias	Method Details
File System Artifact	$MFT	$MFT-1	Check $SI/$FN timestamp rules. If the conditions below are met, it is detected as manipulation. - $SI-M > $SI-E, $SI-C > $FN-C, $SI-C > $SI-A [21] - $SI-E < $SI-MAC, $SI-MACE | $FN-MACE > System time, $SI-E < $FN-E, $SI-MAC > $FN-MAC, $FN-CA < $FN-ME, $FN-M != $FN-E [22]
		$MFT-2	Check for continuity of sequence number and entry number in the $MFT entry. If a file falls outside the range of sequence number and entry number of other files created during the same time period, it is detected as manipulation [23].
		$MFT-3	Compares the timestamp of a child file stored in the $INDX attribute within the $MFT entry of a directory with the actual timestamp of the same file. If the two timestamps are different, it is detected as manipulation [24].
		$MFT-4	Check the 100-nanosecond unit of the timestamp. If the 100-nanosecond unit is 0, it is detected as manipulation [22].
	$LogFile	$LogFile-1	Check $SI-C change events. If a $SI-C change event exists, it is detected as manipulation [25].
		$LogFile-2	Compares the $SI-C of the file creation event in $LogFile with the $SI-C of the same file in $MFT. If the two timestamps are different, it is detected as manipulation [22].
	$UsnJrnl	$UsnJrnl-1	Compares the last BASIC_INFO_CHANGE event time of a specific file in $UsnJrnl with $SI-E of the same file in $MFT. If the two timestamps are not similar, it is detected as manipulation. [26]
Operating System Artifact	Prefetch, / Registy	Execution-1	Check for the execution of a known timestamp manipulation tool. The execution time of the timestamp manipulation tool can be known [26].
	LNK	LNK-1	Compares the timestamp of the LNK file with the timestamp of the linked file. If the conditions below are met, it is detected as manipulation [26]. - LNK.$SI-E != LinkedFile.$SI-E, LNK.$SI-C < LinkedFile.$SI-C, LNK.$SI-A > LinkedFile.$SI-A
	Event Log	EventLog-1	Compares the login duration of the user with $SI-C, $SI-M of the file. If $SI-C and $SI-M of a certain file fall outside the login duration of the user, it is detected as manipulation [26].
	Volume Shadow Copy	VSC-1	Compares the $SI timestamp of a specific file in current volume with $SI timestamp of the same file in the Volume Shadow Copies created on the system. If the $SI timestamp of a particular file in a previously created volume shadow copy is greater than the $SI timestamp of a later created volume shadow copy or the same file in the current volume, it is detected as manipulation [27].

timestamps in $MFT entries. This method looks at the timestamp form and change method of normal files without timestamp manipulation and creates a normal rule (e.g., $SI-M <= $SI-E) through this, nd determines that timestamp modulation has occurred in the file when a particular file violates it. Details of the normal rules disclosed so far can be found in Table 2. The second method ($MFT-2), proposed by Willassen [23], detects timestamp manipulation using the continuity of the sequence and entry numbers of the $MFT entries. This method uses the principle that $MFT entries are allocated sequentially when a file is created, and assumes that multiple files created at the same time will have the same sequence number and sequential entry number. If a specific file is outside the range of sequence number and entry number of other files created at the same time, it is determined that timestamp manipulation has occurred in that file. The third method ($MFT-3), proposed by Minnaard [24], compares the timestamp of a child file stored in the $INDX attribute within the directory's $MFT entry with the actual timestamp of the same file. This method is based on the fact that a tool (SetMACE v1.0.0.5 and later) that directly accesses the disk and changes the timestamp of the target file, but does not change the timestamp of the same file stored in the $INDX attribute within the $MFT entry of the parent directory. If the two timestamps are not the same for a specific file, it is determined that timestamp manipulation has occurred on that file. The fourth method ($MFT-4), proposed by Jang et al. [22] and Bouma et al. [60], is to check the 100-nanosecond unit of the $SI/$FN timestamp of the $MFT entry. This method takes advantage of the fact that the 100-nanosecond timestamp unit of the target file is set to zero when certain tools (e.g., Timestomp) use the SetFileTime() API to perform timestamp manipulations. Therefore, if the timestamp 100-nanosecond unit of a specific file is set to 0, it is determined that timestamp manipulation has occurred in that file.

2) $LOGFILE

The detection methods using $LogFile are the following. The first method ($LogFile-1), proposed by Cho [25], analyzes events in $LogFile to check for changes in the $SI-C of the file. This method checks for changes to $SI-C by analyzing the data in the $LogFile record (UpdateResidentValue) that modifies the data in the $MFT entry. If the

event of changing the $SI-C of a specific file is confirmed, it is determined that timestamp manipulation occurred at the time the $SI-C change event occurred. The second method ($LogFile-2), proposed by Jang et al. [22], analyses events in $LogFile to compare the $SI-C that can be extracted from file creation events with the $SI-C of the same file in $MFT. This method analyzes the data in the $LogFile record (InitialiseFileRecordSegment) that generates the $MFT entry to obtain the $SI-C of the generated $MFT entry and the $SI-C of the same file within $MFT to compare the two timestamps. If the two $SI-Cs for a specfic file are not identical, it is determined that a timestamp manipulation has occurred in that file.

3) $USNJRNL

The detection method ($UsnJrnl-1) using $UsnJrnl, proposed by Palmbach and Breitinger [26], compares the time of the last BASIC_INFO_CHANGE event of a specific file in $UsnJrnl with the $SI-E of the same file in $MFT. This method takes advantage of the fact that if a change operation on a file property was last performed on a specific file, the time of the last BASIC_INFO_CHANGE event for that file in $UsnJrnl and the $SI-E of the same file in $MFT have similar values. If the two time values for a specific file are not similar, it is determined that timestamp manipulation occurred at the time of the last BASIC_INFO_CHANGE event on that file.

B. OPERATING SYSTEM ARTIFACTS-BASED METHODS

The detection methods using operating system artifacts involve using prefetch, registry, LNK files, event log, and volume shadow copy.

1) PREFETCH AND REGISTRY

The method (Execution-1) using prefetch or registry, proposed by Palmbach and Breitinger [26], determines whether and when a timestamp manipulation tool was executed through program execution traces. This method uses the executable file name of a well-known timestamp manipulation tool (e.g., TimeStomp, SetMACE) as a signature to determine whether the tool has been executed and the execution time.

2) LNK FILES

The method (LNK-1) using LNK files, proposed by Palmbach and Breitinger [26], compares the timestamp of the LNK file with the timestamp of the linked target file. This method takes advantage of the fact that the $SI-A and $SI-E of the LNK file must be similar to the same timestamp of the link target file because the $SI-A and $SI-E of the LNK file are updated when a specific file is opened. It also uses the fact that the LNK file cannot be created before the link target file is created. Therefore, if the $SI-A and $SI-E of a specific file is significantly different from the same timestamp of the LNK file, or if the $SI-C of the LNK file is older than the $SI-C of the link target file, it is determined that timestamp manipulation has occurred in that file.

3) EVENT LOGS

The method (EventLog-1) using the Event Log, proposed by Palmbach and Breitinger [26], compares the user login duration, which can be obtained from the Event Log, with the $SI-C and $SI-M of the file. If the $SI-C and $SI-M of a specific file fall outside the time period that the user is logged in, it is determined that timestamp manipulation has occurred in that file.

4) VOLUME SHADOW COPY

The method (VSC-1) using volume shadow copy, proposed by Mohamed and Khalid [27], compares the $SI timestamp of a specific file in the current volume with the $SI timestamp of the same file in the volume shadow copies created on the system. This method determines that timestamp manipulation has occurred in the file if the $SI timestamp of a specific file in a previously created volume shadow copy is greater than the $SI timestamp of a later created volume shadow copy or the same file in the current volume.

IV. INTEGRATED ANALYSIS ON DETECTION METHODS

Looking at the detection methods for file timestamp manipulation in NTFS that have been studied so far, individual detection methods for each artifact have been studied, but an integrated analysis of various detection methods has not been conducted. Therefore, this section performs a comparative analysis of features such as detection type, detection level, detection target information, and anti-forensics resistance of the previous detection methods, as well as the limitations of each. Table 3 summarizes the features and limitations of each detection method.

In the case of detection type, existing detection methods were largely classified into ''Direct'' and ''Indirect'' types. The ''Direct'' type is a method that directly detects the timestamp change event itself, and includes the $LogFile-1 and the $UsnJrnl-1 methods. These methods can know in which file and when the timestamp manipulation occurred, and in addition, for $LogFile-1, the timestamp before and after manipulation can be checked. Therefore, detection methods of this type can provide investigators with information not only about the files used in the attack, but also about the time of the attack, which is most important for timeline analysis. The ''Indirect'' type is a method that detects the traces of manipulation behavior left behind rather than the timestamp manipulation itself, from which it is possible to know whether a timestamp manipulation has occurred in the target file or whether a timestamp manipulation tool has been executed.

In the case of detection level, existing detection methods can be classified into ''Malicious'', ''Suspicious'' and ''Additional'' levels. At the ''Malicious'' level, the possibility of timestamp manipulation is very high. This level of detection uses two or more factors to detect timestamp manipulation, and if detected by the detection method at this level, timestamp manipulation can be determined immediately without additional analysis. However, none of the previous

TABLE 3. Comparative analysis of NTFS timestamp manipulation detection methods.

Detection Methods	Detection Type	Detection Level	Detection Target Info	Anti-Forensic Resistance	Limitation
$MFT-1	Indirect	Suspicious	Which	High	Rules can be bypassed because an attacker can tamper with all eight timestamps. The most common method of changing all timestamps to the same value is not detectable by this method.
$MFT-2	Indirect	Additional	Which	High	The possibility of false positives exists because continuity violations can occur even in normal cases.
$MFT-3	Indirect	Suspicious	Which	High	After timestamp manipulation, access to the file or a reboot will synchronize the two timestamps. Timestamp manipulation via APIs cannot be detected.
$MFT-4	Indirect	Additional	Which	High	When using the timestamp-changing APIs or Powershell, the 100-nanosecond unit is set to zero if it is not set separately. This method has the potential for false positives, as it is not uncommon for a 100-nanosecond unit not to be set in this way, even in normal cases.
$LogFile-1	Direct	Suspicious	Which, When	High	There is a possibility of false positives because this method also detects when $SI-C is changed by file system tunneling. It cannot detect cases where another timestamp (e.g., $SI-M) is manipulated.
$LogFile-2	Indirect	Suspicious	Which	High	
$UsnJrnl-1	Direct	Suspicious	Which, When	High	This method cannot detect when a method that cannot manipulate $SI-E is used (SetFileTime(), Powershell).
Execution-1	Indirect	Additional	When	Low / Medium	This method cannot detect the execution of arbitrary timestamp manipulation tool and which file's timestamp is manipulated.
LNK-1	Indirect	Suspicious	Which	Low	This method cannot detect timestamp manipulation for files that do not create LNK files (e.g. executable files) when the file is opened. The possibility of false positives exists because the $SI-A of a file can be updated by an anti-virus product.
EventLog-1	Indirect	Additional	Which	Low	There is a possibility of false positives for files that are not created and modified by the user (e.g. system files, program files) and for files from outside.
VSC-1	Indirect	Suspicious	Which	Low	This method cannot detect cases where the file was created and the timestamp was manipulated before or after the volume shadow copy was created.

detection methods are at this level. Next, at the ''Suspicious'' level, timestamp manipulation is a possibility, but additional analysis is required for accurate determination. Therefore, if timestamp manipulation is detected by a detection method at this level, cross-analysis with other detection methods should be performed to determine timestamp manipulation. Finally, at the ''Additional'' level, it is difficult to determine timestamp manipulation by the corresponding method alone. Because the conditions or traces used by detection methods at this level can be confirmed even in normal cases and there is a high possibility of false detection, detection methods at this level should not be used alone, but should be used to assist Suspicious-level detection methods.

The detection target information is the information that the method targets for detection and can be classified into ''Which'' and ''When'' types. The ''Which'' type information is used to identify the file where timestamp manipulation occurred and is detected by most detection methods. The ''When'' type is time-related information, such as when a timestamp manipulation occurred or when a manipulation tool was run. This type of information is primarily detected by NTFS journal-based detection methods, which directly detect timestamp manipulation, and by program execution-based detection methods.

Anti-forensic resistance means resistance to anti-forensic operations, such as deletion, tampering, and initialization that an attacker might perform to erase traces of timestamp manipulation. Anti-forensic resistance is classified into ''Low'', ''Medium'', and ''High'' and depends on the type of artifacts used by each detection method. First, in the case of the ''Low'' level, an attacker can directly delete the file where the data are stored or delete the data with a simple command. Detection methods that use operating system artifacts such as prefetch, event log, and volume shadow copy fall into this level. At the ''Medium'' level, the attacker cannot directly delete the file where the data are stored, and to erase traces of the attack, the attacker must understand the structure of the file where the data are stored and delete individual data through APIs supported by the operating system. Detection methods at this level include the Execution-1 method, which uses registry artifacts. Finally, at the ''High'' level, an attacker cannot access the file where

the data are stored and delete the file through normal means, and the only approach is to manipulate the data through direct access to the disk. However, as mentioned earlier, Microsoft has prevented direct access to the system drive, and therefore a kernel vulnerability is required for this to occur. This level includes detection methods that use file system artifacts such as $MFT, $LogFile, and $UsnJrnl. In addition, all the artifacts mentioned, including file system artifacts, can be directly manipulated or destroyed by directly accessing the disk while the system is shut down, or by directly accessing the disk through a safe mode boot. However, given that most attacks are carried out against a running system in a remote location, this is realistically difficult for an attacker to do.

Looking at the limitations of each detection methods studied so far, all methods except the NTFS journal-based method have difficulty detecting the operation to change all timestamps ($SI-MAC or $SI-MACE) of created files to a specific past time immediately after file creation, which is most commonly used by malware. For $MFT-1, changing all timestamps equally to a specific past time is not detected by the detection rules in Table 2. MFT-2 is difficult to use because it often violates continuity rules even in normal cases, and $MFT-3 does not detect timestamp manipulations that use the timestamp-changing APIs most commonly used by malware. $MFT-4 is difficult to use because the 100-nanosecond unit is often zero, even in normal cases. $Execution-1 does not detect the execution of arbitrary malware that performs timestamp manipulation because it uses the name of a well-known timestamp manipulation tool as its signature. LNK-1 does not detect timestamp manipulation in executable files, and EventLog-1 is difficult to use because there are many files with $SI-C and $SI-M outside the user login period. Finally, VSC-1 does not detect manipulation of file timestamps immediately after files are created. On the other hand, $LogFile-1 and $LogFile-2 detect $SI-C manipulation, and therefore they can detect any change in $SI-MAC or $SI-MACE to a specific past time, and $UsnJrnl-1 detects $SI-E manipulation, so it can detect change in $SI-MACE to a specific past time.

Taken together, the results of the above comparative analysis showed that compared to other artifact-based detection methods, NTFS journal-based detection methods directly detected file timestamp manipulation and determine when the timestamp manipulation occurred, which can help investigators identify the most critical time of attack during a digital forensic investigation. And this method uses file system artifacts, making it relatively difficult for an attacker to delete or manipulate it compared to other artifacts. Finally, it detects manipulation of major timestamp, making it easy to detect the most common timestamp manipulation behaviors performed by malware. Therefore, NTFS journal-based detection methods can be judged to the most effective way to detect timestamp manipulation among the various timestamp manipulation detection methods in ditigal forensic investigations. However, previous NTFS journal-based detection methods have limitations such as detecting normal events as manipulation or detecting only limited cases, which makes it difficult to use in real-world environments. Therefore, in the next section, we propose a detection algorithm that improves the limitations of these previous NTFS journal-based detection methods.

Finally, all detection methods have their own limitations, making them difficult to apply to all variety of system environments and situations. Also, all artifacts used in the detection methods have the potential to be deleted and initialized by the attacker's anti-forensic behavior. Therefore, in order to effectively detect timestamp manipulation in various system environments and situations, and to detect sophisticated timestamp manipulation by attackers accompanied by anti-forensic behavior, it is necessary not to rely on a single detection method, but to use multiple detection methods to determine timestamp manipulation in an integrated manner.

V. ADVACNED DETECTION ALGORITHM BASED ON NTFS JOURNALS

As discussed in the previous section, the NTFS journal-based detection method is the most useful file timestamp manipulation detection method in digital forensic investigations because it can directly detect timestamp manipulation behavior and help determine the timing of the manipulation. In this section, we examine the limitations of previous NTFS journal-based detection methods and propose a NTFS journal-based detection algorithm that overcomes the limitations of previous detection methods.

A. LIMITATIONS OF PREVIOUS NTFS JOURNAL BASED DETECTION METHODS

Previous $LogFile-based detection methods ($LogFile-1, $LogFile-2) are all based on the assumption that the $SI-C of a file does not change after the file is created. However, in the real-world environment, $SI-C is frequently changed even after file creation by file system tunneling. File system tunneling is a function that maintains the $SI-C of a file in the file system. The detailed process is as follows. First, when a specific file A is deleted, renamed, or moved, the file name and the $SI-C of file A are cached. After this, within a specific time (default: 15 seconds), if a file B with the same name as the cached file name is created, or if file B is renamed or moved to the same name as the cached file name, file B's $SI-C is changed to the $SI-C of cached file A [31]. Such file system tunneling commonly occurs in programs' temporary file operations, and in this case, a record in which $SI-C is changed is created in $LogFile, resulting in many false positives in previous detection methods. Figure 1 summarizes the cases where file system tunneling occurs.

In addition, previous $LogFile-1 and $LogFile-2 detection methods only detect when $SI-C is manipulated, but do not detect when other timestamps are manipulated. For example, manipulation of $SI-M, a timestamp that attackers usually manipulate to hide their traces along with $SI-C, is not detected. Also, in the case of the previous

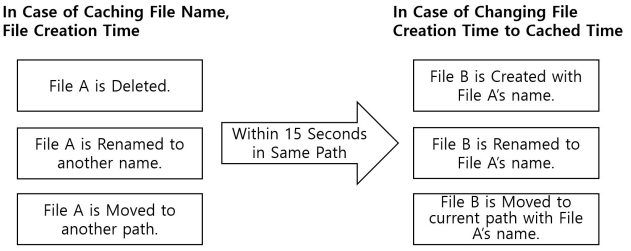

FIGURE 1. File system tunneling.

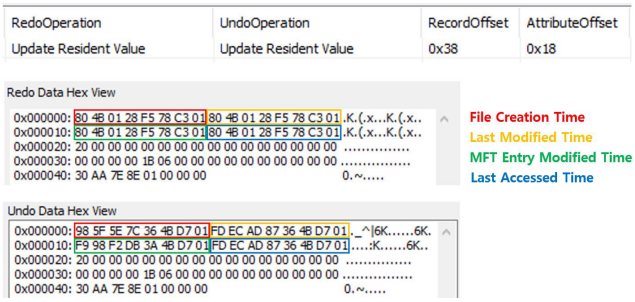

FIGURE 2. $LogFile record created by timestamp manipulation.

TimeStamp(UTC+9)	USN	File/Directory Name	Event
2021-06-05 22:18:08	7369874480	time_manipulation_test_1.txt	Basic_Info_Changed
2021-06-05 22:18:08	7369874600	time_manipulation_test_1.txt	Basic_Info_Changed , File_Closed

FIGURE 3. $UsnJrnl records created by timestamp manipulation.

$UsnJrnl-1 detection method, it cannot detect timestamp manipulation using the SetFileTime() API or Powershell, which cannot change the $SI-E. This is because these manipulation methods change the $SI-E of the target file to the time at which the manipulation is performed, so that the time of the last BASIC_INFO_CHANGE event for that file and the $SI-E have similar values.

B. TYPES OF TIMESTAMP MANIPULATION

There are two main types of file timestamp manipulation. The first type involves using an existing timestamp manipulation tool or the timestamp manipulation function of a backdoor to change the timestamp of a file that has already been created. In this case, there is a difference between the time when the timestamp-manipulated file is created and the time when the timestamp manipulation occurs. In the second type, an attacker creates malware that uses a time manipulation API or Powershell to modify the timestamp of a file. The malware produced in this way immediately manipulates the timestamp of the additional malware that it creates. In this case, there is little difference between the time when the timestamp-manipulated file is created and the time when the timestamp manipulation occurs.

C. TRACES OF TIMESTAMP MANIPULATION IN NTFS JOURNALS

As mentioned earlier, the currently available methods for manipulating file timestamps are SetFileTime(), NtSetInformationFile() API, and Get-Item cmdlet in PowerShell. The following traces are left in $LogFile and $UsnJrnl when timestamp manipulation is performed using this method.

In the case of $LogFile, a record whose Redo OP value of the record header is ''UpdateResidentValue'' (0x7) is created when timestamp manipulation is performed. The Redo OP value of the record means the type of operation performed on the $MFT entry, and the ''UpdateResidentValue'' means updating the attribute data in the $MFT entry. The generated record has a Record Offset value of 0x38, which means that the Record Offset value is the relative position of the attribute in the $MFT entry where the operation will be performed. Therefore, we know that the record will perform the operation on $SI located at 0x38 in the $MFT entry. In addition, the Attribute Offset value of the generated record has a value ranging from 0x18 to 0x30. Because the Attribute Offset value is the relative position of the data to be updated in the attribute to be modified, this means that the update will be performed for four timestamps located between +0x18 and +0x30 within $SI. The timestamp data used for the actual update is stored in Redo Data in the record, and the timestamp data before update is stored in Undo Data in the record. In other words, by analyzing the Redo Data and the Undo Data of the record, the original timestamp and the manipulated timestamp of the file can be accurately known. Figure 2 is an example of a $LogFile record generated when timestamp manipulation occurs.

In the case of $UsnJrnl, a record where the BASIC_INFO_CHANGE (0x8000) value is added to the Reason Flag, the event type, is created when timestamp manipulation is performed [26]. A record where the CLOSE (0x80000000) value is added to the Reason Flag is then created. BASIC_INFO_CHANGE is a value that is set when file attributes such as read-only, hidden, and timestamp are changed [45]. Figure 3 shows an example of $UsnJrnl records created when timestamp manipulation occurs.

D. ADVANCED DETECTION METHOD WITH $LOGFILE

This subsection describes a detection method that improves on the limitations of the previous $LogFile-based detection method. The overall detection process is as follows. The first step is to extract the timestamp change events from the $LogFile records, which consists of finding the timestamp change record and identifying the event target file. The second step is to check the timestamp change contents of the extracted timestamp change events, a nd if the timestamp change contents meet the detection condition, the events are determined as a detection target. The final step is to check whether the timestamp change event determined as the detection target was caused by file system tunneling, and when only $SI-C is changed, the event pattern of file system tunneling is checked. Figure 4 shows the overall process of the improved $LogFile-based detection method. A detailed description of each step is provided below.

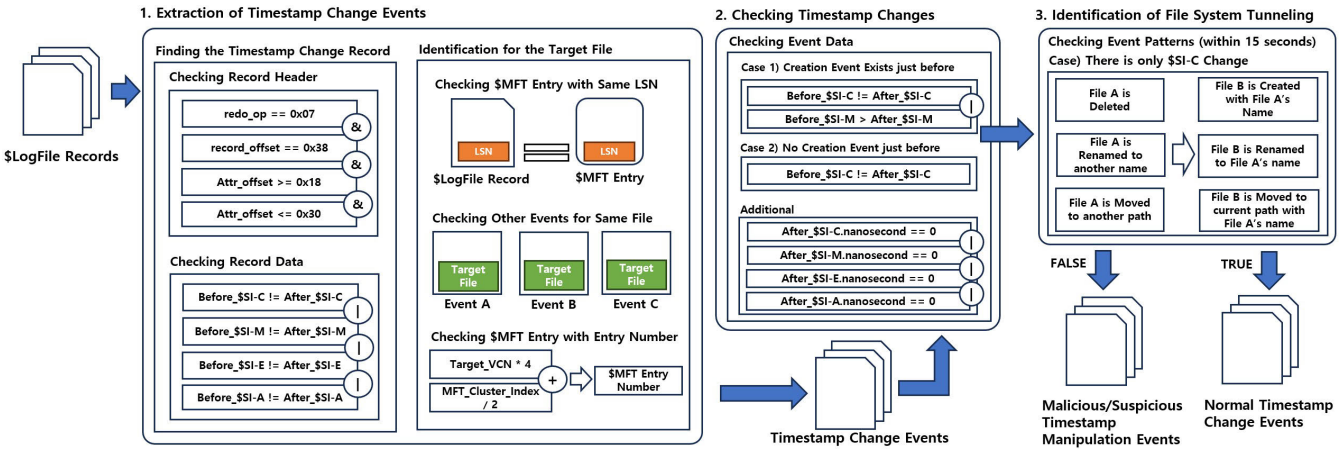

FIGURE 4. Overall procedure of the advanced $LogFile-based timestamp manipulation detection method.

1) EXTRACTION OF TIMESTAMP CHANGE EVENTS

The first task in detecting timestamp manipulation in $LogFile is to extract timestamp change events. However, previous $LogFile-based detection methods have not provided a way to extract timestamp change events [4]. Therefore, this paper proposes an algorithm for extracting timestamp change events.

To extract timestamp change events, the method must first find the record that performed the timestamp change. The timestamp change record can be found by checking whether the Redo OP (0x7), Record Offset (0x38), and Attribute Offset (0x18–0x30) values of the record header and the $SI timestamp stored in Redo Data and Undo Data of the record are different. If a timestamp change record is found, it is necessary to identify the target file of that record, but the timestamp change record does not contain information for identifying the target file in the record data. However, the following methods can identify the file targeted by the record. The first is to use the Log Sequence Number (LSN). The LSN is stored in the record header as information identifying each record of $LogFile, and it is also stored in the $MFT entry as the LSN of the last $LogFile record in which an operation was performed on that entry. Therefore, if there is an $MFT entry with the same LSN as the LSN of the timestamp change record, the file indicated by that entry can be determined as the target file of the record. The second is to use information from other events in $LogFile. This method searches for events with the same Target VCN and MFT Cluster Index as the timestamp change record for events that occurred before the timestamp change record. The Target VCN value means the in-volume address in the cluster where the $MFT entry that is the target of the operation for that record is located, and the MFT Cluter Index value indicates which entry in the cluster the preceding Target VCN value points to is the target $MFT entry [7]. In other words, if the Target VCN and MFT Cluster Index values of the two records are the same, the two records performed the operation for the same $MFT entry. Therefore, if a previous event with the same Target VCN value and MFT Cluster Index value is found, the target file of that event can be determined to be the target file of the record. The third approach is to calculate and use the $MFT entry number based on the Target VCN and MFT Cluster Index values. The $MFT entry number can be calculated by multiplying the Target VCN value by 4 and adding the MFT Cluster Index value divided by 2. If an active entry exists at the calculated entry number position in $MFT, the file pointed to by that entry can be determined as the target file of the record. However, this method has the potential for false positives when the file in which the timestamp change event occurred has been deleted.

Once the timestamp change record has been found and the target file identified, the timestamp change event can be extracted through the information of the found record and the record target file. The detailed process of extracting timestamp change events and identifying the record target file is represented in the pseudocode of Algorithm 1 and Algorithm 2.

2) CHECKING TIMESTAMP CHANGES

If the timestamp change events have been extracted, the next step is to check the timestamp change details of each event. In this paper, the chang es in $SI-C and $SI-M that attackers often make to hide their traces are checked. As for $SI-E and $SI-A, they are not checked because it is meaningless to change only these timestamps from the attacker's point of view.

In the case of $SI-C, any change in the corresponding timestamp is suspicious unless it was changed by file system tunneling. Therefore, in this paper, both cases where $SI-C is changed to the past and the future are determined as detection targets. In the case of $SI-M, on the other hand, it is normal for that timestamp to change to the future, so only if $SI-M changes to the past is it considered a detection target.

Change detection for $SI-C and $SI-M depends on whether the file was created immediately before the change event. If the file was not created immediately before, the

Algorithm 1 Extraction of Timestamp Change Events

```
for record in $LogFile_records[] do
    if record.redo_op == 0x7 then
        if record.record_offset == 0x38 then
            if record.attribute_offset >= 0x18
            && record.attribute_offset <= 0x30 then
                if record.attribute_offset == 0x18 then
                    memcpy(before_$SI-C, record.undo_data, 8)
                    memcpy(before_$SI-M, record.undo_data + 0x8, 8)
                    memcpy(before_$SI-E, record.undo_data + 0x10, 8)
                    memcpy(before_$SI-A, record.undo_data + 0x18, 8)
                    memcpy(after_$SI-C, record.redo_data, 8)
                    memcpy(after_$SI-M, record.redo_data + 0x8, 8)
                    memcpy(after_$SI-E, record.redo_data + 0x10, 8)
                    memcpy(after_$SI-A, record.redo_data + 0x18, 8)
                else if record.attribute_offset == 0x20 then
                    memcpy(before_$SI-M, record.undo_data, 8)
                    memcpy(before_$SI-E, record.undo_data + 0x8, 8)
                    memcpy(before_$SI-A, record.undo_data + 0x10, 8)
                    memcpy(after_$SI-M, record.redo_data, 8)
                    memcpy(after_$SI-E, record.redo_data + 0x8, 8)
                    memcpy(after_$SI-A, record.redo_data + 0x10, 8)
                else if record.attribute_offset == 0x28 then
                    memcpy(before_$SI-E, record.undo_data, 8)
                    memcpy(before_$SI-A, record.undo_data + 0x8, 8)
                    memcpy(after_$SI-E, record.redo_data, 8)
                    memcpy(after_$SI-A, record.redo_data + 0x8, 8)
                else if record.attribute_offset == 0x30 then
                    memcpy(before_$SI-A, record.undo_data, 8)
                    memcpy(after_$SI-A, record.redo_data, 8)
                end if
                if before$SI-C ! = after_$SI-C || before_$SI-M ! = after_$SI-M
                || before_$SI-E ! = after_$SI-E
                || before_$SI-A ! = after_$SI-A then
                    event.before_$SI-C ← before_$SI-C
                    event.after_$SI-C ← after_$SI-C
                    event.before_$SI-M ← before_$SI-M
                    event.after_$SI-M ← after_$SI-M
                    event.before_$SI-E ← before_$SI-E
                    event.after_$SI-E ← after_$SI-E
                    event.before_$SI-A ← before_$SI-A
                    event.after_$SI-A ← after_$SI-A
                    event.target_file ← GetTargetFile(record)
                    arr_timestamp_change_events[record] ← event
                end if
            end if
        end if
    end if
end for
```

method assumes that an existing manipulation tool or the manipulation function of a backdoor has been used to change the timestamp of a file that had already been created, and it checks whether $SI-C has changed or whether $SI-M has changed to the past. Conversely, if the file was created just before, the method assumes that the malware created another malware file and then immediately manipulated the timestamp of that file, and it checks whether $SI-C has changed. In this case, the reason for not checking whether $SI-M has been changed is that it is very common that a file is created and only $SI-M is changed immediately to the past in normal cases, and for malware to change only $SI-M without also changing $SI-C would be meaningless to bypass the timeline analysis.

In addition, if the 100-nanosecond unit of the changed $SI timestamp is zero, it is considered an additional detection factor. This is the case when the 100-nanosecond unit is not set, as confirmed in the $MFT-4 method. This additional detection factor is used after the overall timestamp manipulation detection process has completed to make a more accurate judgement of events that are determined to be timestamp manipulation events. Algorithm 3 is pseudocode for the process of checking for timestamp change content in a timestamp change event.

Algorithm 2 Identification of the Event Target File

```
function GetTargetFile(record)
    if There is a $MFT entry having LSN same as record.LSN then
        file.file_name ← $MFTEntry.file_name
        file.file_path ← $MFTEntry.file_path
        return file
    end if
    i ← 1
    while file_level_event_count-i >= 0 do
        previous_event ← arr_file_level_events[file_level_event_count-i]
        temp_target_vcn ← previous_event.target_vcn
        temp_mft_cluster_index ← previous_event.mft_cluster_index
        if temp_target_vcn == record.target_vcn
        && temp_mft_cluster_index == record.mft_cluster_index then
            if previous_event.event_type == "File Creation"
            || previous_event.event_type == "Renaming File"
            || previous_event.event_type == "File Move" then
                file.name ← previous_event.file_name
                file.path ← previous_event.file_path
                return file
            else if previous_event.event_type == "File Deletion" then
                break
            end if
        end if
    end while
    entry_number ← record.target_vcn * 4
    entry_number ← entry_number + (record.mft_cluster_index / 2)
    if There is a $MFT entry located at entry_number then
        if the $MFT entry is not inactive then
            file.name ← $MFTEntry.file_name
            file.path ← $MFTEntry.file_path
            return file
        end if
    end if
    return NULL
end function
```

Algorithm 3 Checking Timestamp Changes

```
for event in arr_timestamp_change_events[] do
    if there is no creation event just before then
        if event.before_$SI-C != event.after_$SI-C
        || event.before_$SI-M > event.after_$SI-M then
            detection_target ← TRUE
        end if
    else
        if event.before_$SI-C != event.after_$SI-C then
            detection_target ← TRUE
        end if
    end if
    if detection_target == TRUE then
        if event.after_$SI-C.100-nanosecond == 0
        || event.after_$SI-M.100-nanosecond == 0
        || event.after_$SI-E.100-nanosecond == 0
        || event.after_$SI-A.100-nanosecond == 0 then
            event.additional_detection_factor ← TRUE
        end if
        arr_detection_target_events[] ← event
    end if
end for
```

3) IDENTIFICATION OF FILE SYSTEM TUNNELING

If the timestamp change content is the detection target, it should finally be checked whether the event was caused by file system tunneling. In fact, if the $SI-C of a file is changed in a normal situation, most cases are caused by file system tunneling.

File system tunneling is performed when an existing file has been deleted, moved to a different path, or renamed, after which a file is created with an existing file name, renamed, or moved to an existing file name within 15 seconds [31]. Therefore, it is necessary to check whether the above pattern exists and whether changes to $SI-C have occurred by using file creation, deletion, file name change, or movement events that can be extracted through $LogFile analysis. One unusual feature here is that if file system tunneling occurs due to rapid creation, deletion, and re-naming by the program, a $LogFile record related to the timestamp change is created first, and then records related to file creation, deletion, and renaming can be generated. Logically, it seems that records related to file creation, deletion, and renaming that cause file system tunneling will be created, followed by a record related to timestamp change, which is the result of file system tunneling. However, in reality, a record related to timestamp change is often created first, after which records related to file creation, deletion, and renaming are frequently generated in any order. The detailed process of file system tunneling id-entification is represented in the pseudocode of Algorithm 4.

Algorithm 4 Identification of File System Tunneling for $LogFile

```
for event in arr_detection_target_events[] do
    if event.before_$SI-C != event.after_$SI-C
       && there are no other timestamp changes  then
        target_name ← event.target_file.name
        if file named target_name is deleted within the previous 15 seconds
        || file named target_name is renamed within the previous 15 seconds
        || file named target_name is moved within the previous 15 seconds then
            if after that, event.target_file is created with target_name
            ||  after that, event.target_file is renamed to target_name
            ||  after that, event.target_file is moved to target_name then
                this event is caused by file system tunneling
                continue
            end if
        end if
        if file named target_name is deleted within the after 1 second
        || file named target_name is renamed within the after 1 second
        || file named target_name is moved within the after 1 second then
            if event.target_file is created with target_name within the after 1 sec
            ||  event.target_file is renamed to target_name within the after 1 sec
            ||  event.target_file is moved to target_name within the after 1 sec then
                this event is caused by file system tunneling
                continue
            end if
        end if
        arr_timestamp_manipulation_events[] ← event
    else
        arr_timestamp_manipulation_events[] ← event
    end if
end for
```

If among the timestamp change events determined to be detection targets, only the $SI-C change occurred due to file system tunneling, the event is considered a normal event. Otherwise, it is considered a suspicious timestamp manipulation. In addition, among the events determined to be detection targets, the remaining events in which $SI-C has not changed are judged to be suspicious timestamp manipulation. Finally, if an event determined as suspicious timestamp manipulation has an additional detection factor (0 in 100-nanosecond unit), it is determined to be a timestamp manipulation.

The improved $LogFile-based detection method proposed in this paper is named $LogFile-1A, and is a Malicious/Suspicious-level detection method. If an event determined as suspicious timestamp manipulation has an additional detection factor, it is a Malicious-level detection and can be immediately determined to be a timestamp manipulation event. Otherwise, it is a Suspicious-level detection and requires cross-analysis with other methods for accurate determination.

E. ADVANCED DETECTION METHOD WITH $USNJRNL

This subsection describes a detection method that overcomes the limitations of the previous $UsnJrnl-based detection method. The overall detection process is as follows. The first step is to collect file information from the $UsnJrnl records, which consists of identifying the basic detection pattern and grouping the records of files having this pattern. The second step is to check the timestamp status of the collected file information. This step checks the state of the timestamp depending on the presence or absence of a file creation event, and if the condition is met, the event is determined as a detection target. The final step is to check whether the last basic detection pattern of the file determined as the detection target was caused by file system tunneling, and when there is an $SI-C change in the file, the file system tunneling pattern is checked. Figure 5 shows the overall process of the improved $UsnJrnl-based detection method. A detailed description of each step is provided below.

1) COLLECTING INFORMATION OF FILES WITH THE BASIC DETECTION PATTERN

The first task that must be performed to detect timestamp manipulation in $UsnJrnl is to collect information on files having the basic detection pattern. In a previous study, if the BASIC_INFO_CHANGE value was simply added to the Reason Flag of the record, the record was judged to be a BASIC_INFO_CHANGE event and used for detection [26]. However, as a result of executing various tools and malware performing timestamp manipulation in this paper, It was confirmed that timestamp manipulation creates a record added BASIC_INFO_CHANGE value in the Reason Flag and then an additional record added CLOSE value in the Reason Flag within a short time (0-1 second). This pattern means that the timestamp manipulation was performed just before the file was closed, and the pattern is used as the basic detection pattern in this paper. As mentioned earlier, BASIC_INFO_CHANGE events are also generated for changes to file properties other than timestamp changes, and therefore the potential for false positives exists when simply using BASIC_INFO_CHANGE events. Therefore, the basic detection patterns presented in this paper can be used to reduce these false positives and provide more sophisticated detection of timestamp manipulation.

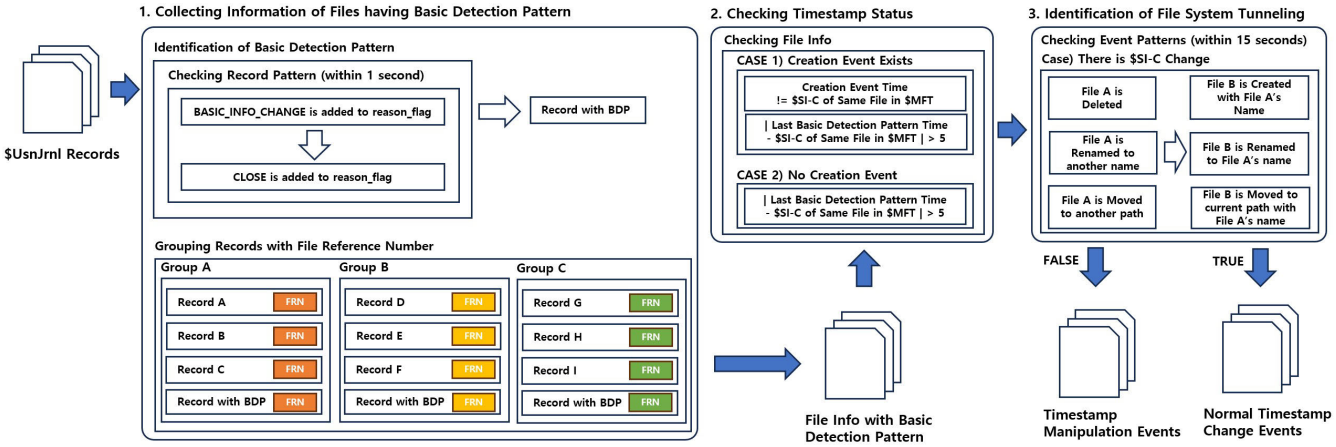

FIGURE 5. **Overall procedure of the advanced $UsnJrnl-based timestamp manipulation detection method.**

To collect information on files where the basic detection pattern occurs, it is first necessary to collect the BASIC_INFO_CHANGE records belonging to the basic detection pattern in the $UsnJrnl records. Then, using the File Reference Number value, which is information that identifies the record's target file, the method identifies and groups $UsnJrnl records that target the same file as the target file of each BASIC_INFO_CHANGE record collected. From such grouped records for the same file, file information such as file name, file path, file reference number, last basic detection pattern time, and creation event time is collected. Algorithm 5 contains pseudocode representing the process of collecting information on files in which the basic detection pattern has occurred.

Algorithm 5 Collecting Information of Files With the Basic Detection Pattern

```
for current_record in $UsnJrnl_records[] do
    if BASIC_INFO_CHANGE is added to current_record.reason_flag then
        if there is another record that CLOSE is added to reason_flag within
          the next second then
            arr_basic_detection_pattern[] ← current_record
        end if
    end if
end for
for record in arr_basic_detection_pattern[] do
    target_file_info ← NULL
    for file_info in arr_target_file_info[] do
        if record.file_reference_number == file_info.file_reference_number then
            find_flag ← TRUE
            target_file_info ← file_info
            break
        end if
    end for
    if find_flag == TRUE then
        target_file_info.last_basic_detection_pattern ← record
    else
        new_file_info.file_name ← record.file_name
        new_file_info.file_path ← record.file_path
        new_file_info.file_reference_number ← record.file_reference_number
        new_file_info.last_basic_detection_pattern_time ← record.timestamp
        new_file_info.create_event_time ← NULL
        target_file_reference_number ← record.file_reference_number
        for record2 in $UsnJrnl_records[] do
            if record2.file_reference_number == target_file_reference_number
            && FILE_CREATE is added to record2.reason_flag then
                new_file_info.create_event_time ← record2.timestamp
                break
            end if
        end for
        arr_target_file_info[] ← new_file_info
    end if
end for
```

2) CHECKING TIMESTAMP STATUS

Once information on files in which the basic detection pattern has occurred has been collected, the timestamp status of each file should be checked. As mentioned earlier, previous detection methods compare the time of the last BASIC_INFO_CHANGE event of the target file with the $SI-E of the same file in $MFT [26]. However, this method has the limitation that it cannot detect timestamp manipulation using the SetFileTime() API or Powershell, which cannot change $SI-E. In this paper, we propose a method to compare the creation event time of the target file with the $SI-C of the same file in $MFT to overcome this limitation of previous detection methods. In this method, the creation event time is the time when the target file was actually created, and the $SI-C of the same file in $MFT is the current $SI-C of the target file. If the two time values are not the same in seconds, it can be determined that $SI-C change of the target file has occurred.

Therefore, the detection method proposed in this paper checks the status of $SI-C and $SI-E. If there is a creation event for a file with a basic detection pattern, the status of both $SI-C and $SI-E are checked. The $SI-C status check compares the creation event time with the $SI-C of the same file in $MFT to determine if a $SI-C change has occurred. If there is an $SI-C change, the file is determined as a detection target. The $SI-E status check compares the time of the last basic detection pattern of the target file with the $SI-E of the same file in the $MFT for similarity. If the two time values are not similar, a change in $SI-E is assumed to have occurred, and the file is considered a detection target. The threshold for similarity is 5 seconds,

which means that two time values are similar if they are within 5 seconds of each other and not similar if they are more than 5 seconds apart. The reason for using 5 seconds as the threshold is that if the last time that a file attribute was changed, there is usually a 0–3 second difference between the time of the last BASIC_INFO_CHANGE event and the $SI-E of the same file in $MFT. Conversely, if there is no creation event, only the status of $SI-E is checked. Algorithm 6 contains pseudocode representing the process of checking the timestamp status of a file in which the basic detection pattern has occurred.

Algorithm 6 Checking Timestamp Status

```
for file_info in arr_target_file_info[] do
    $SI-C ← $SI-C of same file in $MFT
    $SI-E ← $SI-E of same file in $MFT
    record ← file_info.last_basic_detection_pattern
    last_basic_detection_pattern_time ← record.timestamp
    if file_info.create_event_time != NULL then
        if file_info.create_event_time != $SI-C then
            this file is a detection target
            arr_detection_target_files[] ← file_info
        else if | last_basic_detection_pattern_time - $SI-E | > 5 seconds then
            this file is a detection target
            arr_detection_target_files[] ← file_info
        end if
    else
        if | last_basic_detection_pattern_time - $SI-E | > 5 seconds then
            this file is a detection target
            arr_detection_target_files[] ← file_info
        end if
    end if
end for
```

3) IDENTIFICATION OF FILE SYSTEM TUNNELING

If a file with a basic detection pattern is determined to be a detection target, the next step is to determine whether the basic detection pattern in that file was caused by file system tunneling. As mentioned above, because $SI-C can be changed by file system tunneling, if a change to the $SI-C of a file occurs, it is necessary to check whether the basic detection pattern generated in the file was caused by file system tunneling, as in the case of $LogFile. The basic identification method for file system tunneling is the same as for $LogFile. Algorithm 7 contains pseudocode representing the process for identifying file system tunneling.

If among the files determined to be detection targets, it is confirmed that the last basic detection pattern of the file in which the $SI-C change occurred was caused by file system tunneling, the last basic detection pattern of the file is determined to be a normal event. Otherwise, it is considered a suspicious timestamp manipulation. In addition, among the files determined to be detection targets, the last basic detection pattern of the remaining files whose $SI-C change has not been confirmed is determined as a suspicous timestamp manipulation.

The improved detection method based on $UsnJrnl proposed in this paper is named $UsnJrnl-1A and is a Suspicious-level detection method. This method requires cross-analysis with other methods for accurate determination.

Algorithm 7 Identification of File System Tunneling for $UsnJrnl

```
for file_info in arr_detection_target_files[] do
    if file_info.create_event_time != NULL then
        record ← file_info.last_basic_detection_pattern
        target_name ← record.file_name
        if file named target_name is deleted within the previous 15 seconds
        || file named target_name is renamed within the previous 15 seconds
        || file named target_name is moved within the previous 15 seconds then
            if after that, record's target file is created with target_name
            || after that, record's target file is renamed to target_name
            || after that, record's target file is moved to target_name then
                this record is caused by file system tunneling
                continue
            end if
        end if
        if file named target_name is deleted within the after 1 second
        || file named target_name is renamed within the after 1 sec
        || file named target_name is moved within the after 1 sec then
            if record's target file is created with target_name within the after 1 sec
            || record's target file is renamed to target_name within the after 1 sec
            || record's target file is moved to target_name within the after 1 sec
            then
                this record is caused by file system tunneling
                continue
            end if
        end if
        arr_timestamp_manipulation_events[] ← record
    else
        arr_timestamp_manipulation_events[] ← record
    end if
end for
```

F. ADDITIONAL DETECTION METHODS

This subsection describes additional detection methods that can be used to perform cross-analysis with the two detection methods described above.

The first method is to compare whether a file has the same timestamp as a file in the same path or a specific path. This method detects the case where malware performs manipulation using the timestamp of a file in the same path or in a specific path [46]. For $LogFile, the $SI timestamp after the change in the timestamp change event is used for comparison, and for $UsnJrnl, the $SI timestamp of the file where the basic detection pattern occurred is used for comparison. If a file is found that is identical to the $SI timestamp being compared by up to 100-nanoseconds, it is judged that timestamp manipulation has occurred. Algorithm 8 and Algorithm 9 contain pseudocode that describes the process of identifying the same timestamp in $LogFile and $UsnJrnl. The detection methods are named $LogFile-3 and $UsnJrnl-2 respectively.

Algorithm 8 Detection for the Same Timestamp in $LogFile

```
for event in arr_timestamp_change_events[] do
    target_$SI-C ← event.after_$SI-C
    target_$SI-M ← event.after_$SI-M
    target_$SI-E ← event.after_$SI-E
    target_$SI-A ← event.after_$SI-A
    for file in files of the same path or a specific path do
        if  target_$SI-C == file.$SI-C
        || target_$SI-M == file.$SI-M
        || target_$SI-E == file.$SI-E
        || target_$SI-A == file.$SI-A then
            there is a file with same timestamp
            arr_timestamp_manipulation_events[] ← event
        end if
    end for
end for
```

Algorithm 9 Detection for the Same Timestamp in $UsnJrnl

```
for file_info in arr_target_file_info[] do
    target_$SI-C ← file.$SI-C
    target_$SI-M ← file.$SI-M
    target_$SI-E ← file.$SI-E
    target_$SI-A ← file.$SI-A
    record ← file_info.last_basic_detection_pattern
    for file in files of the same path or a specific path do
        if target_$SI-C == file.$SI-C
        || target_$SI-M == file.$SI-M
        || target_$SI-E == file.$SI-E
        || target_$SI-A == file.$SI-A then
            there is a file with same timestamp
            arr_timestamp_manipulation_events[] ← record
        end if
    end for
end for
```

The second method is to detect patterns of $FN timestamp manipulation. As described above, file movement within the same volume is essential to manipulate the $FN timestamp in currently available methods. There are two main cases of manipulating the $FN timestamp. The first case is when a timestamp change occurs for a specific file, after which the file is moved, and the timestamp change occurs again. In this case, the last timestamp change is an action to re-manipulate $SI-E, which has been changed due to file movement, and usually occurs when all timestamps of $SI and $FN are manipulated through the NtSetInformationFile() API. In the second case, there is no last timestamp change as in the first case, and only the MAC timestamps of $SI and $FN are changed using the SetFileTime() API or Powershell's Get-Item Cmdlet. In this case, because there is no intention to manipulate the E timestamp of each attribute, the method does not attempt to manipulate the changed $SI-E time again due to file movement. The detailed method of detecting the pattern of $FN timestamp manipulation in each $LogFile and $UsnJrnl is described in Algorithm 10 and Algorithm 11. The detection methods are named $LogFile-4 and $UsnJrnl-3 respectively.

Algorithm 10 Detection for Pattern of $FN Timestamp Manipulation in $LogFile

```
for event in arr_timestamp_change_events[] do
    if there is a move of event.target_file to another path after event then
        if after that, there is a timestamp change event of event.target_file then
            this is a pattern_1 of $FN timestamp manipulation
            arr_timestamp_manipulation_events[] ← event
        else
            this is a pattern_2 of $FN timestamp manipulation
            arr_timestamp_manipulation_events[] ← event
        end if
    end if
end for
```

The third method checks for traces of timestamp manipulation tool execution. This method takes advantage of the fact that program execution can be detected through creation and modification events of prefetch files in $LogFile and $UsnJrnl and uses the program name of a well-known timestamp manipulation tool as a signature to detect the execution of that program. The detailed method for detecting the execution of well-known timestamp manipulation tools in each $LogFile and $UsnJrnl is described in Algorithm 12 and Algorithm 13. The detection methods are named $LogFile-5 and $UsnJrnl-4 respectively.

Algorithm 11 Detection for Pattern of $FN Timestamp Manipulation in $UsnJrnl

```
for record in arr_basic_detection_pattern[] do
    if there is a move of record's target file to another path before record then
        if after that, there is a basic detection pattern of record's target file then
            this is a pattern_1 of $FN timestamp manipulation
            arr_timestamp_manipulation_events[] ← record
        else
            this is a pattern_2 of $FN timestamp manipulation
            arr_timestamp_manipulation_events[] ← record
        end if
    end if
end for
```

Algorithm 12 Detection for Execution of Timestamp Manipulation Tool in $LogFile

```
for event in arr_logfile_events[] do
    if event.target_file.name includes ".pf" then
        if event.event_info == "File Creation"
        || event.event_info == "File Modification" then
            for signature in arr_manipulation_tool_signatures[] do
                if event.file_name includes signature then
                    this event is the execution of timstamp manipulation tool
                    arr_timestamp_manipulation_tool_execution[] ← event
                end if
            end for
        end if
    end if
end for
```

Algorithm 13 Detection for Execution of Timestamp Manipulation Tool in $UsnJrnl

```
for record in arr_basic_detection_pattern[] do
    if record.file_name includes ".pf" then
        if "File_Created" is added to record.reason_flag
        || "Data_Truncated" is added to record.reason_flag then
            for signature in arr_manipulation_tool_signatures[] do
                if record.file_name includes signature then
                    this event is the execution of timstamp manipulation tool
                    arr_timestamp_manipulation_tool_execution[] ← event
                end if
            end for
        end if
    end if
end for
```

All the additional detection methods mentioned above are difficult to use alone as Additional-level detection methods, and cross-analysis with Suspicious-level detection methods should be performed.

G. ADVANCED DETECTION ALGORITHM

This subsection describes the $LogFile-based and $UsnJrnl-based detection algorithms, which integrate the detection methods proposed earlier. Table 4 summarizes the characteristics of each detection method proposed in this paper.

The $LogFile-based detection algorithm first takes $LogFile and $MFT as input and performs detection using the $LogFile-1A method. In the $LogFile-1A method, if an event determined as suspicious timestamp manipulation has an additional detection factor, the event is immediately determined to be timestamp manipulation without further

TABLE 4. Overview of proposed detection methods for timestamp manipulation.

Detection Methods	Detection Type	Detection Level	Detection Target Info	Anti-Forensic Resistance
$LogFile-1A	Direct	Malicious/Suspicious	Which, When	High
$UsnJrnl-1A	Direct	Suspicious	Which, When	High
$LogFile-3, $UsnJrnl-2	Direct	Additional	Which, When	High
$LogFile-4, $UsnJrnl-3	Direct	Additional	Which, When	High
$LogFile-5, $UsnJrnl-4	Indirect	Additional	When	High

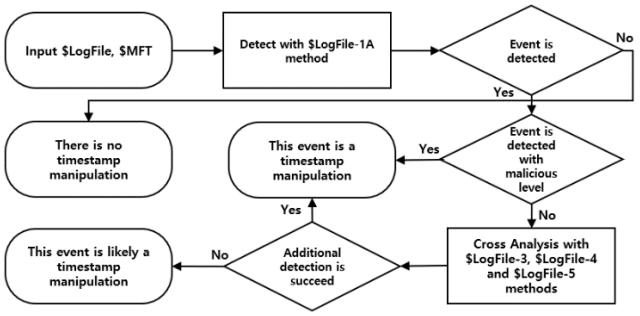

FIGURE 6. Flowchart of $LogFile-based detection algorithm.

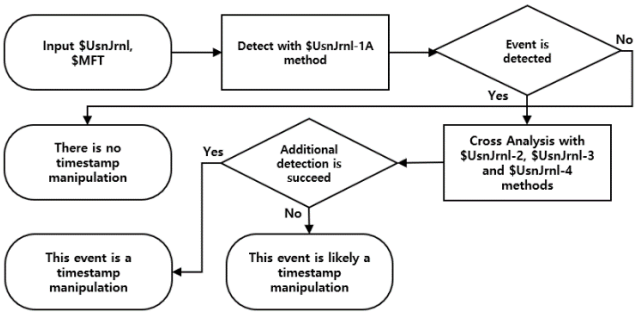

FIGURE 7. Flowchart of $UsnJrnl-based detection algorithm.

analysis because it is a Malicious-level detection. Otherwise, the event is cross-analyzed with the $LogFile-3, $LogFile-4, and $LogFile-5 methods for accurate determination because it is a Suspicious-level detection. For cross-analysis with the $LogFile-3 method, the post-change timestamp of the event detected in the $LogFile-1A method is used to check whether any of the files in the same path or a specific path (e.g., C:\Windows\system32\) have the same timestamp. When using cross-analysis with the $LogFile-4 method, it checks whether the patterns of $FN timestamp manipulation after the events detected in the $LogFile-1A method. Finally, when cross-analyzing with the $LogFile-5 method, it checks whether execution of a well-known timestamp manipulation tool just before the event detected in the $LogFile-1A method. This cross-analysis determines that the event is timestamp manipulation if additional detection is successful, and that the event is possibly timestamp manipulation if it is not. Figure 6 shows a flowchart of the $LogFile-based detection algorithm.

The $UsnJrnl-based detection algorithm also first takes $UsnJrnl and $MFT as input and performs detection using the $UsnJrnl-1A method. Unlike the $LogFile-1A method, the $UsnJrnl-1A method performs only Suspicious-level detection, and therefore the detected events are cross-analysed with the $UsnJrnl-3, $UsnJrnl-4, and $UsnJrnl-5 methods for accurate determination. The cross-analysis methods and the determinations made by each detection method are the same as in the $LogFile-based detection algorithm. Figure 7 shows a flowchart of the $UsnJrnl-based detection algorithm.

VI. EXPERIMENT

This section describes the performance evaluation between previous NTFS journal-based detection methods and the improved NTFS journal-based detection algorithm proposed in this paper.

A. TOOL DEVELOPMENT

1) NTFS LOG TRACKER v1.9

The detection algorithm proposed in this paper was added to the suspicious behavior detection function in NTFS Log Tracker v1.9. This tool performs suspicious behavior detection based on analyzed record data after receiving $LogFile, $UsnJrnl, and $MFT files as inputs. The timestamp manipulation detected by the detection algorithm proposed in this paper belongs to the ''Timestamp Manipulation'' category in the ''Detection Overview'' list, and its output is divided into ''Suspicious'' and ''Malicious'' levels in the ''Detection Detail'' list according to the degree of detection. The implemented tool can be downloaded from https://drive.google.com/drive/folders/1YHr35XVJTctiOjEsWY1W0pp_qXPsGZJR?usp=sharing. Figure 8 shows the result of performing suspicious behavior detection using NTFS Log Tracker v1.9.

2) DETECTION PROGRAMS BASED ON PREVIOUS STUDIES

Previous studies have proposed various timestamp manipulation detection methods based on NTFS journals [22], [25], [26]. However, Cho [25] and Palmbach and Breitinger [26] did not implement their tool, and Jang et al. [22] did not disclose the tool that they implemented. Therefore, we implemented all previous detection methods to evaluate their performance against the detection algorithm proposed in this paper. Table 5 shows a list of the implemented tools.

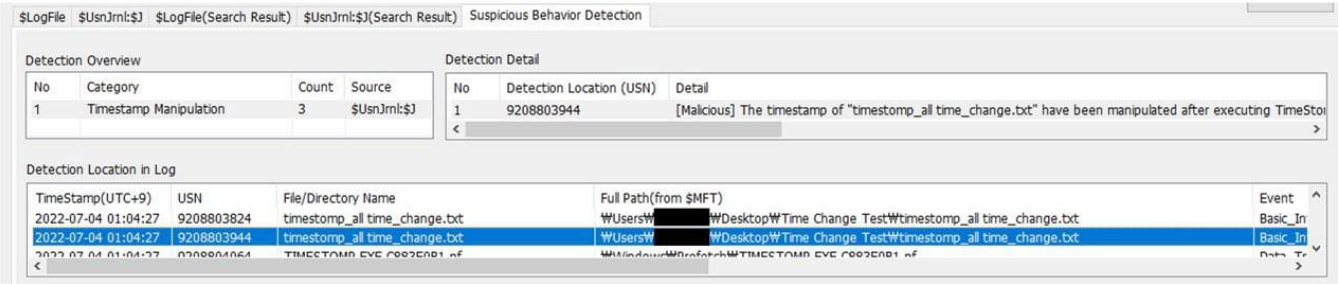

FIGURE 8. Result of timestamp manipulation detection in NTFS log tracker v1.9.

TABLE 5. Detection programs based on previous studies.

Name	Source	Detection Algorithm
Program A	$LogFile	Detects when $SI-C is changed [25]
Program B	$LogFile, $MFT	Detects when $SI-C of a file in $LogFile is different from $SI-C of the same file in $MFT [22]
Program C	$UsnJrnl, $MFT	Detects when the last BASIC_INFO_CHANGE event time of a file is different from $SI-E of the same file in $MFT [26]

Because no previous studies have disclosed the datasets used in their reported experiments, we verified whether the implemented tools had correctly executed the methods of detection proposed in past work based on a performance evaluation on the datasets considered in this paper. In the performance evaluation results, each tool detected all the timestamp manipulations that should have been detected by the previous detection methods, without missing any. Program A detected all cases of intentional manipulation of $SI-C, and Program B also detected all cases of intentional manipulation of $SI-C if the creation event of the target file remained in $LogFile. Program C also detected all cases where the NtSetInformati-onFile() API was used that resulted in a difference between the last BASIC_INFO_CHANGE event time and $SI-E.

B. EVALUATION

The performance evaluation was conducted using Programs A, B, C, and NTFS Log Tracker v1.9, which implement previous NTFS journal-based detection methods [22], [25], [26], and the NTFS journal-based detection algorithm proposed in this paper.

1) DATASETS

The dataset used for performance evaluation was generated by extracting the $LogFile, $UsnJrnl, and $MFT files of the system volume (C:) after performing the tasks corresponding to each test scenario in a Windows 10 Pro x64 environment on a VMware virtual machine. The dataset to which the test scenario was applied was divided into two parts. The first part was used to evaluate whether each detection tool detected well when $SI timestamp manipulation was performed through the three timestamp manipulation methods described above. $SI timestamp manipulation consisted of changing the $SI-C, the $SI-M, and the entire $SI timestamp, which are the items most commonly manipulated by attackers. For this purpose, NewFileTime v6.77 and nTimestomp v1.2, the most recent tools using each timestamp manipulation method, were used, and in addition, Windows Powershell was used to perform manipulation by means of the Get-Item Cmdlet. The second part was used to evaluate whether each detection tool detected file system tunneling and additional detection factors: timestamp zeroing within a 100-nanosecond unit, manipulation using file timestamps in the same path, and $FN timestamp manipulation. To do this, we performed the task to generate file system tunneling and used SetMACE v1.0.0.4, which supports manipulation using the file timestamps of another file and manipulation of $FN timestamps by moving files. In addition, it is possible to evaluate whether each detection tool detects the execution of a well-known timestamp manipulation tool through the tools used to generate the dataset above, and in particular, it is possible to evaluate whether each detection tool detects timestamp zeroing within a 100-nanosecond unit through NewFileTime, which does not allow for 100-nanosecond units to be modified. Table 6 details the tools and methods used to generate each dataset and the test scenarios applied. The dataset can be downloaded from the tool download URL.

2) EVALUATION RESULT

Table 7 shows the detection target information for each data set and the results of each detection tool. In the detection results, 'O' means that the detection target was detected, and 'X' means that it was not detected. A detailed description of the detection results is provided below.

For Program A and B, using the previous detection method based on $LogFile, they detected when $SI-C was manipulated, but not when only $SI-M was manipulated. In addition, they failed to identify file system tunneling, which caused a false positive, in which a normal event was detected as a timestamp manipulation event, and failed to detect any additional detection factors. On the contrary, NTFS Log Tracker v1.9, which uses the improved $LogFile-based

TABLE 6. Data sets applied by test scenarios.

No	Tools / Method	Test Scenarios
1	NewFileTime v6.77 (SetFileTime() API)	Manipulate $SI-C of NewFileTime_SI_C_Manipulation.dll to the past (0 in 100-nanosecond unit)
2		Manipulate $SI-M of NewFileTime_SI_M_Manipulation.dll to the past (0 in 100-nanosecond unit)
3		Manipulate $SI-MAC of NewFileTime_SI_MAC_Manipulation.dll to the past (0 in 100-nanosecond unit)
4	Windows PowerShell (Get-Item Cmdlet)	Manipulate $SI-C of PowerShell_SI_C_Manipulation.dll to the past
5		Manipulate $SI-M of PowerShell_SI_M_Manipulation.dll to the past
6		Manipulate $SI-MAC of PowerShell_SI_MAC_Manipulation.dll to the past
7	nTimestomp (NtSetInformationFile() API)	Manipulate $SI-C of nTimestomp_SI_C_Manipulation.dll to the past
8		Manipulate $SI-M of nTimestomp_SI_M_Manipulation.dll to the past
9		Manipulate $SI-MACE of nTimestomp_SI_MACE_Manipulation.dll to the past
10	File System Tunneling	Change $SI-C of FileSystemTunneling_SI_C_Change.dll by causing file system tunneling
11	SetMACE v1.0.0.4 (NtSetInformationFile() API)	Manipulate $SI-MACE of SetMACE_SI_MACE_Copy_Manipulation.dll using the $SI-MACE of another file in the same path
12		Manipulate $SI-MACE and $FN-MACE of SetMACE_SI_FN_MACE_Manipulation.dll to the past

detection algorithm proposed in this paper, not only detected $SI-C, $SI-M, and all $SI timestamps being manipulated, but also did not generate any false positives due to file system tunneling by identifying file system tunneling. In addition, it detected all additional detection factors that helped to accurately determine timestamp manipulation.

As for Program C, using the previous detection method based on $UsnJrnl, it did not detect any timestamp manipulations using the SetFileTime() API or Powershell, which cannot change $SI-E. Moreover, as in the cases of Programs A and B, each failed to identify file system tunneling and failed to detect any additional detection factors. On the other hand, NTFS Log Tracker v1.9, which uses the improved $UsnJrnl-based detection algorithm proposed in this paper, detected additionally $SI-C and $SI-E manipulation in even cases using the SetFileTime() API and Powershell, and did not generate any false positives due to file system tunneling by identifying file system tunneling. In addition, it detected all additional detection factors.

As described above, the NTFS journal-based detection algorithm proposed in this paper not only detects additional timestamp manipulation that is not detected by previous detection methods, but also identifies file system tunneling that causes false positives in previous detection methods. In addition, it can also detect additional factors required to accurately determine timestamp manipulation events, which will enable more efficient timestamp manipulation detection during digital forensic investigations compared to previous detection methods.

VII. CASE STUDY

In this section, we introduced an example of the application of previous detection methods and the detection algorithm proposed in this paper to detect malware that performs timestamp manipulation in real-world APT attacks.

A. APT MALWARE

The process for selecting the malware used for detection was as follows. First, malware files that perform timestamp modulation (T1070.006) [1] were collected from VirusTotal Collections [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58] for each attack group. The collected malware files were then executed, and behavior analysis was performed. Finally, malware files that performed the same pattern of timestamp manipulation behavior were grouped through behavior analysis, after which one representative malware file was selected from each group and used for detection. Table 8 summarizes the timestamp manipulation behavior of selected APT malware by the attack group used for detection.

B. DETECTION RESULT

To verify the detection of APT malware that performs timestamp manipulation, each malware was run in the dataset generation environment built in the previous section and $LogFile, $UsnJrnl, and $MFT files were collected to generate datasets. Detection was then performed on the generated datasets using each detection tool. The dataset can be downloaded from the tool download URL. Table 9 summarizes the timestamp manipulation behavior of the malware used for detection and the results of each detection tool. In the detection results, 'O' means that timestamp-manipulated files were detected, and 'X' means that they were not. In addition, "(S)" means that manipulation using the timestamp of another file was also detected, and "(F)" means that the $FN timestamp manipulation pattern was also detected. "(Z)" means that manipulation with timestamp zeroing within a 100-nanosecond unit was also detected.

Looking at the detection results of each detection tool, Program A and B, using the previous detection method based on $LogFile, detected all files with manipulated

TABLE 7. Performance evaluation results.

Datasets	Detection Target	Program A	Program B	NTFS Log Tracker v1.9 with $LogFile	Program C	NTFS Log Tracker v1.9 with $UsnJrnl
1	$SI-C manipulation with SetFileTime()	O	O	O	X	O
2	$SI-M manipulation with SetFileTime()	X	X	O	X	X
3	$SI-MAC manipulation with SetFileTime()	O	O	O	X	O
4	$SI-C manipulation with PowerShell	O	O	O	X	O
5	$SI-M manipulation with PowerShell	X	X	O	X	X
6	$SI-MAC manipulation with PowerShell	O	O	O	X	O
7	$SI-C manipulation with NtSetInformationFile()	O	O	O	O	O
8	$SI-M manipulation with NtSetInformationFile()	X	X	O	O	O
9	$SI-MACE manipulation with NtSetInformationFile()	O	O	O	O	O
10	File System Tunneling	X	X	O	X	O
1-3	Manipulation with 0 in 100-nanoseconds	X	X	O	X	O
11	Manipulation using another file's timestamp	X	X	O	X	O
12	$FN timestamp manipulation using file move	X	X	O	X	O
1-3, 7-9, 11-12	Execution of well-known timestamp manipulation tool	X	X	O	X	O

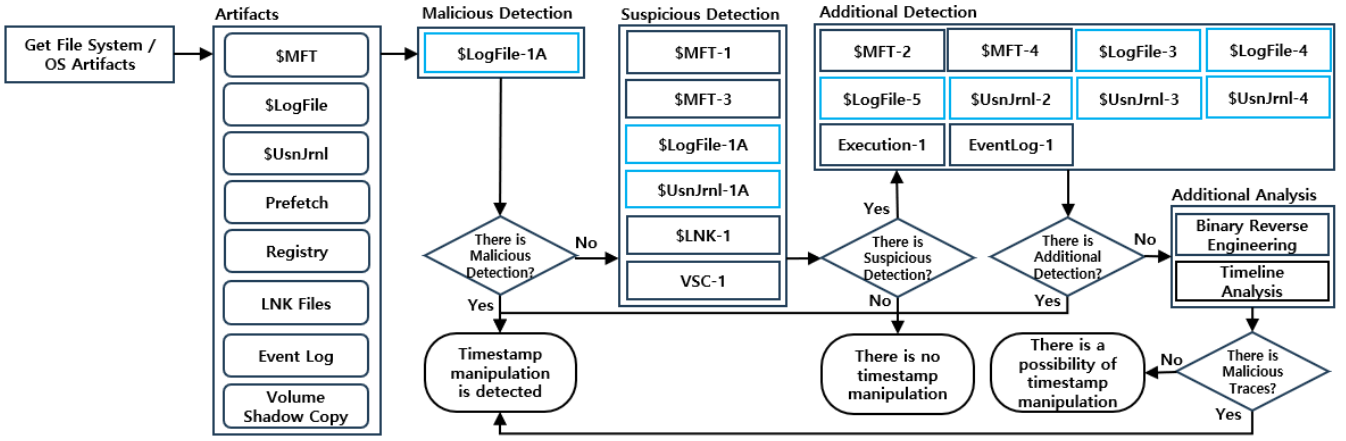

FIGURE 9. Process of timestamp manipulation decision in NTFS.

timestamps. This occurred because all malware used for detection manipulates $SI-C. On the other hand, NTFS Log Tracker v1.9, which uses the improved $LogFile-based detection algorithm proposed in this paper, not only detected all timestamp-manipulated files, but also detected both manipulations using the timestamps of other files, the $FN timestamp manipulation pattern, and timestamp zeroing within a 100-nanosecond unit. These additional detections can be used to determine that the timestamp of a file has been maliciously manipulated, without the need for cross-analysis with other artifacts.

In case of Program C, the previously proposed method of detection based on $UsnJrnl, none of the files with manipulated timestamps were detected. This was the case because all malware used for detection performed timestamp manipulation using the SetFileTime() API or PowerShell. Therefore, the previous $UsnJrnl-based detection method is considered difficult to use in real-world digital forensic investigations. On the other hand, NTFS Log Tracker v1.9, using the improved $UsnJrnl-based detection algorithm proposed in this paper, not only detected all files with manipulated timestamps, but also detected all three additional detection factors as in the case of $LogFile.

As mentioned above, the detection algorithm proposed in this paper can detect additional detection factors that previous detection methods cannot detect in $LogFile and can detect timestamp-manipulated files that previous detection methods cannot detect at all in $UsnJrnl. Therefore, the

TABLE 8. Behavior details of malware files from each attack group.

No	Group	SHA-1 Hash	Behavior Details of Timestamp Manipulation
1	APT17	87e37d09be4b9bf2624c75625fce9c60d3dd2b8a	File Creation (under \Windows\SysWOW64\) ➔ Writing File Data ➔ Manipulating $SI-MC using timestamp of svchost.exe in same path, Manipulating $SI-A to past
2	APT19	48e04cb52f1077b5f5aab75baff6c27b0ee4ade1	File Creation (under ~\AppData\Local\Temp\) ➔ Manipulating $SI-MC to past with the same value ➔ Writing File Data ➔ File move (to \Users\Public\Documents\) ➔ Writing File Data ➔ $SI-M Update ($SI-M changes back to the time before manipulation)
3	APT21	1b57eb997e7dd89ead0ba07c1df49d7596d9b4e4	File Creation (under ~\AppData\Local\Temp\) ➔ Manipulating $SI-MC to past with the same value ➔ Writing File Data
4	APT28	c23f18de9779c4f14a3655823f235f8e221d0f6a	File Creation (under \Program Files\Common Files\Microsoft shared\MSInfo\) ➔ Writing File Data ➔ Manipulating $SI-MC to past with the same value
5	APT29	2345cd5c112e55ba631dac539c8efab850c536b2	1) File Creation (under \Windows\SysWOW64\) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value (Performed on three files) 2) File Creation (under \Windows\SysWOW64\) ➔ Writing File Data ➔ Manipulating $SI-MAC to future with the same value (Performed on three files)
6	APT30	df48a7cd6c4a8f78f5847bad3776abc0458499a6	Directory Creation (\Program Files\Internet Exp1orer) ➔ File Creation (under \Program Files\Internet Exp1orer\) ➔ Writing File Data ➔ Manipulating $SI-MAC with the same value➔ Adding Hidden Attribute to File Attribute
7	APT37	e389470e5728c0f09158edbb7be4a45fc90232bc	File Creation (under ~\AppData\Roaming\Microsoft\Windows\) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value ➔ Writing File Data ➔ $SI-M Update ($SI-M changes back to the time before manipulation)
8	APT38	55daa1fca210ebf66b1a1d2db1aa3373b06da680	File Creation (under \Windows\SysWOW64\) ➔ Writing File Data ➔ Manipulating $SI-MAC using timestamp of runonce.exe in same path
9	APT40	244001126e4685eddc43b6d05415d6ddc92d0368	File Creation (under ~\AppData\Local\Temp\) ➔ Manipulating $SI-MC to past with the same value ➔ Writing File Data ➔ File move (to \Windows\)
10	Dark Hotel	6635e1ed92df4b225de32cc3ea3976eced2af159	1) Directory Creation (~\AppData\Roaming\visual) ➔ File Creation (under ~\AppData\Roaming\visual\) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value ➔ Writing File Data ➔ $SI-M Update ($SI-M changes back to the time before manipulation5) Manipulating $SI-MA to past with the same value 2) File Creation (under ~\AppData\Roaming\visual\) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value
11	Dark Hotel	bbd24fe828905b6e64981283b74fa0f0c9c06b2a	File Creation (under \Windows\SysWOW64\ or ~\AppData\Local\Temp\) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value
12	Kimsuky	e5358b0caa5ea21b38c4169ec7bfdd7202a4a005	File Creation (under \Windows\SysWOW64\) ➔ Writing File Data ➔ Manipulating $SI-MAC using timestamp of kernel32.dll in same path (Performed on three files)
13	Winnti	731335466523a958c16a512c3ebf244823d6b85d	File Creation (under ~\AppData\Local\Temp\) ➔ Writing File Data ➔Manipulating $SI-MC to past with the same value ➔ File move (to \Program Files (x86)\Common Files\Microsoft Shared\VGX\)
14	Winnti	53bfac12403c84993f959e511daec16d87b47161	File Creation (under \Windows\System32\ or \Windows\Help\ or \Windows\System32\wbem) ➔ Writing File Data ➔ Manipulating $SI-MAC to past with the same value

detection algorithm proposed in this paper is expected to help investigators efficiently find timestamp-manipulated files in real-world digital forensic investigations.

VIII. DISCUSSION

NTFS journal-based detection methods have limited detection range due to the default data capacity of NTFS journals. $LogFile has a base capacity of 64MB, and $UsnJrnl has a base capacity of 32MB. The data retention period of each journal varies depending on the volume of file operations on the system, but typically $LogFile has 2–3 hours of data, and $UsnJrnl has 30–40 hours of data [7]. Therefore, to overcome this limitation, it is necessary to increase the size of NTFS journal files from a forensic readiness perspective [59]. The maximum capacity of each NTFS journal file is 4GB, so if the storage device has sufficient capacity, setting both journal files to the maximum size will help prepare for digital forensic investigations.

The detection algorithm proposed in this paper prevents the normal event generated by file system tunneling by the operating system from being detected as a timestamp manipulation event. However, looking at the real-world environment, there are cases where timestamps are changed by legitimate programs. For example, there are cases where anti-virus and security products change the timestamp of a file, or where compression programs change the timestamp of

TABLE 9. Result of detecting timestamp manipulation by APT malware files.

No	$SI Timestamp Manipulation Details				$FN Timestamp Manipulation	Manipulation using another file's timestamp	Program A	Program B	NTFS Log Tracker v1.9 with $LogFile	Program C	NTFS Log Tracker v1.9 with $UsnJrnl
	M	A	C	E							
1	✓	✓	✓	×	×	✓	O	O	O(S)	X	O(S)
2	✓	×	✓	×	✓	×	O	O	O(F)(Z)	X	O(F)(Z)
3	✓	×	✓	×	×	×	O	O	O(Z)	X	O(Z)
4	✓	×	✓	×	×	×	O	O	O	X	O
5	✓	✓	✓	×	×	×	O	O	O	X	O
6	✓	✓	✓	×	×	×	O	O	O	X	O
7	✓	✓	✓	×	×	×	O	O	O(Z)	X	O(Z)
8	✓	✓	✓	×	×	✓	O	O	O(S)	X	O(S)
9	✓	×	✓	×	✓	×	O	O	O(F)(Z)	X	O(F)(Z)
10	✓	✓	✓	×	×	×	O	O	O	X	O
11	✓	✓	✓	×	×	×	O	O	O(Z)	X	O(Z)
12	✓	✓	✓	×	×	✓	O	O	O(S)	X	O(S)
13	✓	×	✓	×	✓	×	O	O	O(F)(Z)	X	O(F)(Z)
14	✓	✓	✓	×	×	✓	O	O	O(S)	X	O(S)

a decompressed file. Because the timestamp change methods used by these legitimate programs are in principle the same as those used by malware, the proposed detection algorithm may generate false positives. Therefore, accurate determination of file timestamp manipulation in NTFS should be performed comprehensively, including not only the NTFS journal-based detection methods proposed in this paper, but also the other detection methods mentioned in Section III and additional analyses (e.g., binary reverse engineering, timeline analysis). Figure 9 shows a decision-making process that integrates the detection methods proposed in this paper with previous detection methods and additional analysis methods to make a comprehensive determination of timestamp manipulation in NTFS. This process may help investigators determine file timestamp manipulation during digital forensic investigations.

IX. CONCLUSION AND FUTURE WORK

File systems are the primary structure that most operating systems use to store data, making them an important forensic target for investigators to analyze during digital forensic investigations. As key metadata in a filesystem, file timestamps are very important information that can be used to reconstruct file events in chronological order. For this reason, malicious users or attackers will attempt to manipulate timestamps to avoid their trace being detected. Therefore, detecting file timestamp manipulation can be very helpful in digital forensic investigations by detecting traces that malicious users or attackers try to hide.

In this paper, we studied the detection of file timestamp manipulation in NTFS, one of the most popular file systems in the world. We examined various existing artifact-based detection methods for detecting file timestamp manipulation in NTFS and compared them, finding that an NTFS journal-based detection method can most effectively detect file timestamp manipulation. However, existing NTFS journal-based detection methods have limitations that make them difficult to use in real-world digital forensic investigations. As such, we proposed a new detection algorithm to overcome these limitations. The proposed detection algorithm was evaluated against existing detection methods to confirm its improved performance and has been made publicly available as a tool. Finally, we applied existing detection methods and the proposed detection algorithm to a malware performed file timestamp manipulation in real-world APT attacks and verified the superiority of the proposed detection algorithm.

The results of this paper are expected to help investigators detect file timestamp manipulation during the real-world digital forensic investigation process and find files that malicious users or attackers want to hide.

As future work, we plan to study machine learning-based timestamp manipulation detection to reduce false positives that may occur with the detection algorithms proposed in this paper.

REFERENCES

[1] B. Carrier, *File System Analysis*. Reading, MA, USA: Addison-Wesley Professional, 2005.

[2] G.-S. Cho, "NTFS directory index analysis for computer forensics," presented at the *Proc. 9th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.*, Santa Cantarina, Brazil, Jul. 2015, pp. 441–446.

[3] K. Hansem and F. Toolan, "Decoding the APFS file system," *Digit. Invest.*, vol. 22, pp. 107–132, Sep. 2017.

[4] K. D. Fairbanks, "An analysis of Ext4 for digital forensics," *Digit. Invest.*, vol. 9, pp. 118–130, Aug. 2012.
[5] D. Kim, J. Park, K. Lee, and S. Lee, "Forensic analysis of Android phone using Ext4 file system journal log," *Future Inf. Technol., Appl., Service*, vol. 1, pp. 435–446, Jun. 2012.
[6] D. Cowen. (2013). *NTFS Triforce—A Deeper Look Inside the Artifacts*. Accessed: May 8, 2024. [Online]. Available: https://www.hecfblog.com/2013/01/ntfs-triforce-deeper-look-inside.html
[7] J. Oh. (2013). *NTFS Log Tracker*. Accessed: May 8, 2024. [Online]. Available: http://forensicinsight.org/wp-content/uploads/2013/06/F-INSIGHT-NTFS-Log-TrackerEnglish.pdf
[8] J. Oh. (2013). *Advanced $UsnJrnl Forensics*. Accessed: May 8, 2024. [Online]. Available: http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf
[9] J. Oh, S. Lee, and H. Hwang, "NTFS data tracker: Tracking file data history based on $LogFile," *Digit. Invest.*, vol. 39, Dec. 2021, Art. no. 301309.
[10] J. Schicht. (2014). *LogFileParser*. [Online]. Available: https://github.com/jschicht/LogFileParser
[11] X. Lin, "Deleted file recovery in NTFS," in *Introductory Computer Forensics*. New York, NY, USA: Springer, Nov. 2018, pp. 199–210.
[12] J. Plum and A. Dewald, "Forensic APFS file recovery," in *Proc. 13th Int. Conf. Availability, Rel. Security*, Hamburg, Germany, 2018, pp. 1–10.
[13] J. Oh. (2013). *Advanced $UsnJrnl Forensics*. Accessed: May 8, 2024. [Online]. Available: http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf
[14] M. Fuchs. (2018). *MFTEntryCarver*. Accessed: May 8, 2024. [Online]. Available: https://github.com/cyb3rfox/MFTEntryCarver
[15] H. Segev. (2021). *INDXRipper*. Accessed: May 8, 2024. [Online]. Available: https://github.com/harelsegev/INDXRipper
[16] LSoft Technologies. *APFS Recovery Methodologies*. Accessed: May 8, 2024. [Online]. Available: https://www.ntfs.com/apfs-recovery.htm
[17] A. Dewald and S. Seufert, "AFEIC: Advanced forensic Ext4 inode carving," *Digit. Invest.*, vol. 20, pp. S83–S91, Mar. 2017.
[18] *X-Ways Forensics*. Accessed: May 8, 2024. [Online]. Available: http://www.x-ways.net/winhex/manual.pdf
[19] S. Garfinkel. (2012). *BulkExtractor*. Accessed: May 8, 2024. [Online]. Available: https://github.com/simsong/bulk_extractor
[20] J. Oh, S. Lee, and H. Hwang, "Forensic recovery of file system metadata for digital forensic investigation," *IEEE Access*, vol. 10, pp. 111591–111606, 2022.
[21] X. Ding and H. Zou, "Reliable time based forensics in NTFS," in *Proc. Annu. Comput. Security Appl. Conf.* Shanghai, China: Shanghai Jiao Tong University, 2010, pp. 1–2.
[22] D. Jang, G. J. Ahn, H. Hwang, and K. Kim, "Understanding anti-forensic techniques with timestamp manipulation," in *Proc. 17th Int. Conf. Inf. Reuse Integr.*, Jul. 2016, pp. 609–614.
[23] S. Willassen, "Finding evidence of antedating in digital investigations," in *Proc. 3rd Int. Conf. Availability, Rel. Security*, Mar. 2008, pp. 26–32.
[24] W. Minnaard, "Timestomping NTFS," M.S. thesis, Dept. Math. Comput. Sci., Fac. Natural Sci., Univ. Amsterdam, Amsterdam, The Netherlands, 2014, pp. 6–10.
[25] G. Cho, "A computer forensic method for detecting timestamp forgery in NTFS," *Comput. Secur.*, vol. 34, pp. 36–46, May 2013.
[26] D. Palmbach and F. Breitinger, "Artifacts for detecting timestamp manipulation in NTFS on windows and their reliability," in *Proc. DFRWS*, 2020.
[27] A. Mohamed and C. Khalid, "Detection of suspicious timestamps in NTFS using volume shadow copies," *Int. J. Comput. Netw. Inf. Secur.*, vol. 12, no. 4, pp. 62–69, 2021.
[28] MITRE ATT&CK. *Indicator Removal: Timestomp*. Accessed: May 8, 2024. [Online]. Available: https://attack.mitre.org/techniques/T1070/006/
[29] B. Inglot, L. Liu, and N. Antonopoulos, "A framework for enhanced timeline analysis in digital forensics," in *Proc. IEEE Int. Conf. Green Comput. Commun.*, Nov. 2012, pp. 253–256.
[30] Mandiant. *M-Trends 2022 Executive Summary*. Accessed: 2023-05-08. [Online]. Available: https://www.mandiant.com/sites/default/files/
[31] (2019). *File System Tunneling in Windows*. Accessed: May 8, 2024. [Online]. Available: https://www.senturean.com/posts/19_04_13_windows-file-system-tunneling/
[32] (2021). *File Times*. Accessed: May 8, 2024. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/sysinfo/file-times
[33] *Windows Journal Parser*. Accessed: May 8, 2024. [Online]. Available: https:// tzworks.net/prototype_page.php?proto_id=5
[34] *SetFileTime*. Accessed: May 8, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/file
[35] *NewFileTime*. Accessed: May 8, 2024. [Online]. Available: https://www.softwareok.com/?Download=NewFileTime
[36] *SKTimestamp*. Accessed: May 8, 2024. [Online]. Available: https://tools.stefankueng.com/SKTimeStamp.html
[37] *BulkFileChanger*. Accessed: May 8, 2024. [Online]. Available: https://www.nirsoft.net/utils/bulk_file_changer.html
[38] *EXpress Timestamp Toucher*. Accessed: May 8, 2024. [Online]. Available: https://www.softpedia.com/get/PORTABLE-SOFTWARE/System/File-management/Portable-eXpress-TimeStamp-Toucher.shtml
[39] *Get-Item*. Accessed: May 8, 2024. [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.power
[40] M. Brinkmann. (2017). *How to Edit Timestamps With Windows PowerShell*. Accessed: May 8, 2024. [Online]. Available: https://www.ghacks.net/2017/10/09/how-to-edit-timestamps-with-windows-powershell/
[41] *NtSetInformationFile*. Accessed: May 8, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardwae/drivers/ddi/ntifs/nf-ntifs-ntsetinformationfile
[42] *Timestomp*. Accessed: May 8, 2024. [Online]. Available: https://forensicswiki.xyz/wiki/index.php?title=Timestomp
[43] *NTimestomp*. Accessed: May 8, 2024. [Online]. Available: https://github.com/limbenjamin/nTimetools
[44] *SetMACE*. Accessed: May 8, 2024. [Online]. Available: https://github.com/jschicht/SetMace
[45] *USN_RECORD_V2*. Accessed: May 8, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-usn_recor_v2
[46] (2020). *FASTCash 2.0: North Korea's BeagleBoyz Robbing Banks*. Accessed: May 8, 2024. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa20-239a
[47] *VirusTotal—Colloections APTClass: APT17*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/1dfebc4527c9f44a352f7d7bdf777c6c8bf0cf003ca412dba0da676f6f41b972
[48] *VirusTotal—Colloections Cyber-Research: APT19*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/006396e6f254e97d6c1b34cbc77abd7d98d13f28799df97b14c237f2afc9d634
[49] *VirusTotal—Colloections Cyber-Research: APT21*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/ecd35bdee8f6796352c38523e8354e6d4297f7bf8f7969f02219e26cf7d68ce6
[50] *VirusTotal—Colloections APTClass: APT28*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/106b003814966b09ee616eb126e9d9307e81424602974f8cd5696226c92ce090
[51] *VirusTotal—Colloections Cyber-Research: APT29*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/fb17c3f9ff70c654e85e9d7ff6947be399a8bac0d906ccbd8c180932df749d87
[52] *VirusTotal—Colloections Cyber-Research: APT30*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/11430e309f2ce4714c1ea9169cc37f656c768e5c1fb26c08b9ad5998907e353a
[53] *VirusTotal—Colloections APTClass: APT37*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/c0f40f6e3f69c7eff58afd7169bde529de167389cb558a40e8f2741f2daf99bd
[54] *VirusTotal—Colloections APTClass: Lazarus*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/bd5eac34d8dbb9513eaa125873539235e4b5e4864000df8d8f067df564bb79e4/ttps
[55] *VirusTotal—Colloections APTClass: APT40*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/ea8ff2e4ba849aede2db8b84c1cab1be3c652b32e9a35af486d375381fde7de6

[56] *VirusTotal—Colloections Cyber-Research: Dark Hotel*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/36908d14af6856bf598a0949ea839b35f8506585852274159b6784a307a69 896
[57] *VirusTotal—Colloections APTClass: Kimsuky*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/9054b6706c3d1f92cf25dd60be46d71e16a3dd99b7a5f4fddcb95f04449c27b5
[58] *VirusTotal—Colloections Cyber-Research: Winnti*. Accessed: May 8, 2024. [Online]. Available: https://www.virustotal.com/gui/collection/9152438f2d4efc7ce7d3672071abcef958f132997bd486d136efd185b9c478e7
[59] J. Tan. (2001). *Forensic Readiness*. Accessed: May 8, 2024. [Online]. Available: http://bit.ly/2D9rnAR
[60] J. Bouma, H. Jonker, V. van der Meer, and E. Van Den Aker, "Reconstructing timelines: From NTFS timestamps to file histories," in *Proc. 18th Int. Conf. Availability, Rel. Secur.*, 2023.

JUNGHOON OH received the B.S. degree from the Division of Computer, Information Communication Engineering, Dongguk University, in 2010. He is currently pursuing the Ph.D. degree with the Graduate School of Information Security, Korea University. His research interests include digital forensics, filesystem forensics, incident response, and artificial intelligence.

SANGJIN LEE received the Ph.D. degree from the Department of Mathematics, Korea University, in 1994. From 1989 to 1999, he was a Senior Researcher with the Electronics and Telecommunications Research Institute, South Korea. He has been running the Digital Forensic Research Center, Korea University, since 2008. He is currently the President of the Division of Information Security, Korea University. He has authored or coauthored more than 130 papers in various archival journals and conference proceedings and more than 200 articles in domestic journals. His research interests include digital forensics, data processing, forensic framework, and incident response.

HYUNUK HWANG (Member, IEEE) received the Ph.D. degree from the Department of Information Security, Chonnam National University, in 2004. He is currently the Head of Department of ETRI. His research interests include digital forensics, vulnerability verification, malware, and artificial intelligence.