Journal of The Korea Society of Computer and Information
Vol. 23 No. 8, pp. 85-93, August 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.08.085

A Method of Data Hiding in a File System by Modifying Directory Information

Gyu-Sang Cho*

Abstract

In this research, it is proposed that a method to hide data by modifying directory index entry information. It consists of two methods: a directory list hiding and a file contents hiding. The directory list hiding method is to avoid the list of files from appearing in the file explorer window or the command prompt window. By modifying the file names of several index entries to make them duplicated, if the duplicated files are deleted, then the only the original file is deleted, but the modified files are retained in the MFT entry intact. So, the fact that these files are hidden is not exposed.

The file contents hiding is to allocate data to be hidden on an empty index record page that is not used. If many files are made in the directory, several 4KB index records are allocated. NTFS leaves the empty index records unchanged after deleting the files. By modifying the run-list of the index record with the cluster number of the file-to-hide, the contents of the file-to-hide are hidden in the index record.

By applying the proposed method to the case of hiding two files, the file lists are not exposed in the file explorer and the command prompt window, and the contents of the file-to-hide are hidden in the empty index record. It is proved that the proposed method has effectiveness and validity.

▸Keyword: Data hiding, Directory, Modifying direcotory information, File System, NTFS

I. Introduction

데이터를 숨기기 위한 방법으로써 암호화, 스테가노그라피, 워터마킹 방법들을 사용한다. 숨기려는 대상과 목적에 따라서 다양한 방법이 적용될 수 있다[1]. 컴퓨터시스템에서는 데이터를 저장하고 관리를 하기 위한 목적으로 파일시스템을 사용하고 있는데 파일시스템의 구조나 동작 특성을 이용하여 데이터를 숨기는 여러 방법들이 연구되었다[1,2].

Raggo와 Hosmer의 저서[1]에서는 윈도우즈와 리눅스 운영체제에 사용할 수 있는 다양한 방법과 도구에 대하여 소개하고 있다. 윈도우즈의 ADS(Alternate Data Streams), Stealth ADS, Volume Shadowing방법 등을 소개하고 있고, 리눅스의 Linux Filename Trickery, Extended File System Data Hiding, TrueCrypt등의 방법 등을 소개하고 있다.

최근에 N. A. Hassan and R. Hijazi[2]등의 저서에서는 윈도우즈에서의 데이터 숨기기 기법에 관한 여러가지 방법 즉, ADS, 복원점, 레지스트리, 파일슬랙 공간, 은닉 파티션, 디스크 배드블록(bad blocks), HPA(Host Protected Area)와 DCO(Device Configuration Overlay)등을 이용하는 방법을 소개하고 있다. 또한, 데이터 삭제를 위한 하드디스크 지우기방법과 휴지통, 레지스트리, 복원점, 가상메모리, 프리페치(Prefetch)등에 관한 다양한 안티포렌식 기술들에 대해 논하고 있다.

데이터 숨기기를 위한 전통적인 방법인 Metasploits의 Slacker[3]는 NTFS (New Technology File System)에서 실제보다 더 큰 파일을 생성하여 데이터를 숨기는 기능의 오픈소스 도구로 잘 알려져 있다. FragFS[4]는 Thompson이 개발한

• First Author: Gyu-Sang Cho, Corresponding Author: Gyu-Sang Cho
*Gyu-Sang Cho (cho@dyu.ac.kr), School of Public Technology Service, Dongyang University
• Received: 2018. 08. 01, Revised: 2018. 08. 20, Accepted: 2018. 08. 21.
• This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2016R1D1A1B03935646)

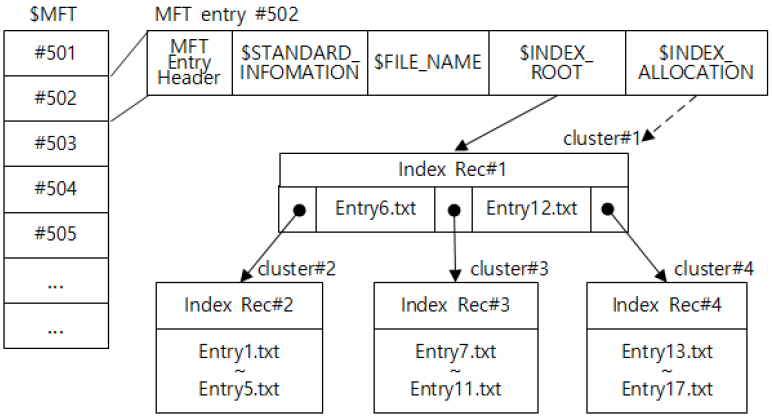

Fig. 1. MFT entry and B-tree structure

NTFS의 MFT(Master File Table) 영역 안에 데이터를 숨기는 안티포렌식 도구이다. Linux의 파일시스템인 Ext2/3(Extended File System 2/3)에서 안티포렌식을 하기 위해 숨겨진 데이터를 찾아내는 방법에 관한 연구가 수행되었는데 S. Piper등[5]이 제안한 이것은 파일시스템의 사용되지 않는 예약된 영역에 숨겨진 데이터를 검색하는 방법이다.

Huebner등[6]은 NTFS파일 시스템에 숨겨진 데이터를 탐지하고 복구하는 방법을 소개하였다. 이 연구는 파일시스템의 구조만을 이용한 방법으로 파일시스템의 데이터구조의 특성을 이용하여 데이터를 숨기는 방식을 적용하고 있다. $BadClus, $DATA 속성, $Boot 파일 등의 메타파일을 이용하여 데이터를 숨기는 방법과 ADS, 디렉토리 $DATA속성 등의 데이터 파일을 이용한 방법, 그리고 볼륨, 파일시스템, 파일의 슬랙 등의 슬랙을 이용한 방법들이 디지털 포렌식 분석방법에 의하여 감지되지 않도록 하는 것을 목적으로 하고 있다.

Cho[7]와 Neuner 등[8]은 파일시스템의 타임스탬프에 데이터를 숨기는 연구를 수행하였다. 두 연구는 같은 시기에 유사한 방식을 제안하였는데 각기 NTFS의 타임스탬프의 1초미만 시간 표시부분에 데이터를 숨기는 방법이다. T. Gobel 등[9]은 리눅스의 Ext4 파일시스템에서 같은 방식을 구현한 연구를 수행하였다.

P. Grd[10]등은 파일시스템에서 B-tree의 동작의 영향으로 발생한 정보에 대하여 컴퓨터 포렌식 관점에서 분석한 연구를 수행하였다. 이 연구보다 진보된 연구는 Cho의 연구[11]에서 수행되었는데 NTFS 파일시스템에서 적용된 B-tree의 방식에 대한 연구이다. 이론적인 B-tree와 NTFS안에서 구현된 B-tree의 차이점에 대한 분석연구를 수행한 것이다. 연구[12]에서는 디렉토리의 인덱스 레코드에 데이터를 숨기는 새로운 방법을 제안하였다. 파일이 삭제될 때 인덱스 레코드 안의 인덱스 엔트리들이 재정렬하면서 배치되는 방식을 이용하여 인덱스 레코드에 데이터를 숨겨 드러나지 않도록 한다.

한편, 파일시스템의 정보를 변조한 연구가 시도되었는데 A. Srinivasan[13]등은 FAT12 파일시스템에서 파일명을 중복하여 앞선 파일 ID를 가진 파일의 내용이 사용자에게 보여지는 방식의 데이터 숨기기를 시도하였다. 이 방법은 파일 시스템의 정보를 변조한다는 시도가 의미가 있지만 구식 FAT12 파일시스템에서 구현되었고 포렌식 분석으로 쉽게 노출될 수 있다는 단점이 있다. Fu-Hau등[14]은 윈도우즈 XP에서 MFT 레코드의 파일의 데이터 섹터의 위치와 순서를 바꾸어 숨기는 방식을 채용하여 복구툴을 사용해도 데이터가 숨겨진 사실이 드러나지 않는 방법을 연구하였다.

이 연구에서는 NTFS 파일시스템에서 디렉토리 목록정보를 변조하여 데이터를 숨기는 새로운 방법을 제안한다. 이 방법은 두 부분으로 구성된다. 인덱스 레코드에 들어있는 인덱스 엔트리의 파일명을 변조하여 여러 파일명을 중복시킨 후 삭제하면 MFT엔트리에서 원본 파일은 그대로 남아 있지만 디렉토리 목록에서는 삭제되어 탐색기 창이나 명령 프롬프트에서 파일명이 표시되지 않도록 하는 것이 파일목록 숨기기 방법이다.

디렉토리에 파일을 많이 만들면 4KB 크기의 인덱스 레코드가 여러 개 생성된다. 파일들을 삭제한 후에도 빈 인덱스 레코드들을 회수하지 않고 빈 상태로 그대로 두는 특징이 있다. 이것의 런-리스트 정보를 구하여 숨기려는 파일의 클러스터 번호로 변조하면 숨기려는 파일의 내용이 인덱스 레코드로 옮겨져서 내용이 들어 있는지 알지 못하게 하는 것이 파일의 내용 숨기기 방법이다.

이 연구의 2장에서는 제안한 방법의 원리에 대한 이해를 돕기 위하여 필요한 NTFS 파일시스템의 인덱스 레코드와 인덱스 엔트리에 대한 데이터 구조를 설명하기로 한다. 3장에서는 이 연구에서 제안한 디렉토리 인덱스 엔트리 정보를 변조하여 데이터 감추기 방법에서 사용하는 파일목록 숨기기 방법과 파일내용 숨기기 방법의 두 가지 방법을 소개한다. 4장에서는 숨기려는 파일의 내용을 인덱스 레코드에 연결하여 파일탐색기 창이나 명령프롬프트 창에 표시되지 않고 데이터가 숨겨진 사실이 노출되지 않는다는 것을 사례를 들어 입증하기로 한다. 5장에서는 제안된 방법에 관련한 논의점에 대하여 언급하며 6장에서 결론으로 마무리 한다.

II. Directory Indexing of NTFS

1. B-Tree Directory Indexing

NTFS에서 디렉토리 내의 파일들을 관리하기 위한 방법으로 B-트리 데이터 구조를 사용하고 있다. 이것은 많은 파일들을 효과적으로 관리할 수 있고 빠르게 파일을 검색할 수 있는 기능을 갖추고 있다. 이것은 파일의 개수가 많이 증가하여도 트리의 높이가 급격하게 높아지지 않는 균형트리이다[11,15].

파일과 디렉토리에 관한 정보는 $MFT파일 안에 기록되어 있다. 이 안에 1KB 저장 공간을 갖고 있는 MFT 엔트리들이 구성되어 있는데 파일이나 디렉토리에 관한 정보를 여러 속성들에 담고 있다. 그림 1의 왼쪽 편에는 $MFT 파일 안의 여러 엔트리들을 나타낸 것이다. 이 그림에서 #502번의 MFT 엔트리가 디렉토리 정보를 갖는 형태이다. 예를 들어 이 디렉토리 이름이 “testDir”라고 할 경우 $STANDARD_INFORMATION과 $FILE_NAME 구조 안에 “testDir”의 이름과 타임스탬프들을 저장하게 된다. #502번의 MFT 엔트리는 디렉토리이므로 $INDEX_ROOT속성을 갖게 되고 이 안에는 B-트리의 루트노드의 위치에 관한 정보가 들어 있다. 트리가 여러 개의 노드로 구성되는 경우에는 MFT 엔트리 밖에 외주(Non-resident)속성으로 노드가 구성된다. 이 정보는 $INDEX_ALLOCATION에 들어있는 런-리스트(Run-list) 정보에 의해 유지된다[11,15]. Fig. 1의 경우로 설명하면 cluster#1~#4가 놓인 곳의 정보를 나타낸다.

디렉토리의 목록이 많아지면 외주속성으로 노드를 만들게 된다. 이 노드는 4KB 크기를 갖는 인덱스 레코드라고 부른다. 이 안에 디렉토리 목록들(파일명)이 오름차순으로 소팅되어 저장된다[11,15]. 인덱스 레코드에 디렉토리 목록의 저장공간이 부족하게 되면 새로운 인덱스 레코드를 할당하고 목록들을 분산배치하게 된다. Fig. 1에는 3개의 리프노드 Index Rec#2~#4에 저장되어 있는 경우를 나타낸 것이다. Index Rec#1이 루트 노드역할을 하는데 그 안에는 두 개의 인덱스 키가 들어 있다. “Entry6.txt”보다 작은 파일명들은 Index Rec#2에 들어가게 되고 “Entry6.txt”보다 크고 “Entry12.txt”보다 작은 파일명들은 Index Rec#3에 저장되고 “Entry12.txt”보다 큰 파일명들은 Index Rec#4에 저장된다.

2. Data Structure of Index Record

인덱스 레코드를 분석하기 위해서는 3가지 형식의 구조를 이해해야 한다. 이것은 인덱스 레코드의 헤더를 포함한 전체 구조, 인덱스 엔트리가 저장되는 구조, 그 안에 들어 있는 $FILE_NAME속성 구조 등으로 구성되어 있다.

인덱스 레코드는 4KB의 크기로 구성되어 있다. 그 안에는 인덱스 레코드 헤더 구조와 인덱스 엔트리가 구성된다. 인덱스 레코드의 헤더에는 인덱스 레코드에 관한 정보가 저장되어 있다. 인덱스 엔트리는 파일명 길이에 따라 크기가 가변적으로 결정된다. 64바이트 오프셋까지는 고정크기를 갖는다. Table 1에는 11개의 구성요소에 대한 데이터 크기와 설명이 들어있다. 첫 번째 4바이트는 "INDX" 라는 시그니처가 저장된다. 다음의 2바이트는 Fixup Array가 저장되어 있는 오프셋 값을 나타내고 6-7오프셋의 2바이트는 Fixup Array의 개수를 나타낸다. 8-15의 8바이트 오프셋은 LSN값을 나타낸다. 16-23의 8바이트 오프셋에는 인덱스 레코드의 VCN번호가 저장되어 있다. 24-27의 4바이트 오프셋은 인덱스 엔트리가 시작하는 곳의 위치를 나타낸다. 28-31의 4바이트는 실제 인덱스 엔트리의 끝 위치를 나타낸다. 32-35의 4바이트는 인덱스 엔트리에 할당된 크기의 오프셋을 나타낸다. 36-39의 4바이트는 플래그 값을 나타내고, 40-63의 24바이트는 Fixup Array가 저장되어 있는 곳이다. 64바이트 오프셋 이후로는 인덱스 엔트리가 저장된다[11,15].

Table 1. Structure of an index record [11,15]

Offset	Size (byte)	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 Arrays
64-4095	variable	Index Entries

3. Data Structure of Index Entry

인덱스 레코드의 64바이트 오프셋 이후에는 인덱스 엔트리들이 저장된다. 파일명의 길이는 가변적인 요소이기 때문에 이것에 따라 인덱스 엔트리의 길이가 결정된다. 이 안에는 디렉토리 안에 들어 있는 목록들의 정보가 저장된다. 이것은 16바이트 크기의 인덱스 엔트리 헤더가 놓이고 그 후에 $FILE_NAME속성이 구성된다. 첫 번째 8바이트에는 MFT 레퍼런스 번호가 저장되고, 그 후에 4바이트 크기의 플래그 값(1인 경우 자식노드가 있음, 2인 경우는 마지막 엔트리) 2바이트 크기(10-11 오프셋)의 $FILE_NAME속성의 크기가 저장되고, 그 뒤에 $FILE_NAME속성이 저장된다. 이것은 $MFT파일 안에 들어 있는 것과 동일한 속성이다. 인덱스 엔트리가 인덱스 노드에 들어 있는 경우는 자식노드를 갖게 되는데 8바이트 크기의 자식 인덱스 레코드의 VCN값을 갖는다(Table 2).

16바이트 오프셋 위치부터 $FILE_NAME속성이 놓이게 된다(Table 3). 첫 번째 8바이트에는 부모 노드의 MFT레퍼런스 번호가 저장된다. 그 후에 생성시간, 수정시간, MFT엔트리 수정시간, 접근시간 등의 4가지 타임스탬프(각 8바이트)가 연속으로 저장된다. 다음에는 파일의 할당크기와 실제 파일크기(각 8바이트), Flag와 Reparse정보(각 4바이트)가 저장된다. 파일명 길이(64바이트 오프셋), 파일명 네임스페이스(65바이트 오

프셋)이 저장되고 가변적인 길이는 갖는 파일명이 66바이트 오프셋에서부터 저장된다[11,15].

Table 2. Structure of an index entry [11,15]

Offset	Size (byte)	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
Last 8bytes	8	VCN of Child Node

Table 3. Structure of $FILE_NAME attribute [11,15]

Offset	Size (byte)	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 Entry 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

III. Data Hiding Method by Modifying Directory Information

1. Overview

디렉토리 정보를 변조하여 데이터를 감추기 위한 방법은 NTFS 파일시스템의 데이터 구조를 변조하여 파일의 디렉토리 목록이 파일탐색기나 명령프롬프트 창에 보이지 않도록 하고 이 파일의 내용을 사용하지 않는 디렉토리 레코드에 숨겨두어 내용이 노출되지 않도록 하는 방법이다.

이 방법은 파일목록 숨기기 방법(Fig. 2의 Method-I 부분)과 파일내용 숨기기 방법(Fig. 2의 Method-II 부분)의 두 가지 방법으로 구성된다.

파일목록 숨기기 방법은 디렉토리의 목록 정보가 들어 있는 인덱스 레코드에서 정상 파일의 인덱스 엔트리에 들어있는 파일의 이름과 숨기려는 파일의 이름을 변조하여 중복시키는 방식으로 구현된다. 중복된 파일이름을 삭제하면 MFT엔트리의 원래의 파일은 실제로 삭제되지만 나머지 변조하여 중복시킨 이름의 파일은 인덱스 엔트리에서만 사라진다. 이것의 MFT 엔트리의 원래 파일이름은 변경되지 않고 삭제도 되지 않은 상태로 파일이 존재한다. 결국 디렉토리 목록에서만 삭제되고 해당 파일은 MFT엔트리에 원 상태로 존재한다. 탐색기 창의 파일목록에 나타나지 않게 함으로써 파일을 숨길 수 있게 된다.

파일내용 숨기기 방법은 디렉토리 안에 여러 파일들을 생성하면 이것들의 목록들이 4KB 크기의 인덱스 레코드에 기록된다. 이 공간은 안전하고 정당하게 할당된다. 이 인덱스 레코드는 한번 할당된 이후에 파일목록들이 삭제되어도 안전한 공간으로 그대로 남아 있다. 이런 특징을 데이터 숨기기에 활용한다. 타겟 디렉토리의 빈 인덱스 레코드에 대한 런-리스트 정보를 구하여 숨기려는 파일의 클러스터 번호로 변조하면 숨기려는 파일의 내용이 인덱스 레코드로 옮겨지고 파일의 내용이 사라지게 된다. 빈 인덱스 레코드에 데이터가 들어있다는 사실은 누구도 짐작하기 어렵다.

2. File Directory Hiding

절차 ①: readyDiskEdit() 디스크 드라이브를 직접 편집이 가능한 상태로 준비한다. 특히, 다른 프로세스에 의하여 점유되어 사용불가 상태가 되지 않도록 바이러스 백신을 포함한 실행 프로세스를 체크한다. 또한, 디스크 캐시 기능으로 인한 부작용을 최소화하기 위해서 사용불가 설정을 한다.

절차②: selDir(dir_{src}) or creDir(dir_{src}) - 숨길 파일이 들어 있는 소스 디렉토리를 선택하거나 생성한다. 기존의 디렉토리를 이용하거나 새로운 디렉토리를 만들어서 숨길 파일들에 대한 작업을 수행한다.

절차③: creFile($name^{n}_{src}$), n=6~N, N은 dir_{src}에 들어 있는 전체 파일의 수를 나타낸다.

절차④: 인덱스 레코드가 생성되기 위해서는 디렉토리 안에 파일의 수가 최소 6개가 필요하다. 디렉토리 내의 파일의 수가 MFT 엔트리의 $INDEX_ROOT 속성의 공간이 허용하는 파일의 수가 한정적이다. 이 공간에 저장할 수 없게 되면 non-resident속성의 $INDEX_ALLOCATION을 속성을 만들고 4KB 크기의 인덱스 레코드를 생성한다. 목록의 수가 많아지면 인덱스 레코드들이 여러 개 생성되는데 이것들은 B-트리 구조에 의하여 관리된다[11,15].

절차⑤: selFileHide($conts^{m}_{hide}$) - 숨기려는 파일을 선택하고 데이터를 파일 내용으로 작성한 후에 저장한다. dir_{src}안에 들어 있는 전체 파일의 수가 N인 경우에 숨길 파일로 선택가능한 수는 m=1~N_{hide}이다. 전체 파일목록 중에 최소한 한 개는 정상 파일로 남겨 두어야 한다.

절차⑥: chgFileName($file^{m}_{nor}$, $file^{m}_{hide}$) -숨기려는 파일명을 정상 파일명과 동일하게 디스크 에디터를 사용하여 변경한다. 숨기려는 파일이 여러 개일 때에도 모두 정상 파일명과 같은 파일명으로 변경한다. 변경의 대상이되는 파일명은 인덱스 레코드 안에 들어 있는 인덱스 엔트리의 $FILE_NAME속성에 들어 있는 파일명이다. 이것과 동일한 속성이 MFT엔트리도 들어 있지만 이 안에 들어 있는 파일명은 변경하지 않고 그대로 둔다.

절차⑦: 숨기려는 파일의 수는 N_{hide}로 나타낸다. m이 이 값

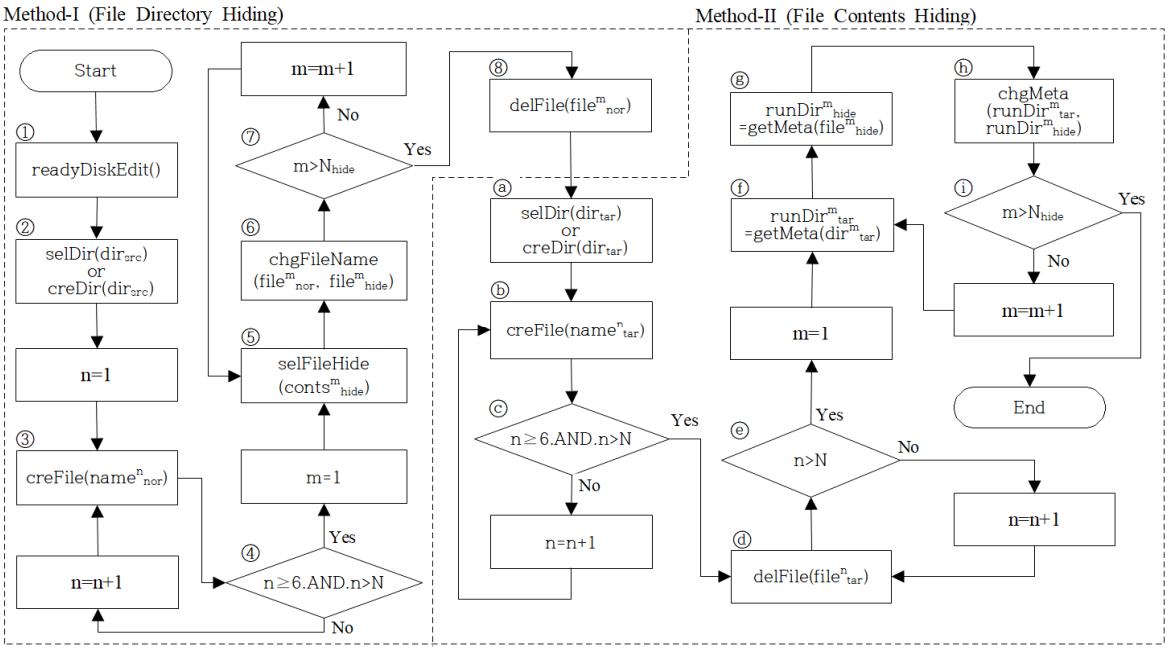

Fig. 2. Procedure of File Data Hiding in an Index Record

에 도달하면 selFileHide(conts$^m_{hide}$)와 chgFileName(file$^m_{nor}$, file$^m_{hide}$)절차를 종료하고 다음 절차로 이동한다. 숨기려는 파일의 수 N_{hide}는 N_{hide}≤N-1이어야 한다.

절차⑧: delFile(file$^m_{nor}$) - 파일 file$^m_{nor}$을 한 개 삭제하면 동일하게 변경된 file$^m_{hide}$ 파일들이 동시에 삭제된다. 숨기려는 file$^m_{hide}$ 파일들은 dir_{src}의 인덱스 레코드에서의 목록이 보이지 않게 된다. 그러나 MFT엔트리에는 파일정보는 유지하고 있다. 파일은 실제로 존재하지만 파일탐색기 창이나 명령프롬프트 창에서는 목록이 나타나지 않게 된다.

3. File Contents Hiding

절차ⓐ: selDir(dir_{tar}) or creDir(dir_{tar}) - 데이터를 숨길 타겟 디렉토리를 선택하거나 생성한다.

절차ⓑ: creFile($name_{tar}$), n=6~N, N은 dir_{tar}에 들어 있는 파일의 수를 의미한다.

절차ⓒ: Method-I의 절차④에서 설명한 바와 같이 이것은 최소 6개 이상의 파일로 구성되어야 한다. 이 숫자는 인덱스 레코드를 생성하기 위하여 필요한 최소한의 파일 수를 의미한다.

절차ⓓ: delFile(file$^n_{tar}$) - 디렉토리에 들어 있는 file$^n_{tar}$ 파일들을 차례로 삭제한다.

절차ⓔ: 절차ⓓ의 과정을 dir_{tar} 디렉토리에 들어 있는 모든 파일들에 대하여 수행한다. 이 과정은 인덱스 레코드에는 파일들의 인덱스 엔트리가 남아있지 않도록 하기 위함이다. 즉, 빈 인덱스 레코드를 만드는 과정이다.

절차ⓕ: runDir$^m_{tar}$를 구하는 과정이다. dir_{tar}의 런-리스트 정보를 의미하는데 이것은 인덱스 레코드의 클러스터 번호를 의미하는 것이다. 이것은 MFT 엔트리 내의 속성 정보들 중에서 $INDEX_ALLOCATION에 들어 있는 런-리스트를 구한다.

절차ⓖ: runDir$^m_{hide}$를 구하는 과정이다. 숨기려는 파일의 MFT entry에 들어 있는 $DATA속성의 런-리스트 정보를 구한다.

절차ⓗ: chgMeta(runDir$^m_{tar}$, runDir$^m_{hide}$) - 이것은 절차ⓕ와 절차ⓖ에서 구한 runDir$^m_{tar}$와 runDir$^m_{hide}$를 서로 바꾸는 과정이다. runDir$^m_{tar}$는 빈 인덱스 레코드이다. 이것을 숨길 데이터가 들어 있는 runDir$^m_{hide}$클러스터 번호로 바꾸는 것이다. 이 과정을 수행한 후에도 디스크의 무결성 측면에서 문제가 발생하지 않는다.

절차ⓘ: 이 절차는 숨기려는 파일의 수 N_{hide}에 도달할 때까지 반복된다. 이 값에 도달하면 getMeta(dir$^m_{tar}$), getMeta(file$^m_{hide}$), chgMeta(runDir$^m_{tar}$, runDir$^m_{hide}$)절차가 끝나고 전체 데이터를 숨기는 과정이 종료된다.

IV. Experiments

1. Experimental Environment

이 연구에서 수행하는 사례들은 다음과 같은 테스트 환경에서 작업이 이루어진다.

OS : Windows 10 Pro(Ver. 10.0.17134.165)
Disc Format : NTFS v3.1
Storage drive: 2.5“ HDD SCSI
Storage Space : 100GB Partition(Total 931GB)
Disc Allocation Cluster Size : 4,096 bytes
Test File : dataHide01.txt ~ dataHide06.txt
Source(Working) Directory : e:₩hideDataSrc
Target Directory : e:₩hideDataTarget1,

e:₩hideDataTarget2

Disk Editing Tool : WinHex Ver. 18.4

2. Experimental Case

파일디렉토리 숨기기 방법은 다음의 절차①~⑧의 과정으로 수행된다.

절차①: 다른 프로세스에 의하여 점유되지 않도록 프로세스 관리를 하고 디스크 에디트 도구 WinHex를 준비한다.

절차②: 소스 디렉토리 e:₩hideDataSrc를 생성한다.

절차③: dataHide01.txt을 생성한다.

절차④: dataHide06.txt 파일까지 6개 파일을 생성한다. (Fig. 3)

절차⑤: 숨기려는 파일을 dataHide02.txt와 dataHide03.txt을 선택하고 각각 3,150바이트와 2,851바이트의 데이터를 저장한다. N_{hide}=2가 된다. 숨기는 파일과 연관된 정상 파일은 filenor은 dataHide01.txt로 선택한다.

절차⑥: 인덱스 엔트리의 파일명 dataHide02.txt와 dataHide03.txt를 WinHex툴을 사용하여 dataHide01.txt으로 변경한다(Fig. 4).

절차⑦: 숨기려는 파일의 수가 N_{hide}=2이므로 절차⑤부터 절차⑧까지 두 파일에 대하여 작업을 수행한다.

절차⑧: $file_{nor}$은 dataHide01.txt파일을 삭제하면 같은 이름의 나머지 두 파일이 동시에 삭제된다(Fig. 5).

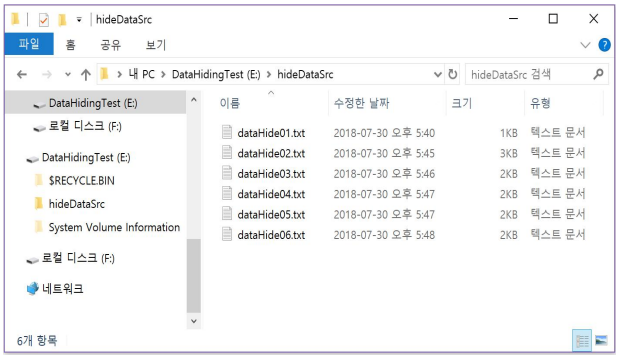

Fig. 3. Directory and File Creation

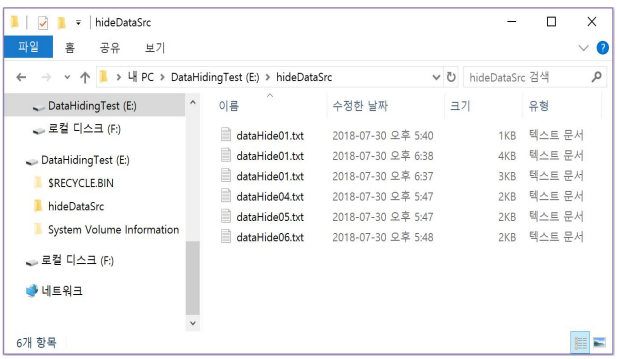

Fig. 4. File Name Duplication in an Index Record

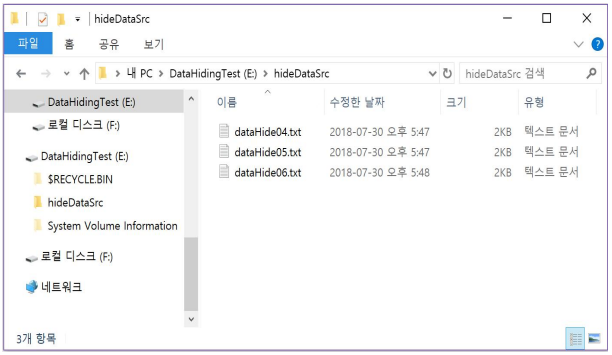

Fig. 5. File Name Duplication in an Index Record

3개의 파일이 삭제되고 난 후에 각 파일의 MFT엔트리의 정보를 확인하면 Fig. 6에서 “00 00”은 해당파일이 삭제되었다는 표식이다. Fig. 7과 8에서 “01 00”은 해당파일이 사용중이라는 플래그 표시이다. Fig. 4에서 본 바와 같이 파일이름이 dataHide01.txt로 바뀌었는데 MFT엔트리 정보 내에서는 Fig. 7과 8의 파일명에는 각각 dataHide02.txt와 dataHide03.txt로 원래의 것 그대로 표시되어 있다. Fig. 4의 탐색기 창에서 파일명이 변경되었고 그 후에 디렉토리 목록에서 사라졌다.

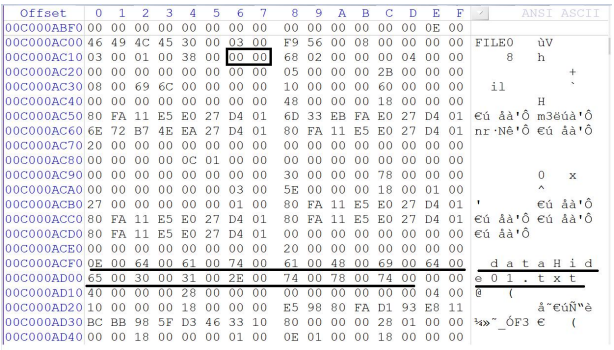

Fig. 6. MFT entry of dataHide01.txt

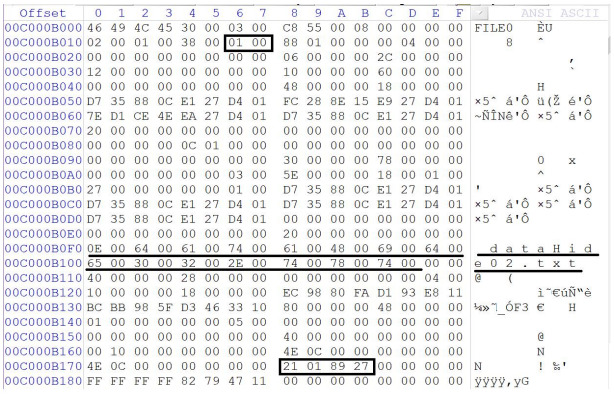

Fig. 7. MFT entry of dataHide02.txt

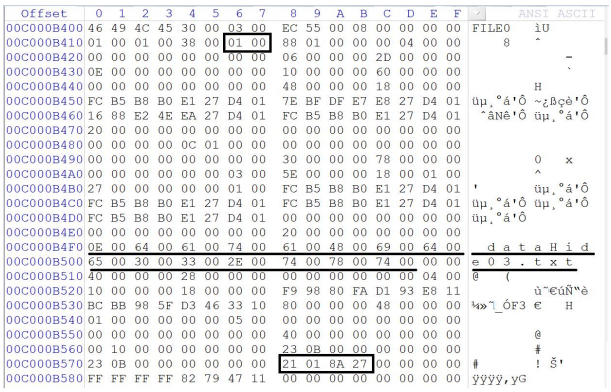

Fig. 8. MFT entry of dataHide03.txt

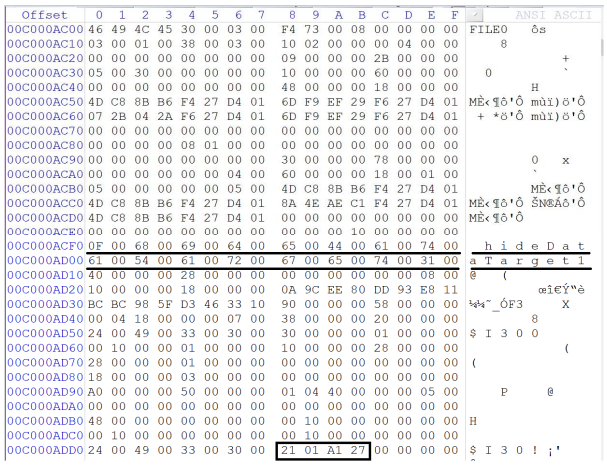

Fig. 9. MFT information of hideDataTarget1 Directory

파일내용 숨기기 방법은 다음의 절차ⓐ~절차ⓘ의 과정으로 수행된다.

절차ⓐ: 데이터를 숨길 타겟 디렉토리 e:₩hideDataTarget1과 e:₩hideDataTarget2를 생성한다.

절차ⓑ: targetHide01.txt을 생성한다.

절차ⓒ: targetHide06.txt파일까지 6개의 파일을 생성한다.

절차ⓓ: targetHide01.txt ~ targetHide06.txt까지의 파일을 차례로 삭제하여 빈 디렉토리를 만든다.

절차ⓔ: e:₩hideDataTarget1과 e:₩hideDataTarget2내의 모든 파일들이 삭제될 때까지 수행된다.

절차ⓕ: Fig. 9에서 “21 01 A1 27"은 ”21“에서 ”2“는 두자리의 런-리스트의 시작 클러스터 번호를 의미하는데 ”A1 27" 이 여기에 해당한다. 이것은 리틀엔디언으로 저장되어서 실제로는 0x27A1에 해당하는 값이다. ”1“은 연속된 클러스터의 갯수가 들어 있는 자리의 크기를 의미한다. ”01“값이 여기에 해당한다. 즉 연속된 클러스터의 수를 나타내는 한 자리의 숫자에 ”01“이 들어 있어서 연속된 클러스터의 수는 1개라고 해석된다. hideDataTarget1 디렉토리의 인덱스 레코드는 0x27A1에서 부터 1개 클러스터가 할당되어 있다는 의미이다.

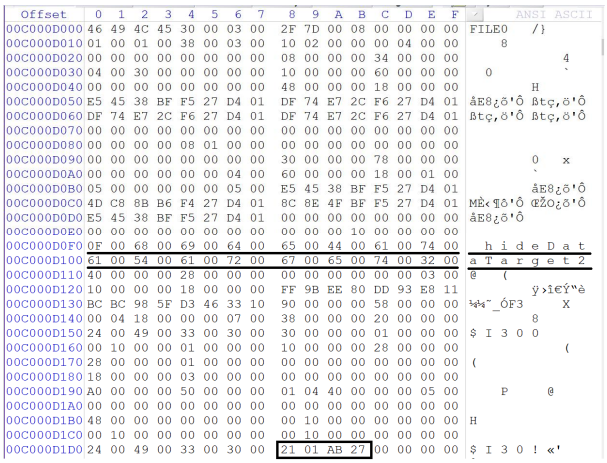

Fig. 10. MFT information of hideDataTarget2 Directory

Fig. 10에서 “21 01 AB 27"은 절차ⓕ에서와 동일하게 해석하여 hideDataTarget2 디렉토리의 인덱스 레코드는 0x27AB에서 부터 1개 클러스터가 할당되어 있다는 의미이다.

절차ⓖ: dataHide02.txt의 숨길 데이터는 Fig. 7의 “21 01 89 27”을 해석하면 0x2789부터 1개의 클러스터에 저장되어 있고 dataHide03.txt의 숨길 데이터는 Fig. 8의 “21 01 8A 27”에서 0x278A부터 1개의 클러스터에 들어 있다.

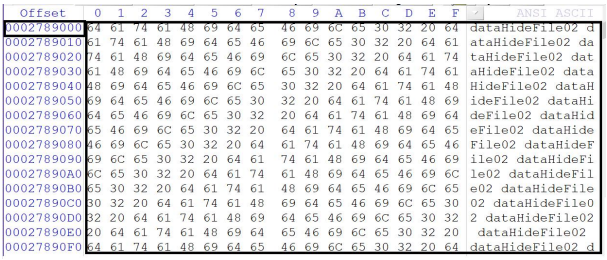

Fig. 11. Changed Index Record of hideDataTarget1 Dir.

절차ⓗ: 절차ⓕ와 절차ⓖ에서 구한 hideDataTarget1 디렉토리의 인덱스 레코드 번호 0x27A1의 1개 클러스터를 dataHide02.txt의 숨길 데이터 0x2789의 1개 클러스터로 바꿔 놓는다. 이 절차의 결과는 Fig. 11에 나타나 있다. hideDataTarget1의 클러스터 시작 번호가 0x2789000으로 바뀌어 있고 내용도 dataHide02.txt의 것으로 변경되었다.

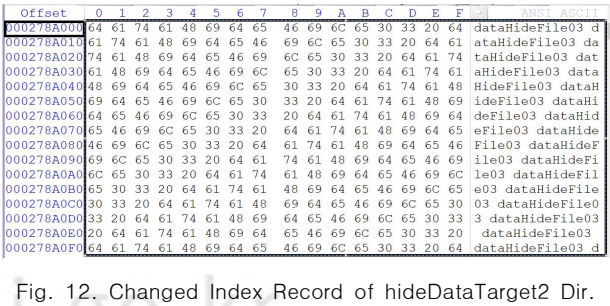

Fig. 12. Changed Index Record of hideDataTarget2 Dir.

hideDataTarget2 디렉토리의 인덱스 레코드 번호 0x27AB의 1개 클러스터를 dataHide03.txt의 숨길 데이터 0x278A의 1개 클러스터로 바꿔 놓는다. 이 절차의 결과는 Fig. 12에 나타나 있다. hideDataTarget2의 클러스터 시작 번호가 0x278A000으로 바뀌어 있고 내용도 dataHide03.txt의 것으로 변경되었다.

절차ⓘ: 이 사례에서는 숨기려는 파일의 수 N_{hide}=2개의 파일에 적용된 후 전체 과정을 종료한다.

V. Discussions

이 방법은 디스크를 직접 에디팅할 수 있어야 가능한 방법이다. X-Ways사의 WinHex Ver. 18.4를 사용하였는데 이것은 디스크를 직접 수정할 수 있는 in-place edit기능을 갖고 있다. 이 툴은 프로그램의 설치 없이 외장 디스크에 포터블(portable) 방식으로 프로그램을 실행시킬 수 있다는 장점이 있다. 프로그램을 사용한 흔적을 남기는 것을 최소한으로 남길 수 있어 로카르드 교환법칙(Locard's exchange principle) 측면에서 유리한 면이 있다.

파일시스템을 경유하지 않고 디스크를 직접 에디팅할 경우에는 디스크 무결성 문제를 야기할 수 있다. 파일시스템에 대한 정확한 이해가 우선시 되어야 이런 문제를 피할 수 있다. 또한 어떤 프로세스가 디스크 장치를 점유하고 있는 경우에도 디스크를 직접 에디팅하지 못하게 하는 원인이 된다. 일부 바이러스 백신의 디스크 감시 기능들이 여기에 해당한다. 이런 여러 가지 문제들을 해결한 뒤에 디스크를 직접 에디팅할 수 있는 상태가 된다.

이 연구에서 사용한 방법은 빈 인덱스 레코드를 확보한 후에 여기에 데이터가 들어 있는 클러스터를 바꿔 넣는 것이다. B-tree의 노드로 사용되는 인덱스 레코드는 인덱스 엔트리가 4KB공간에 채워지면 두 개의 인덱스 레코드로 분할되는 방식을 사용하고 있다[11,15]. 데이터를 감추기 위한 공간의 확보를 위하여 인덱스 레코드를 확보하기 위하여 파일을 생성한 후에 그 파일들을 다시 삭제하면서 빈 인덱스 레코드를 확보하는 방식을 사용한다. 빈 인덱스 레코드가 생겨도 디렉토리가 삭제되지 않고 남아 있다면 다시 회수하지 않는 것은 NTFS 파일시스템의 특징이다[11,15].

4KB 크기의 인덱스 레코드에 채워지는 인덱스 엔트리의 수는 파일명의 길이에 관련있다. 인덱스 엔트리는 확장자를 포함한 파일명의 길이가 8자에서 11자까지는 38개가 저장될 수 있고 12~15자까지는 35개 저장될 수 있다. 한 개의 인덱스 레코드가 채워지면 중간의 키 한 개가 상위노드로 이동하고 나머지 엔트리들은 자식 노드 두 개로 분리된다. 이런 방식이 적용된다는 것을 알고 있으면 필요한 인덱스 레코드의 개수와 파일을 생성하여 인덱스 엔트리의 개수를 계산할 수 있다[16].

이 방식은 절차⑧과 절차ⓓ에서 파일을 생성한 후에 파일을 모두 지우는 방식을 채택하고 있다. 디렉토리에 파일이 남아 있거나 새로운 파일들이 생성되거나 복사되는 경우에 사용중인 파일의 메타정보가 추가 되고 수정되기 때문에 인덱스 레코드 안에 들어 있는 인덱스 엔트리에 변화가 생긴다. 이런 경우는 숨긴 데이터와 충돌이 생긴다. 디렉토리 안의 파일을 모두 삭제하는 것이 안전하여 이 방식을 적용하고 있다. 좀 더 안전한 데이터 숨기기를 위한 방법을 위하여 여러 개의 빈 인덱스 레코드를 연속으로 여러 개 확보하고 앞쪽에 1~2개의 빈 인덱스 레코드를 두고 그 후의 인덱스 레코드에 숨길 데이터를 연결하는 방법을 사용할 수 있다.

VI. Conclusion

이 연구에서는 디렉토리 인덱스 엔트리 정보를 변조하여 데이터를 숨기는 방법을 제안하였다. 이것은 파일목록 숨기기 방법과 파일내용 숨기기 방법의 두가지 방법으로 구성되었다. 숨기려는 파일의 목록을 파일탐색기 창이나 명령프롬프트 창에 표시되지 않도록 하기 위해서 디스크 에디트 툴을 사용하여 인덱스 엔트리를 직접 수정하는 방법을 사용하였다. 숨길 데이터를 B-tree방식으로 구현된 인덱스 레코드를 비어 있는 상태로 만들어서 데이터를 숨기는 공간으로 사용하였다. 이 방법을 6개의 파일이 들어 있는 디렉토리에 2개의 파일의 목록을 숨기기고 데이터를 숨기는 통하여 실험 과정을 보이면서 파일 목록과 데이터가 숨겨져 있는 것이 노출되지 않는다는 것을 구현하였다.

이 연구의 후속 연구로써 제안한 방법을 도구로 구현한 안티포렌식(anti-forensics) 툴이 제작되어야 한다. 이 방법은 기존에 소개되지 않은 방법이므로 이 방법을 적용하였을 때를 대비한 대응 도구를 개발하는데 있어서 시험대로 사용할 수 있는 기능을 갖춘 도구로 개발하는 것이 필요하다.

또한 제안한 방법은 NTFS 파일시스템에서 적용되는 방법만을 소개하고 있다. macOS의 HFS+나 linux의 Ext3/4 파일시스템도 유사한 방식으로 디렉토리 인덱스를 구현하고 있다는 점에 착안하면 다른 파일시스템에서도 이 연구에서 제안한 방법과 유사한 방식의 데이터 감추기 방법이 적용될 수 있을 것이다.

REFERENCES

[1] Michael T. Raggo, Chet Hosmer, "*Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems, Mobile*

Devices and Network Protocols", Syngress, 2012.
[2] N. A. Hassan and R. Hijazi, "*Data Hiding Techniques in Windows OS*", Elsevier, 2017.
[3] Metasploit Anti Forensics Project, http://www.metasploit.com/research/projects/antiforensics/
[4] I. Thompson, and M. Monroe, "FragFS: an advanced data hiding technique", BlackHat Federal. Jan. 2006.
[5] Piper et al., "Detecting hidden data in ext2/ext3 file systems," "*Advanced in Digital Forensics*", pp. 245-256, Springer, 2005.
[6] Ewa Huebner , "Data hiding in the NTFS file system,", Digital Investigation, Vol. 3, Issue 4, pp. 211-226, Dec. 2006.
[7] Gyu-Sang Cho, "Data Hiding in NTFS Timestamps for Anti-Forensics", International Journal of Internet, Broadcasting and Communication, vol. 8, no. 3, pp. 31-40, Aug. 2016.
[8] Neuner, S. et. al., "Time is on my side: steganography in filesystem metadata," Digital Investigation, 18, pp. S76-S86. 2016.
[9] T. Gobel and H. Baier, "Anti-forensics in ext4: On secrecy and usability of timestamp-based data hiding," Digital Investigation, 24, pp. S111-S120, 2018.
[10] P. Grd and M. Bača, "Analysis of B-tree data structure and its usage in computer forensics," Proc. of the 21st Cent. Euro. Conf. on Infor. and Intelli. Sys. pp. 423-428, Sep. 2010.
[11] Gyu-Sang Cho, "Ordinary B-tree vs NTFS B-tree: A Digital Forensics Perspectives," Journal of The Korea Society of Computer and Information, Vol. 22 No. 8, pp. 73-83, Aug. 2017.
[12] Gyu-Sang 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, pp. 327-337, Aug. 2015.
[13] A. Srinivasan, S. Kolli, and J. Wu, "Steganographic information hiding that exploits a novel file system vulnerability," Int. J. Security and Networks, Vol. 8, No. 2, Aug. 2013.
[14] Fu-Hau Hsu1 et. al., "Data concealments with high privacy in new technology file system," Journal of Supercomputing, Vol. 72, Issue 1, pp 120-140, Jan. 2016.
[15] B. Carrier, "*File System Forensic Analysis*", Addison-Wesley, pp. 273-396, 2005.
[16] Gyu-Sang Cho, "A Maximum Data Allocation Rule for an Anti-forensic Data Hiding Method in NTFS Index Record," International Journal of Internet, Broadcasting and Communication, Vol.9, No.3, pp. 17-26, Aug. 2017.

Authors

Gyu-Sang Cho received the B.S., M.S. and Ph.D. degrees in Electronic Engineering from Hanyang University, in 1986, 1989 and 1997, respectively. Dr. Cho joined the faculty of the Department of Computer Inforamtion at Dongyang University, Yeongju, Korea, in 1996. He is currently a Professor School of Public Technology Service at Dongyang University, Dongducheon, since 2017. He is interested in digital forensics, system security, and IoT security.