International Journal of Internet, Broadcasting and Communication Vol.8 No.3 31-40 (2016)
http://dx.doi.org/10.7236/IJIBC.2016.8.3.31

IJIBC 16-3-4

Data Hiding in NTFS Timestamps for Anti-Forensics

Gyu-Sang Cho

Dept. of Computer Information Warfare, Dongyang University
cho@dyu.ac.kr

Abstract

In this paper, we propose a new anti-forensic method for hiding data in the timestamp of a file in the Windows NTFS filesystem. The main idea of the proposed method is to utilize the 16 least significant bits of the 64 bits in the timestamps. The 64-bit timestamp format represents a number of 100-nanosecond intervals, which are small enough to appear in less than a second, and are not commonly displayed with full precision in the Windows Explorer window or the file browsers of forensic tools. This allows them to be manipulated for other purposes. Every file has $STANDARD_INFORMATION and $FILE_NAME attributes, and each attribute has four timestamps respectively, so we can use 16 bytes to hide data. Without any changes in an original timestamp of "year-month-day hour:min:sec" format, we intentionally put manipulated data into the 16 least significant bits, making the existence of the hidden data in the timestamps difficult to uncover or detect. We demonstrated the applicability and feasibility of the proposed method with a test case.

Keywords: *Data hiding, Timestamp, NTFS filesystem, Anti-Forensics, Forensic tool*

1. Introduction

Data hiding techniques have recently become critically important, and a focus of concern in many application areas. Audio, video, and still images are increasingly embedded with distinguishing but inconspicuous marks that may contain hidden important information, or a serial code, with the goal of protecting the marked data from unauthorized access, tampering and direct copy [1].

Steganography and cryptographic codes have been widely used in warfare since ancient times. In many cases the success of secret missions, involving communication with agents abroad, communication between criminal and terrorist organizations, international espionage or APT attack, has depended on the ability to issues commands and communicate securely. The goals of secret communications have not changed in the past 30 centuries, but the methods and techniques of hiding data have continually evolved as new approaches have been developed [2].

Over the past decade the media involved in data hiding has steadily changed, from digital images to multimedia files, and has recently moved to mobile devices, as computing capabilities and network

Manuscript Received: Jun. 11, 2016 / Revised: Jun. 20, 2016 / Accepted: Jul. 12, 2016
Corresponding Author:cho@dyu.ac.kr
Tel:+82-54-630-1119, Fax: +82-54-630-1179
Dept. of Computer Information Warfare, Dongyang University

bandwidth have increased sufficiently. This evolution means that information leakage and covert communication can occur anywhere and anytime [2].

There are numerous ways of hiding data in a computer. For example, in Windows Systems, popular methods include the use of Alternate Data Streams (ADS), creating a file directory for an invisible hidden folder, encrypting files with encryption tools such as TrueCrypt and Windows BitLocker, using steganography tools such as Camouflage, OpenStego and QuckStego etc., using compressed files such as zip, 7-zip and rar etc., and taking advantage of physical features of the drive architecture, such as a reserved area of the disk called the Host Protected Area (HPA)[2].

A significant number of studies have investigated the art of data hiding by taking advantage of the structural characteristics of the filesystems of operating systems (OSs). K. Eckstein and M. Jahnke discussed variant techniques related to advanced file systems, and proposed a data hiding method which stored substantial data within the ext3 journaling file systems of Linux, with low detectability [3].

E. Huebner et al. surveyed the various data hiding methods in the Windows NTFS file system, and discussed analysis techniques which can be applied to detect and recover hidden data. Such methods are made possible by the structure of the NTFS file system. They include metadata files based methods (e.g., data hiding in a $BadClus file, data hiding in a $DATA attribute, and data hiding in a $Boot file), data files based methods (e.g., data files based methods, $DATA attribute in a directory, and data hiding in added clusters), and slack space based hiding methods (e.g., volume slack space, file slack space and file system slack space) [4].

Cho's work [5] described an anti-forensic technique for hiding data in an NTFS directory index which utilized B-tree behavior, and his subsequent research enhanced the functionality and the applicability of earlier work, which applied functions that used non-allowed characters, converted Hangul to Unicode, and binary data to extended Unicode [6] .

In this research, we propose a new anti-forensic method to hide data in the timestamps of a file in the Windows NTFS filesystem. The 64-bit timestamp format represents a number of 100-nanosecond intervals, which are short enough to appear in less than a second, and because they are not commonly displayed with full precision in the Windows Explorer window or the file browser of forensic tools, we can manipulate them for other purposes. Every file has $STANDARD_INFORMATION and $FILE_NAME attributes, and each attribute has four timestamps, respectively, so we can use 16 bytes to hide data. In Chapter 2, we describe the NTFS timestamp format and familiar timestamp changing tools, such as "Timestomp", "SetMace" and "File touch". In Chapter 3, the proposed method of hiding data in a timestamp is explained, and we show the results of data hiding 16 characters in a test file. In Chapter 4 we provide concluding remarks.

2. Timestamp ChangingTools

2.1 Timestamp format of NTFS

According to the Microsoft MSDN documentation [8] of the FILETIME data structure, an NTFS file timestamp has a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). The file times are recorded when applications create, access, and write to files. The FAT file system records the time stamp values based on the local time of the computer, but the NTFS file system records the timestamp values in the UTC format, so they are not affected by changes in time zone or daylight saving time [8].

Expressing one-second with 100-nanosecond granularity might seem superfluous, but two NTFS operations performed successively in a low-latency thread may require that different timestamps be produced by the system clock closely, one after the other, so a 100-nanosecond resolution would be necessary in order to represent distinct timestamps [10].

The $STANDARD_INFORMATION attribute stores the basic metadata of a file or directory. It has four time values, i.e., creation time, modified (write) time, MFT entry modified time and accessed time. The creation time stores the moment when the file is created, the modified time stores the moment that the file is updated, while the MFT entry modified time stores the moment of file metadata change, and the access time stores the moment the file is read [7, 9].

The $FILE_NAME attribute stores the file's name and parent directory information, and it may have multiple file name attributes to support an MS-DOS-based short file name. It has the same four $STANDARD_INFORMATION timestamps, but the attribute contains a different time value [7, 9].

2.2 Windows API for addressing file time

Windows API provides several file time-related functions to handle timestamps. The GetFileTime() function is used to retrieve the file times for a specified file. This function copies the creation, last access, and last write times to individual FILETIME structures. The SetFileTime() function is used to set the file times for a file. This function lets a user modify the creation, last access, and last write times without changing the content of the file. To compare the times of different files, the CompareFileTime() function is used. The function compares two file times and returns 0 if the times are equal. The SystemTimeToFileTime() function is used to convert a date and time of day to a file time, and to obtain the system time in a FILETIME structure by calling the GetSystemTimeAsFileTime() function. The FileTimeToSystemTime() function is used to make the file time easy to display to a user, and converts the file time and copies the month, day, year, and time of day from the file time to a SYSTEMTIME structure. The FILETIME value to be translated must be 0x7FFFFFFFFFFFFFFF or less. This corresponds to the time 30828-09-14 02:48:05.4775807.

If we try to change the timestamps of a file using the SetFileTime() function, we are only allowed to change three timestamps: creation time, modification time and access time. It does not provide access to change the MFT entry modification time, so, using forensic analysis tools, it can easily be determined whether the file has been manipulated by file time change tools [7]. However, fortunately, the Windows explorer window and the command prompt do not display the MFT entry modification time, so it is not easy to recognize that the timestamps have been changed.

Nevertheless, as mentioned above, it is not appropriate to employ a timestamp change tool using the SetFileTime() function for anti-forensic purposes. The timestamp change tools known to us, e.g., FileTouch.exe (http://www.softtreetech.com/), chtime.exe (https://github.com/Loadmaster/chtime-win32), and xtst.exe (http://www.irnis.net) are supposed to be employed using the appropriate function.

2.3 Timestamp manipulating tools: Timestomp

File system timestamps are not designed to be manipulated by the user, however, a powerful user can modify these timestamps using various tools. One of those methods involves using software applications that are designed to change timestamps.

Timestomp, which was made by James C. Foster and Vincent Lie, can delete or modify the timestamp. It contains a function that modifies the timestamp MFT Entry modification time. It cannot change $FN directly, however, it can be implemented using file move, by applying a series of commands, such as "timestomp.exe→file move command→timestomp.exe". This series of commands is based on the fact that the four timestamps of the $SI are copied to the four timestamps of the $FN after the file move command [7].

Another weak point of the timestomp.exe is that it cannot modify timestamps of less than a second, such as “c:\>timestomp.exe c:\test.txt -z "Saturday 10/08/2005 2:02:02 PM"”.

This program has options to select –m <data> (set the last written time), -a <data> (set the last accessed time), -c <data> (set the created time), -e <data> (set the MFT entry modified time), -z (set all four attributes), -f <src file> (set MACE of <filename> equal to MACE of <src file>), -r (the same as -b except it works recursively on a directory), -v (show the UTC MACE values for <filename>), and –h (show help)

2.4 Timestamp manipulating tools: SetMace

SetMACE is a more elaborate manipulating tool, influenced by timestomp.exe. The most recent version is Ver. 1.0.0.16 released in November, 2014, and is no longer dependent on the NtSetInformationFile() function, so it is completely based on resolving the filesystem internally and writing the timestamps directly to the physical disk, effectively bypassing the filesystem. Unlike timestomp.exe, it has the ability to directly modify the timestamps of the $FILE_NAME attribute.

This program has four arguments and several options to select. The first argument is designated to the input file. The second argument is used to change the timestamp of the target file with options, i.e., "-m" (modification time), "-a" (access time), "-c" (creation time), "-e" (MFT entry modification time), "-z" (all of the four timestamps change), and "-d" (dump existing timestamps given in UTC 0.00, including those in the INDX of the parent). The third argument designates the timestamp value to modify as “YYYY:MM:DD:HH:MM:SS:MSMSMS:NSNSNSNS”. The smallest possible value that can be set is "1601:01:01:00:00:00:000:0001", and the timestamps are written as UTC. The fourth argument determines if the $STANDARD_INFORMATION or the $FILE_NAME attribute or both should be modified. "-si" modifies timestamps in the four timestamps of $STANDARD_INFORMATION, "-fn" modifies timestamps in the four timestamps of the $FILE_NAME for short file names, eight timestamps for long names, and "-x" modifies timestamps in both the $FILE_NAME and $STANDARD_INFORMATION.

SetMace features a storing time of less than a second, it has 3 decimal digits for a milli-second (MS), and 4 decimal digits for a nano-second (NS). Therefore, 1/10,000,000 second (100 nano second) per bit can be displayed. Among timestamp manipulating tools, SetMACE is the only tool capable of setting elaborate time in less than a second.

Since NT 6.x (from Windows Vista and Windows Server 2008), Microsoft has banned direct write access to within the volume space, in order to avoid needing to implement a driver that can set the SL_FORCE_DIRECT_WRITE flag. This is a problem with 64-bit Windows, and Microsoft has implemented "PatchGuard", which will protect the kernel in memory, and prevent loading drivers without a certified signature. To circumvent the security feature, there are three possible options for properly using SetMace on a 64-bit nt6.x OS: 1) boot with TESTSIGNING configured and use a test signed driver; 2) crack PatchGuard (and thus there is no need to configure TESTSIGNING) and use a test signed driver; 3) find a way to use a properly signed driver, which is not possible now.

3. A New Method to Hide Data in Timestamps

In this chapter, we propose an original method to hide data in an NTFS timestamp, using a method to put a hexadecimal code in the two least significant bytes in the timestamps of the $STANDARD_INFORMATION and $FILE_NAME attributes.

3.1 Algorithm Used to Hide Data in Timestamps

1. Input an arbitrary length of data to hide. Calculate the character length (dl) and get the number of files used to hide the data. Allocate 16 bytes to lenFB (length per file block).

$$data2hide=input() \tag{1}$$
$$lenFB=16 \tag{2}$$

2. Calculate the number of files to handle with data2hide and lenFB.

$$noFile=ceiling(data2hide, lenFB) \tag{3}$$

, where the ceiling() function executes the calculation as INTEGER(data2hide/lenFB)+1.

3. Select the names of the files according to noFile.

$$fileName[i]=fileSelect(noFile) \tag{4}$$

4. If we save data in several files, the order of the selected files acts as a secret key saved to an array of key[noFile]. If we save data to a single file, there is no need to keep the order of the selected file. The key[noFile] must not be kept in the same place as the file with the hidden data.

$$key[i]=fileOrder(fileNmae[i]) \tag{5}$$

5. Applying the Windows API NtQueryInformation() function to the selected file, the four timestamps of the $SI and $FN are retrieved, respectively, in hexadecimal, where TS_{org_x} is a representative of the four original timestamps C, M, E, and A before they are manipulated.

$$TS_{org\_x}=getTimestampsHex(fileName[i]) \tag{6}$$

6. If each timestamp of TS_{org_x} is to do a bitwise AND mask with 0xFFFFFFFFFFFF0000, then the result of TS_{org_x} is as shown below.

$$TS_{org\_x} = T_7T_6T_5T_4T_3T_2 0_1 0_0 \tag{7}$$

7. Data to hide $data2hide_{\_x}$ in the original timestamp and TS_{org_x} is operated by an addition as follows.

$$TS_{hide\_x} = TS_{org\_x} + data2hide_{\_x} \tag{8}$$
$$TS_{hide\_x} = T_7T_6T_5T_4T_3T_2D_1D_0 \tag{9}$$

8. Call the FuncsetMace() function in arguments with the designated timestamp.

$$FuncsetMace(TS_{hide\_C}, TS_{hide\_W}, TS_{hide\_E}, TS_{hide\_A}) \tag{10}$$

9. Go to step3, and repeat if the process is not completed. Otherwise, go to end.

4. Application to a Test Case

4.1 Test Environments

We executed a test case of the proposed method by hiding data in timestamps in the Windows 7 operating system. The Development and Test environments are as follows.

Development environment

OS : Windows 7 Ultimate K Service Pack 1

Development Tool : Visual Studio 2012

Program Language : C/C++, MFC

Application Type : Windows dialogue program

Test environment

Disc Format : NTFS v3.1

Storage drive: Samsung SSD

Storage Space : 1TB

Disc Allocation Cluster Size : 4,096 bytes

Working Directory : c:₩timestompTest

SetMace Directory : c:₩SetMace

Test File : TimestampHideTest.txt

Data to Hide : "Hello hide world"

4.2 Application to a Test Case

We applied the proposed method to a test file named c:\timestompTest\TimestampHideTest.txt, and used a data string with 16 characters, "Hello hide world". The timestamps of the original file, and after hiding the data in the timestamp, are listed below. Figure 1 and Figure 4 show a screen shot of the $MFT file information of the TimestampHideTest.txt before and after applying the proposed data hiding method, respectively. After applying the proposed method, in Figure 4, the two least significant bytes of the timestamps of the $SI and $FN are changed, as 0x48→H, 0x65→e, 0x6C→l, 0x6C→l, 0x6F→o, 0x20→(space), 0x68→h, 0x69→i, 0x64→d, 0x65→e, 0x20→(space), 0x77→w, 0x6F→o, 0x72→r, 0x6C→l, 0x64→d.

Original timestamps of TimestampHideTest.txt:

$STANDARA_INFORMATION attribute

Creation : 01 D1 73 05 31 6D DC 79

Modification : 01 CA 11 A4 B2 78 08 00

MFT entry modification : 01 D1 C6 47 5A 46 76 94

Access : 01 D1 73 05 31 6D DC 79

$FILE_NAME attribute

Creation : 01 D1 73 05 31 6D DC 79

Modification : 01 CA 11 A4 B2 78 08 00

MFT entry modification : 01 D1 C6 47 5A 46 76 94

Access : 01 D1 73 05 31 6D DC 79

The shaded area below shows the hidden 16 bytes in the timestamp of the $SI and $FN, which is the same as that in Figure 4.

After data hiding in the timestamps of TimestampHideTest.txt

$STANDARA_INFORMATION attribute

- Creation : 01 D1 73 05 31 6D <u>65 48</u>
- Modification : 01 CA 11 A4 B2 78 <u>6C 6C</u>
- MFT entry modification : 01 D1 C6 47 5A 46 <u>20 6F</u>
- Access : 01 D1 73 05 31 6D <u>69 68</u>

$FILE_NAME attribute

- Creation : 01 D1 73 05 31 6D <u>65 64</u>
- Modification : 01 CA 11 A4 B2 78 <u>77 20</u>
- MFT entry modification : 01 D1 C6 47 5A 46 <u>72 6F</u>
- Access : 01 D1 73 05 31 6D <u>64 6C</u>

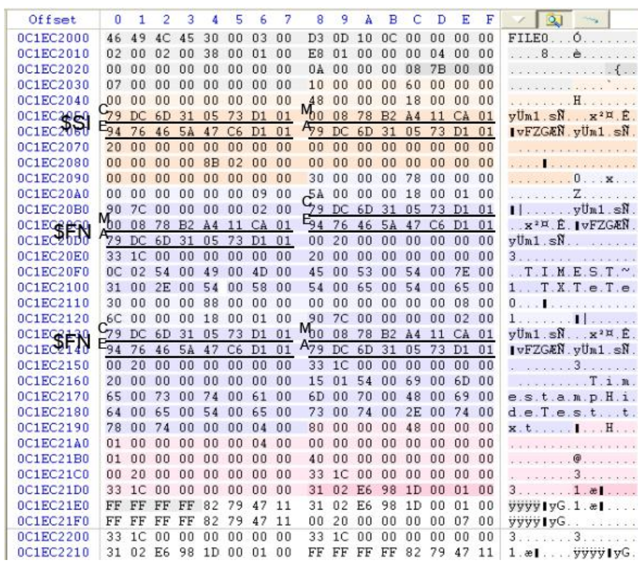

Figure 1. Screen shot of WinHex (original file)

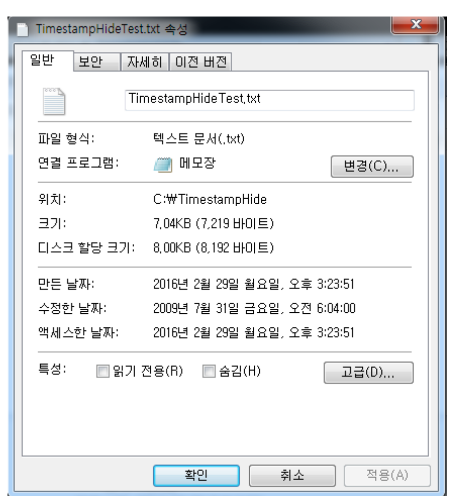

Figure 2. Screen shot of Property windows (original file)

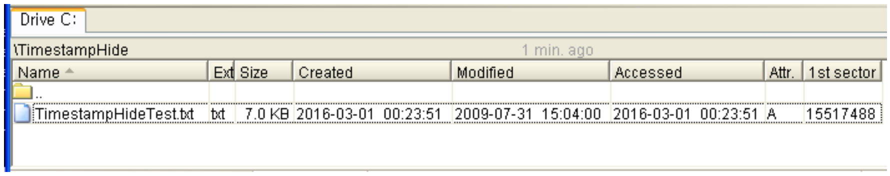

Figure 3. Screen shot of WinHex file browser (original file)

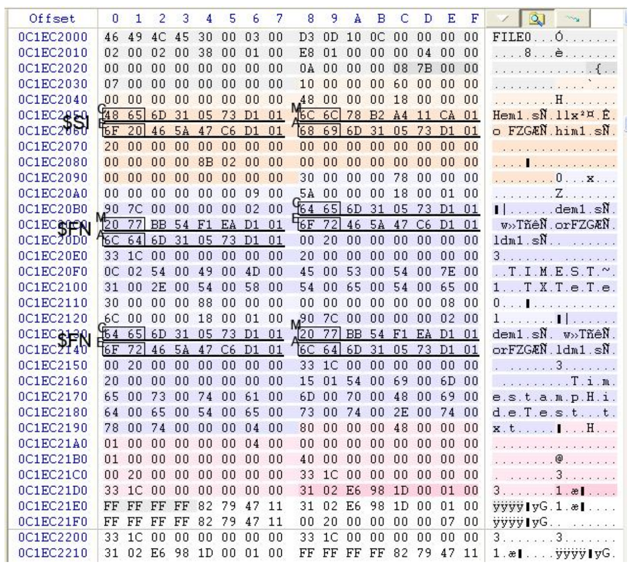

Figure 4. Screen shot of WinHex (data hidden file)

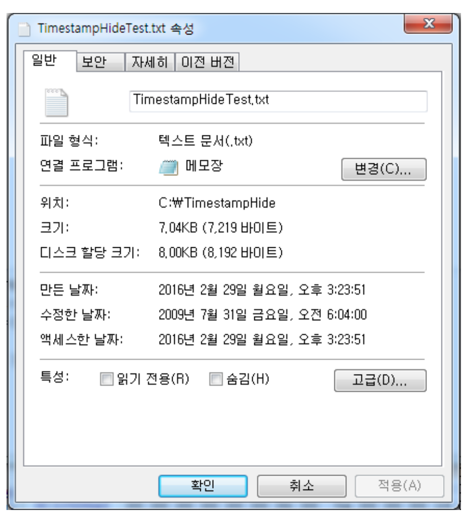

Figure 5. Screen shot of property window (data hidden file)

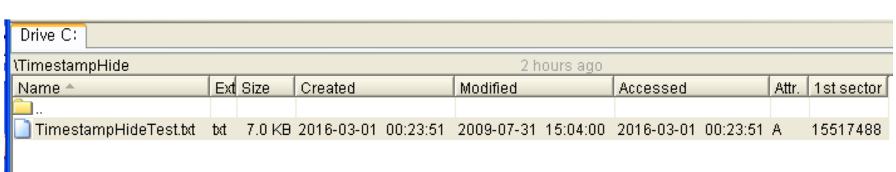

Figure 6. Screen shot of WinHex file browser (data hidden file)

Figure 2 and Figure 5 show screen shots of the Windows property window before and after applying the proposed data hiding method, respectively. Note that the two property windows are exactly the same in the two figures, where the timestamps of Creation time, “2016-2-29 3:23:51 PM”, Modification time, “2009-7-31 6:04:00 AM” and Access time, “2016-2-29 3:23:51 PM” are exactly the same. In Figure 3 and Figure 6, the two screen shots of the WinHex file list browser are also the same. We found that the proposed data hiding method works well.

5. Limitations and Future Works

This method proposes a new anti-forensic method of data hiding in timestamps. While examiners using forensic tools can see all of the timestamps, it is difficult to uncover or detect the existence of hidden data in the timestamps. Therefore, a method which hides data in that space is a good approach.

There are, however, a few limitations in this method. Firstly, the size of the data that can be hidden is small: only 16 bytes can be stored in each file. This small space is appropriate for small sized data such as a secret cryptography key. Secondly, when there are multiple users in one computer, a file with hidden data could be deleted accidently. To avoid this circumstance, it is recommended that the system file, software program files and library files be used to hide data.

It is always necessary to improve the order of the selected file which is to be used as a part of a key. It is important to develop ways of logically finding the relationship between easy-to-remember-words or numeric keys and the order of selected files. The tool developed in this research has minimum functionality, and the SetMace program is used in Step 8 in combination with the proposed method. For a practical anti-forensic tool, the tool should be improved for completeness. It is designed to work in environments of NT 6.x or later, and 64 bit Windows.

6. Conclusions

In this paper, we proposed a new anti-forensic method for hiding data in the timestamps of a file in the Windows NTFS filesystem. To the best knowledge of the author of this research, this is the first work to hide data in the timestamps of a file in the NTFS filesystem. The main idea of the proposed method is to utilize the 16 least significant bits of the 64 bits in the timestamps of the $STANDARD_INFORMATION and the $FILE_NAME attributes, respectively. The 64-bit timestamp format has 100-nanosecond precision, which is sufficiently small to be expressed in less than a second, and is not displayed below a second in either the Windows Explorer window or the file browsers of forensic tools. We demonstrated the performance of the proposed method by applying it to a sample file. Without any changes in an original timestamp of the “year-month-day hour:min:sec” format, we were able to insert manipulated data into the 16 least significant bits. We found that afterward it was difficult to uncover or detect the existence of the hidden data in the timestamps.

To create a sophisticated anti-forensic tool using the proposed method, additional elaborate work is needed, i.e., the tool needs more GUI interface features to make it easier to use, to access system files, added cryptography to prevent forensic examiners from intuitively or accidentally finding the hidden data, and development for application to other recent filesystems, such as FAT32, Ext3,4, and 5, HFS+, and yaffs etc.

Acknowledgements

This study was supported by grant from Dong Yang University in 2015.

References

[1] F. A. P. Petitcolas, R. J. Anderson, AND M. G. Kuhn, “Information Hiding—A Survey,” Proceedings of the IEEE, Vol. 87, No. 7, pp. 1062-1078, July 1999.

[2] Michael T. Raggo and Chet Hosmer, *Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems, Mobile Devices and Network Protocols*, Elsevier, 2013.

[3] K. Eckstein and M. Jahnke, “Data Hiding in Journaling File Systems”, Proceedings of Digital Forensic Research Workshop (DFRWS), pp. 1-8, 2005.

[4] Ewa Huebner, Derek Bem and Cheong Kai Wee, “Data Hiding in the NTFS File System,” Digital Investigation, Vol. 3, Issue 4, 2006, pp. 211-226.

[5] G.-S. Cho, “A New NTFS Anti-Forensic Technique for NTFS Index Entry,” The Journal of Korea Institute of Information, Electronics, and Communication Technology (ISSN 2005-081X), vol. 8, no. 4, 2015.

[6] G.-S. Cho, “An Anti-Forensic Technique for Hiding Data in NTFS Index Record with a Unicode Transformation,” Journal of Korea Convergence Security Association, Vol. 16, No. 7, pp. 75-84, July 2015.

[7] G.-S. Cho, “A Computer Forensic Method for Detecting Timestamp Forgery in NTFS”, Computer & Security, Vol. 34, 2013, pp. 36-46.

[8] Microsoft MSDN, File Times, *https://msdn.microsoft.com/en-us/library/windows/desktop/ms724290(v=vs.85).aspx*

[9] B. Carrier, File System Forensic Analysis, Addison-Wesley, pp. 273-396, 2005

[10] Wicher Minnaard, "Timestomping NTFS," IMSc final research project report, University of Amsterdam, Faculty of Natural Sciences, Mathematics and Computer Science, 2014.