International Journal of Internet, Broadcasting and Communication Vol.9 No.3 17-26 (2017)
https://doi.org/10.7236/IJIBC.2017.9.3.17

IJIBC 17-3-3

A Maximum Data Allocation Rule for an Anti-forensic Data Hiding Method in NTFS Index Record

Gyu-Sang Cho

School of Public Technology Service, Dongyang University
cho@dyu.ac.kr

Abstract

An anti-forensic data hiding method in an NTFS index record is a method designed for anti-forensics, which records data as a file name in index entries and thereafter the index entries are made to remain in the intentionally generated slack area in a 4KB-sized index record[7]. In this paper, we propose a maximum data allocation rule for an anti-forensic data hiding method in an NTFS index record; i.e., a computational method for storing optimal data to hide data in an index record of NTFS is developed and the optimal solution is obtained by applying the method. We confirm that the result of analyzing the case where the number of index entries n = 7 is the maximum case, and show the screen captures of index entries as experimental results.

Keywords: *NTFS filesystem, Index Record Data Hiding, Anti-forensic*

1. Introduction

Data hiding is the process of making data difficult to find while allowing access and use with limitations on identification and collection of evidence by investigators. Obfuscation and encryption of data are the representative techniques for achieving this purpose[1]. Using a combination of data hiding methods, e.g. encryption, steganography and other various forms of data concealment, makes digital forensic examinations more difficult.

Raggo and Hosmer[2] introduce various types of media for data hiding methods. They also describe data hiding methods in the operating system, mainly for Windows and Linux operating systems, such as ADS (alternate data stream), stealth ADS, and volume shadowing methods for Windows and filename trickery, extended file system data hiding, and TrueCrypt for Linux. Carvey[3] also describes data hiding methods in the operating system, such as ADS, registry, office documents, and OLE structured storage.

Hiding data throughout various locations of a computer system, e.g., slack space, bad sectors, alternate data streams, hidden partitions of disk and hidden directories, involve the use of various tools and techniques[1]. Slacker is part of the Metasploit framework, which is one of the well-known open source tools used for data hiding. Slacker allows you to hide data in the slack space of NTFS, which is created when a file system allocates more space for a file to be written than it actually uses, and it gives an ideal data-hiding ground for the hacker[4].

Huebner et al. describe a method to hide data in the NTFS file system, and discuss detection and recovery

Manuscript Received: May. 20, 2017 / Revised: June. 3, 2017 / Accepted: June. 10, 2017
*Corresponding Author: cho@dyu.ac.kr
Tel:+82-31-839-9066, Fax: +82-31-839-9066
School of Public Technology Services, Dongyang University, Korea

analysis techniques of hidden data. They focus on sophisticated data hiding methods to prevent detection by a forensic analysis and the methods are made possible by the structure of the NTFS file system. The methods include metadata file based methods, data file based methods, and slack space hiding methods in NTFS[5].

K. Eckstein and M. Jahnke propose a data hiding method [6] related to advanced file systems, where data is stored to be hidden within the ext3 journaling file systems of Linux, with a low possibility of detectability. The method exploits the fact that journaling file systems audit only recent modifications recorded in the journal, which allows the intentional insertion of user data into file system data structures.

Recently, Cho[7] proposed a new data hiding method to hide in the NTFS index record, which writes data as a file name in index entries in an index record, and these are intentionally made to remain a slack area in a 4KB-sized index record. Applying previous work, he announced a technique of data hiding in the NTFS index record with a Unicode transformation that converts Hangul to Unicode and binary data to extended Unicode conversion, for non-allowed character usage[8].

In this paper, we propose a maximum data allocation rule for an anti-forensic data hiding method in a NTFS index record. This problem is a function of the length of the character and the number of index entries, with some difficult conditions, as described in Chapter 2. In Chap. 3, an anti-forensic data hiding method in a NTFS index record is introduced. This method is a previously published approach[7], but only the basic method is presented in the algorithm. Therefore, a more specific algorithm is required for a fully equipped anti-forensic tool. A computational method for storing optimal data to hide data in an index record of NTFS is proposed and the optimal solution is obtained by applying the method. The function for the problem is presented in Section 4, and the optimum solution is obtained by using the method. In Section 5, we show the result of analyzing the case where the number of index entries is n = 7 with screen captures of index entries. In Chapter 6, we present conclusions of this study and discuss future research.

2. Problems

The logical method for allocating maximum data gives a logical way to determine the maximum number of characters that record as file names in the index entry. In order to obtain the logical solution for the method, there are many factors that make it difficult to solve the problem, and thus effects of the factors should be considered deliberately. The factors are listed below.

- **Limitation on file name length.** The file name uses 2-bytes Unicode characters. From 1 to 255 characters can be used.

- **8-bytes unit for file name allocation.** The file name is recorded in units of 8 bytes. Even if one character is recorded, 8 bytes are allocated, and if four characters are used, 8 bytes are still allocated for the file name. In this case, there is a buffer zone when selecting the optimal character.

- **Path length included in a file name.** When calculating the file name length, the path length is included. However, when actually writing a file name, the path is not included. In other words, when considering the length of a file name, only up to 255 characters are allowed, including the path, but only the portion of the file name excluding the path is saved actually. Sometimes it is difficult to find the reason why the file name length cannot store up to 255 characters if the path length is overlooked.

- **The longest file does not guarantee the maximum.** Choosing the longest file name does not guarantee that the most optimal storage of the data will be secured. Due to the relationship between the fixed size of the index record and the storage format of the index entry, it is necessary to analyze the correlation of the optimal storage length considering space efficiency.

- **End of an index entry should be considered.** At the end of the index entry, the 16-byte last index entry is used as a marker to indicate the end of the entire list. It should be considered that this may not be taken into account when calculating the entire character of an index record.

- **2 bytes for file name header.** Two bytes in the header of the file name string are used to indicate the information of the file name. The first byte represents the length of the entire filename and the second byte is used to represent the file namespace.

3. Algorithm of data hiding method and data structure

3.1 Data hiding method in an index record of NTFS

In this section, we introduce an anti-forensic data hiding method in an NTFS index record. This method is a previously published approach[7], and we simply explain the main algorithm as follows.

(1) **Initialization and data input**

- Enter the name of the working directory d_{work}.
- Enter the message m_{hide} to hide
- Set the block size b of the length message m_{hide}
- Initialize all variables

(2) **Divide messages to hide by n**

- Divide the message to hide into blocks of a certain size.
- Insert a bogus character to prevent the empty part of the end of the last message block from becoming a multiple of n.

(3) **Insert head number h_{pre} for block ordering**

- Index entries are automatically sorted by alphabetical order. When the message m_{hide} is divided into several blocks, it is attached to the head of the block so that the order is always maintained.
- Specify a number or letter as the heading number.
- Attached to the head of the message, and it is stored in each message array.

(4) **Create file f_{hide} to hide and working directory d_{work}**

- Create a directory to contain files to hide.
- Set the file name to hide and create n files.

(5) **File index entry to hide Record in slack area**

- After recording the slack in the index record, delete the file after changing the file name, and the deleted file in the last MFT entry will be recorded as a camouflage file f_{camo}.
- The name of the file to hide is set to the preceding character in alphabetical order to the header character. Because index entries are always sorted in alphabetical order, the file name is selected to precede the index entry to the message to hide.

- The first block (file name) of the message to hide is immediately after the file to be left on the index entry.
- After deleting the first file to hide, change the name of the file to hide at the end to the name of the camouflage file f_{camo}. Delete the file name immediately after changing it.
- Re-create the first file to hide. As a result, the last file to hide is deleted, and the last entry in the index entry remains in the slack area. In the MFT entry, the information about the file to hide is not left whereas the information of the camouflaged file f_{camo} is deleted.
- Repeat the above steps for all files to hide. Finally, only the anchor file will remain in the directory.

3.2 Schematic diagram of a relationship between MFT entry and index record

In this section, we simply explain an MFT entry and an index record with a schematic diagram, and describe the data structure of an index record and an index entry.

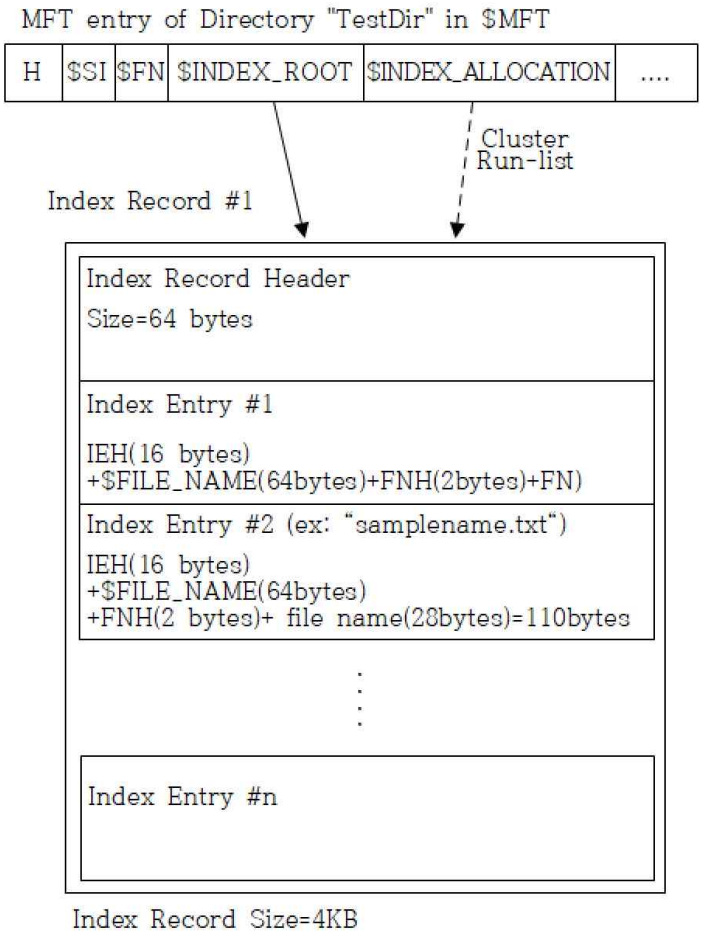

Figure 1. MFT entry and Index Record

In figure 1, there are several lists of the directory. Surplus index entries cannot be contained in the $INDEX_ROOT as a resident type, so $INDEX_ALLOCATION attribute is used to store in the index record as a non-resident type. The 4KB-sized index record#1 contains several index entries. The index entry (abbr.

IE) has a 16-byte index entry header(abbr. IEH), and then the $FILE_NAME (abbr. $FN) attribute follows, which stores Unicode characters 1~ 255 characters long as a file name (abbr. FN) after the 64-bytes $FILE_NAME attribute header. The first two bytes (abbr. FNH) in the file name part are used to indicate the information of the file name; i.e., the first one represents the length of an entire filename, and the second is used to represent a file namespace.

3.3 Data structure of an index record and an index entry

(1) Structure of an index record[9]

One index record size is 4KB, and it contains an index record header and several index entries. It consists of an index record header (0 to 23 bytes) and an index node header (24 to 63 bytes) from the beginning. Then the index entry (64 bytes) follows. A fixup array is stored from 40 bytes to 63 bytes[9].

Table 1. Structure of index record

Offset	Size	Descriptions
0-3	**4**	"INDX" Signature
4-5	**2**	Offset to Fixup Array
6-7	**2**	Entry Number of Fixup Array
8-15	**8**	$LogFile Sequence Number (LSN)
16-23	**8**	VCN of This Index Record
24-27	**4**	Offset to Start Position of Index Entry
28-31	**4**	Offset to End of Used List of Index Entry
32-35	**4**	Offset to End of Allocated Index Entry
36-39	**4**	Flags
40-63	**24**	Fixup Array
64-4095	variable	Index Entries

(2) Structure of an index entry[9]

An index entry has information of the file and directory list in a directory, which has the MFT reference number (8 bytes) and entry length information(2 bytes). The $FILE_NAME attribute length and flags (child exists or last entry) follows. If the flag information is set to child exists, then the last 8-bytes is added for the VCN value of the child node [9].

Table 2. Structure of index entry

Offset	Size	Descriptions
0-7	**8**	MFT Reference of This File Name
8-9	**2**	Length of Entry
10-11	**2**	Length of $FILE_NAME Attribute
12-15	**4**	Flags(1:Child Exists,2:Last Entry)
16+	variable	$FILE_NAME Attribute(If flags=1)
Last 8bytes	**8**	VCN of Child Node

(3) Structure of $FILE_NAME[9]

Every index entry has a $FIL_NAME attribute located at 16bytes offset from the beginning. The attribute contains the following: File Reference of Parent Node (8 bytes), File Creation Time (8 bytes), File Modification Time (8 bytes), MFT Modification Time (8 bytes), File Access Time (8 bytes), File Allocation Size (8 bytes), Real File Size (8 bytes), Flag (4 bytes), Reparse (8 bytes), Length of File Name (1 byte), and File Namespace (1 byte). The variable size of File Name is shown in Table 3[9].

Table 3. Structure of $FILE_NAME

Offset	Size	Descriptions
0-7	**8**	MFT Reference of Parent Node
8-15	**8**	File Creation Time
16-23	**8**	File Modification Time
24-31	**8**	MFT Modification Time
32-39	**8**	File Access Time
40-47	**8**	File Allocation Size
48-55	**8**	Real File Size
56-59	**4**	Flag
60-63	**4**	Reparse
64-64	**1**	Length of File Name
65-65	**1**	Namespace
66+	variable	File Name

4. A maximum data allocation rule for the anti-forensic data hiding method

To calculate the number of index entries that can be placed in an index record, the fixed and the variable factors to consider are as follows.

1) Index record size: 4KB(4,096 bytes)
2) Index record header (IRH): 64 bytes
3) Index entry header (IEH): 16 bytes
4) Fixed elements of $FILE_NAME attribute (FNA): 64 bytes
5) File name of $FILE_NAME attribute: variable
6) End mark of index entry (EIE): 16 bytes

In addition, there is one additional factor to consider. It should be further taken into account that the file name is allocated in 8-byte units.

7) File name allocation unit: 8 bytes
8) The first two bytes are used for file name information, one is for the length information and the other is for the namespace. Therefore, 6 bytes (3 Unicode characters) (i.e., except 2 bytes) are allocated to the shortest file name. For the file name, a total of 512 bytes are used, but only 255 characters (510 bytes can be used) are available excluding the first 2 bytes.

9) When considering the length of the file name, the directory path is included. Only the path name excluding the drive letter is considered in the file name. However, it should be noted that the path itself is not stored in the file name. For example, if a file is listed in d:\DirectoryPath1\SubPath1\SubSub2\, you can store only 223 characters except for the path name of 32 characters excluding the drive letter "d:\". If the directory path is included, this means that the maximum of 255 characters cannot be stored in the file name itself.

Among these various factors, only the file name is variable. Therefore, in order to calculate the optimal number of characters that can fit in an index record, a linear equation can be solved by taking into account variable file names. After dividing the whole data to hide into blocks of arbitrary sizes, file names are made with the data blocks, and then the maximum number of characters that can be entered into the index record is determined by the following formula.

$$TotalIR = IRH + (IEH + FNA + LFN)*n + EIE \quad (1)$$

where n is the number of index entries, LFN is the length of the file name, and TotalIR is all of the characters written in a Index Record including the index record header, index entries, etc.

To maximize the number of characters that can be written in an index record, it is necessary to solve the function of the length of the file name and the number of index entries. This is the process of determining the length of a file name so that it can store the largest number of characters in a 4KB-sized index record. IRH + IEH + FNA + EIE are the fixed elements. If the number of files is large, the number of fixed elements IEH + FNA of the index entry increases. Therefore, it is preferable to calculate the space efficiency by selecting the smallest number of files.

The number of possible index entries (the number of files) for the maximum possible length of 255 characters by the following calculation is found as an integer value of 6.

$$TotalIR = IRH + (IEH + FNA + LFN)*n + EIE \quad (2)$$
$$= 64+ (16 + 64 + 2+ 255*2) * n + 16 = 4{,}096$$
$$n = \lfloor (4{,}096 - 80) / 592 \rfloor = \lfloor 6.78 \rfloor = 6$$

The minimum number of characters constituting six index entries is 244 characters. However, since the number of characters from 244 to 247 is allocated to the same 8-byte block, the largest value of 211 characters must be calculated. The results are as follows.

$$TotalIR = IRH + (IEH + FNA + LFN)*n + EIE \quad (3)$$
$$= 64+ (16 + 64 + 2+ 247*2) * n + 16 = 4{,}096$$
$$n = \lfloor (4{,}096 - 80) / 576 \rfloor = \lfloor 6.78 \rfloor = 6$$

When the file name has a length of 244 to 255 characters, six index entries can be constructed in the index record. When the file name has 208 to 243 characters, the number of index entries is 7. For the maximum case of 243 characters, the following results are obtained.

$$TotalIR = IRH + (IEH + FNA + FNH + LFN)*n + EIE \quad (4)$$
$$= 64+ (16 + 64 + 2+ 243*2) * n + 16 = 4{,}096$$
$$4096= 64+568n+16$$

$$n = \llcorner (4{,}096 - 80) / 568) \lrcorner = \llcorner 7.07 \lrcorner = 7$$

If the number of index entries is 7, the minimum number of characters is 208, but 208 to 211 are allocated to the same 8-byte block. The following results can be obtained by calculating the largest value of 211 characters.

$$\begin{aligned} TotalIR &= IRH + (IEH + FNA + FNH + LFN)*n + EIE \quad (5) \\ &= 64+ (16 + 64 + 2+ 211*2) * n + 16 = 4{,}096 \\ 4096&= 64+504n+16 \\ n &= \llcorner (4{,}096 - 80) / 504 \lrcorner = \llcorner 7.96 \lrcorner = 7 \end{aligned}$$

The same method can be applied to calculate the minimum number of characters, one, as a file name. In this case the number of index entries is 45, which can be obtained as listed in Table 4. In Table 4, the cases of index entries n=6~45 are summarized. For the case of index entries n=7, 1,701 characters are recorded, which is the most efficient. It can be seen that the efficiency is worst when the number of index entries is n = 45, because the number of characters to be stored is very small compared to the number of header characters.

Table 4. Structure of $FILE_NAME

No of IE per IR	Chars. range	One IE Size	Max Stored chars.	Header size of IR	Header size per IE	Last Entry Mark	All chars in IR	Empty space in IR
6	**244~254**	592	1,524				3,632	-464
7	**208~243**	568	1,701				4,056	-40
8	**180~207**	496	1,656				4,048	-48
9	**160~179**	440	1,611	64	80	16	4,040	-56
10	**140~159**	400	1,590				4,080	-16
…	**…**	…	…				…	…
45	**1~3**	88	135				4,040	-56

5. Experiments

We executed a test case of the proposed the maximum data allocation rule for the anti-forensic data hiding method in a Windows 10 operating system and an NTFS v3.1 file system. The test environment specifications are as follows.

Disk Format: NTFS v3.1
Storage drive: Samsung SSD(463GB)
Disk Allocation Cluster Size: 4KB(4,096 bytes)
Working Directory: d:\testDir
Disk Analysis Tool: WinHex v15.4(X-Ways Software)

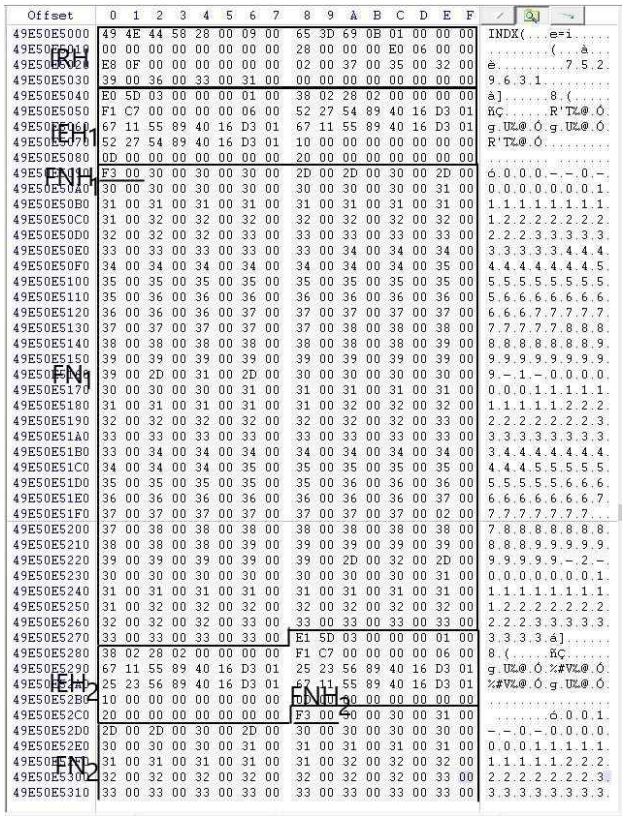

Figure 2. Index record header and Index entry#1

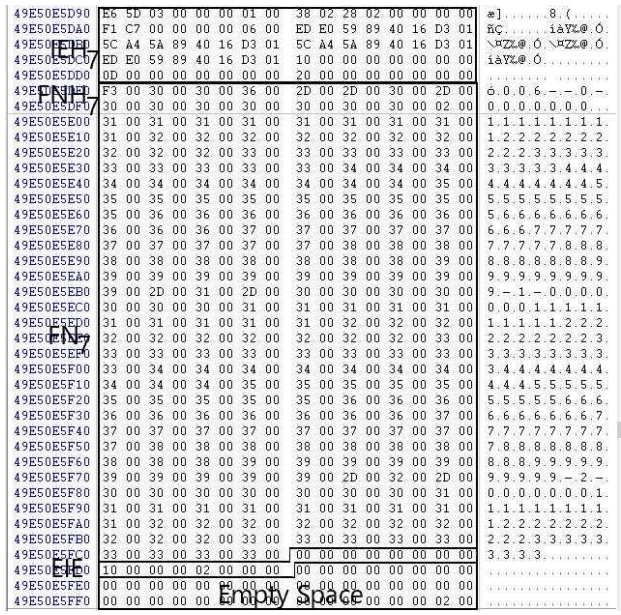

Figure 3. Index entry#7, End of index entry, Empty space

The purpose of this experiment is to confirm the maximum character allocation for the proposed method. In Section 4, a method to find the optimal solution through Equations 1 ~ 5 is provided. In this case, we can verify that the maximum number of index entries with n = 7, which is the maximum condition found in

Table 4, gives the maximum value that can be stored in the index entry when the length of the file name is 243 characters. Figure 2 shows the interior of an index record, where a 64-byte index record header, marked as “IRH”, and 7 index entries are contained. The first index entry marked the index entry header as “IEH#1”, a file name header as “FNH#1”, and a file name as “FN#1”. File information for the first entry 0xF3(243) gives the file name length and 00 is the file namespace.

Figure 3 shows the last entry marked as “IEH#7”+“FNH#7”+“FN#7”, which has the same size and structure as index entry #1. The “EIE” is a 16-byte size of information indicating the end of an index entry. The next following space is empty, and the empty space is 40 bytes, as determined by calculating Eq. 1~5 and as given in Table 4. It furthermore can be confirmed that the free space is 40 bytes in Figure 4.

5. Conclusion

In this study, we proposed a computational method for storing optimal data to hide data in an index record of NTFS and obtained the optimal solution by applying the method. The method to hide data in a 4KB-sized index record is designed for anti-forensics. which records data to hide as a file name in index entries and thereafter the index entries are made to remain in the intentionally generated slack area.

This problem is a function of the length of the character and the number of index entries, with some difficult conditions that are described in Section 2. The solved function for the problem is presented in Section 4, and the optimum solution is obtained by using the method. In Section 5, we show the result of analyzing the case where the number of index entries is n = 7 with screen captures of index entries.

The software tools for this method developed thus far have only basic functions. Developing a well-suited tool is a future work, which has features such as the ability to read, write, and delete data at specific index entry locations. It is also necessary to apply encryption of data to further enhance security.

Acknowledgement

This study was supported by a grant from Dong Yang University in 2016 and by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education(NRF-2016R1D1A1B03935646).

References

[1] Wikipedia, Anti-computer forensics, https://en.wikipedia.org/wiki/Anti-computer_forensics
[2] Michael T. Raggo and Chet Hosmer, Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems, Mobile Devices and Network Protocols, Elsevier, 2013.
[3] H. Carvey, Windows Forensics and Incident Recovery, Addison-Wesley, 2005.
[4] Metasploit, Anti Forensics Project, http://www.metasploit.com/research/projects/antiforensics/
[5] 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.
[6] K. Eckstein and M. Jahnke, “Data Hiding in Journaling File Systems,” Proceedings of Digital Forensic Research Workshop (DFRWS 2005), pp. 1-8, Aug. 2005.
[7] G.-S. Cho, “A New NTFS Anti-Forensic Technique for NTFS Index Entry,” The Journal of Korea Institute of Information, Electronics, and Communication Technology, Vol. 8, No. 4, 2015.
[8] 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.
[9] B. Carrier, File System Forensic Analysis, Addison-Wesley, 2005.