디지털산업정보학회 논문지 제11권 제4호-2015년 12월
http://dx.doi.org/10.17662/ksdim.2015.11.4.069

디렉토리 인덱스 안티포렌식 기법에서 Windows 파일명에 사용할 수 없는 문자 문제의 해결방법*

조 규 상**

A Problem Solving Method for Non-Admittable Characters of a Windows File Name in a Directory Index Anti-Forensic Technique

Cho Gyusang

〈Abstract〉

This research proposes a modified data hiding method to hide data in a slack space of an NTFS index record. The existing data hiding method is for anti-forensics, which uses traces of file names of an index entry in an index record when files are deleted in a direcotry. The proposed method in this paper modifies the existing method to make non-admittable ASCII characters for a file name applicable. By improving the existing method, problems of a file creation error due to non-admittable characters are remedied; including the non-admittable 9 characters (i. e. slash /, colon :, greater than >, less than <, question mark ?, back slash \, vertical bar |, semi-colon ;, esterisk *), reserved file names(i. e. CON, PRN, AUX, NUL, COM1~COM9, LPT1~LPT9) and two non-admittable characters for an ending character of the file name(i. e. space and dot). Two results of the two message with non-admittable ASCII characters by keyboard inputs show the applicability of the proposed method.

Key Words : Data Hiding, Direcotory Index, Digital Forensics, NTFS, Windows, B-tree

Ⅰ. 서론

NTFS는 Windows의 기본 파일시스템으로 사용되고 있고 안정된 파일시스템으로 알려져 있다. 디지털 포렌식의 관점에서 NTFS 파일시스템은 다른 파일 시스템에 비하여 좀 더 복잡한 구조를 갖고 있어서 이에 대한 동작원리와 구조 등에 대해서 다루고 있는 문서에 의존하여 지식을 얻어야 한다. 이것에 대한 많은 부분들이 각종의 문헌을 통해 잘 알려져있다. 우선 Wikipedia[1]는 NTFS의 역사와 기본적인 구조를 충실히 기술하고 있다. Microsoft의 Technet[2]과 Carrier[3]의 저서에는 상세하게 NTFS의 자료구조가 소개되어 있다.

* 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임. (NRF-2013R1A1A2064426)
** 동양대학교 컴퓨터정보전학과 교수

NTFS에서 디렉토리 인덱스를 위해서 B-tree가 사용되고 있다. 그러나 이것에 대해 자세하게 설명된 기술 문서를 찾아보기 힘들지만 일부 블로그들에서 디지털 포렌식을 위한 B-tree 인덱스에 관련된 자료를 찾을 수 있다. 그 중에서 Ballenthin[4]는 디렉토리 목록을 파싱하고 슬랙 영역에 기록된 내용을 찾을 수 있는 파이썬 프로그램 INDXParser.py를 소개하고 있다. Tilbury의 SANS DFIR 블로그[5]에서는 FTK 툴, icat 툴, EnScript를 사용하여 NTFS 인덱스에 관한 정보를 구할 수 있는 사례를 소개하고 있다. Ballenthin과 Hamm[6]는 4장의 내용으로 구성된 블로그 문서에서 NTFS의 인덱스의 구조에 대한 설명과 관련된 도구의 사용사례, B-tree 인덱스에서 파일이 삭제될 때의 동작에 의해 야기되는 특징을 설명하고 있다.

Huebner등은 포렌식 분석을 하기 위한 목적으로 NTFS파일 시스템에서 데이터를 감추는 방법과 그 방법을 이용한 감지와 복원에 대한 연구를 수행하였다[7]. 이 연구에서는 파일시스템의 데이터구조의 특성을 이용한 여러 가지 방식의 데이터 숨김 방식을 소개하고 있는데 메타데이터 파일 중에서 $BadClus를 이용하는 방법, $DATA 속성에 숨기는 방법, $Boot 파일에 숨기는 방법을 소개하고 있고, 데이터 파일에서 ADS, 디렉토리의 $DATA속성, 추가 클러스터에 데이터를 숨기는 방법, 그리고 볼륨 슬랙, 파일시스템 슬랙, 파일 슬랙 등 파일의 특징을 이용한 방법 등 NTFS의 파일시스템의 특징을 이용하여 데이터 숨기는 전반적인 방법들에 대해서 기술하고 있다.

최근 저자는 타임스탬프 변화패턴을 근거로 한 평가함수에 의한 디지털 포렌식 방법[8]에 관한 연구를 수행하였고 디렉토리에 대한 디지털 포렌식 분석방법[9]에서 디렉토리 안에서 파일연산을 할 때 디렉토리의 정보변화를 분석하였다. 또한 연구[10]에서 디렉토리 안에서 많은 파일들이 생성될 때 NTFS의 B-tree는 어떻게 인덱스 엔트리가 확장되고 파일을 삭제할 때는 어떤 동작 특성을 보이는지 분석함으로써 디렉토리 인덱스의 포렌식 분석에 필요한 정보를 얻는 과정을 소개하였다. 그의 후속연구[11]에서는 디렉토리의 인덱스 구조 내에 메시지를 숨기기 위한 새로운 방법을 제안하였다. 이것은 B-tree의 동작특성을 이용하여 인덱스 레코드의 슬랙 영역에 메시지를 숨기는 방법이다.

이 연구에서는 Windows NTFS에서 디렉토리 인덱스 엔트리 목록이 삭제될 때 생성되는 인덱스의 흔적을 이용하여 데이터를 숨기는 방법에서 파일명에 사용할 수 없는 제약이 있는 문자들에 대한 처리 가능한 개선된 방법을 제안한다. 이 방법은 기존의 연구[11]에서 키보드로 입력하는 ASCII 문자 중에서 9개의 문자들에 대해서는 파일생성 에러가 발생하여 메시지 숨김 처리를 할 수 없는 경우와 파일명으로 사용할 수 없는 예약된 문자열, 파일명 끝 문자로 사용할 수 없는 특별한 경우의 문자처리의 문제점들을 해결한 디렉토리 인덱스에 데이터를 숨김 처리할 수 있는 방법을 제안한다.

제안된 방법에 대한 이해를 위하여 이 연구의 2장에서 디렉토리 인덱스의 데이터 구조를 설명하고 B-tree로 구현된 디렉토리 인덱스에 대한 설명을 한다. 3장에서는 디렉토리 인덱스의 인덱스 레코드에 남는 흔적들을 이용하여 데이터를 숨기는 알고리즘을 소개한다. 4장에서는 파일명으로 사용할 수 없는 문자들에 대한 분류를 하고 5장에서 파일명으로 사용할 수 없는 문자들에 대한 처리방안을 제안한다. 6장에서는 제안된 방식을 두 가지 사례에 적용한 결과를 보인다. 7장에서 제안된 방법의 의미와 새로운 추후 과제에 대해서 논의하며 결론을 내리기로 한다.

Ⅱ. 디렉토리 인덱스 구조

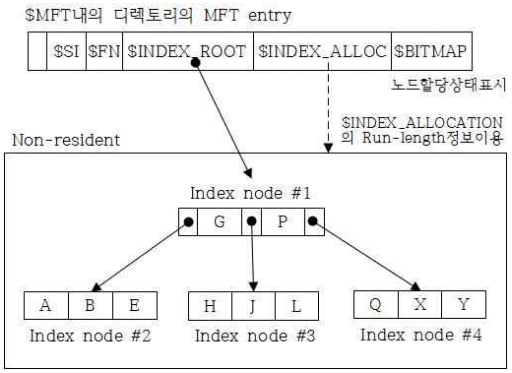
2.1 디렉토리 인덱스를 위한 NTFS의 B-트리

NTFS는 디렉토리 내의 파일들을 관리하기 위하여 B-트리 인덱스 구조를 사용하고 있다. 이것은 파일들을 다루는데 효율적이며 빠른 파일 검색을 할 수 있는 기능을 갖추고 있다. 이것은 트리 내에서 엔트리의 수가 많이 증가하여도 깊이 수준(depth level)을 증가시키는 것을 억제하기 위해 균형을 유지하는 균형트리(balanced tree) 방식이다[3].

NTFS에서 루트 노드 역할은 $INDEX_ROOT속성이 담당하고 있다. 하위노드에 대한 정보는 외주(Non-resident)속성으로 $INDEX_ALLOCATION에 들어있는 Run-length 정보에 의해 유지된다[3]. $INDEX_ROOT속성의 공간은 크기는 대략 700여 바이트 정도 가능하다. 파일명의 길이에 따라 다르지만 8. 3형식의 파일명을 가지는 경우는 보통 5~6개 정도의 디렉토리 목록이 기록될 수 있다.

디렉토리의 목록이 늘어나면 새로운 노드를 외주 속성으로 만든다. 이 때 4KB 크기의 인덱스 레코드(즉, 인덱스 노드)를 만들고 그 안에 많은 디렉토리 목록들이 저장되어 있고 항상 자동적으로 오름차순으로 아스키 코드순으로 소팅된다. 대소문자는 구분하지 않는다. 파일의 길이가 길면 8. 3 포맷의 짧은 파일명과 긴 파일명이 동시에 저장된다[3, 10, 12].

디렉토리 목록들이 증가하면 여러 개의 인덱스 레코들이 여러 개로 확장하게 된다[8]. 인덱스 레코드는 4KB크기를 갖는다. 디렉토리에서 파일 목록들이 삭제되면 인덱스 레코드 내에서 리스트들이 삭제된다. 이미 생성된 인덱스 레코드들은 해당 인덱스 레코드 내에 목록이 들어 있지 않아도 인덱스 레코드 할당영역이 삭제하지 않고 그대로 유지된다. 다만 전체 디렉토리에서 최소한 파일이 1개 이상은 남아 있어야 할당된 인덱스 레코드들이 그대로 유지된다. 디렉토리의 목록이 0개가 되면 할당된 인덱스 레코드들은 모두 해제된다.



<그림 1> NTFS의 B-트리 구성예

Ⅲ. 디렉토리 인덱스 레코드의 슬랙 영역에 메시지를 숨기기 위한 방법

3.1 알고리즘

디렉토리 인덱스 레코드의 슬랙 영역에 메시지를 숨기 위한 방법은 모두 11단계로 구성된다. 각 단계별 내용은 다음과 같다. 이 방법은 참고문헌[9]의 방법을 기초로 만들어 진 것이다. 처음 발표된 알고리즘을 개선한 것이다. 이 논문에서 제안된 방법은 단계 2와 3에 적용된다.

단계 1: 초기화 및 입력

- 작업 디렉토리의 이름을 입력받는다.
- 숨길 메시지를 입력한다.
- 숨길 메시지의 블록 크기를 설정한다.
- 각 변수를 초기화한다.

단계 2: 문자블록 전처리

- 입력된 메시지를 스캔하여 파일명으로 사용할 수 없는 9개의 문자와 이스케이프(escape) 문자로 사용되는 '#'에 대해서 처리한다.

단계 3: 숨길 메시지의 n 등분

- 마지막 메시지 처리할 때 위장파일이 덮어쓰는 부분에 공백문자나 의미없는 문자를 채워넣는다.
- 숨길 메시지를 블록의 크기 b 단위로 일정한 크기를 갖는 n개의 블록으로 나눈다.
- 블록의 끝 문자로 할당된 문자 중에서 끝 문자로 사용할 수 없는 ' '(공백문자, space), '. '에 대해서 이스케이프 문자처리한다.
- 4문자 이하로 블록이 구성된 경우는 예약된 파일명에 해당하는지 비교하고 예약된 파일명이면 앞에 #을 붙인다.

단계 4. 블록 순서지정 위한 머릿번호 삽입

- 인덱스 엔트리는 인덱스 레코드 내에서 알파벳 오름차순으로 자동으로 정렬된다.
- 숨길 메시지가 여러 블록으로 나뉠 때 블록의 선두에 붙여서 항상 순서가 유지되도록 한다.
- 숫자 또는 문자를 머릿번호로 지정한다.
- 메시지 블록의 개수에 따라서 자릿수를 정한다.
- 이것을 메시지의 선두부에 합성하여 각 메시지 배열에 저장한다.

단계 5. 작업 디렉토리 설정

- 숨김 파일들이 들어갈 작업 디렉토리를 설정한다.

단계 6. 고정 파일 생성

- 디렉토리 안에는 최소 1개의 고정 파일이 있게 하여 인덱스 레코드의 할당이 해제되지 않도록 한다.
- 이 파일명은 단계 4의 머릿번호보다 알파벳 순으로 앞선 이름으로 생성한다.

단계 7. 숨길 파일 생성

- 단계 4에서 생성한 메시지로 파일이름으로 정하고 n개의 파일을 생성한다.

단계 8. 첫 번째 메시지 파일 삭제

- 첫 번째 메시지 파일을 삭제하면 알파벳순으로 전체 목록이 재정렬된다.
- 이 때 전체 목록의 맨 끝의 메시지 파일이 있던 자리의 흔적이 남는다.
- 마지막 메시지 처리 시에는 이 단계를 생략한다.

단계 9. 맨 끝 메시지 파일명을 위장 파일명으로 변경

- 맨 끝 파일명을 임의의 위장 파일명으로 변경한다.
- 이것에 의해 MFT 엔트리 내의 파일명 정보에 긴 메시지로 된 파일명 대신에 위장 파일명이 기록된다.

단계 10. 위장 파일 삭제

- MFT 엔트리에서 위장 파일을 삭제한다.
- 메시지 파일과 관련된 정보는 위장 파일을 사용함으로써 MFT 엔트리 내에서 보이지 않고 위장파일이 지워진 흔적만 남는다.

단계 11. 첫 번째 메시지 파일 다시 생성

- 첫 번째 메시지 파일을 다시 생성한다. 맨 끝의 메시지 한 개가 삭제되었고 그 흔적이 인덱스 목록의 맨 끝에 남게 된다.
- 마지막 메시지 처리 시에는 이 단계를 생략한다.
- 처리할 메시지가 1개 이상 남아 있으면 단계 8로 이동한다.

단계 1~11까지의 모든 절차를 마치고 난 후에 디렉토리에는 고정 파일만 남게 된다. Windows의 파일탐색기로는 인덱스 레코드의 슬랙영역에 기록되어 있는 숨김 메시지들을 발견할 수 없다. 디지털 포렌식 도구들을 사용하는 포렌식 과정에서 파일명 검색으로 숨긴 메시지가 들어 있다는 사실을 알 수는 없다.

Ⅳ. 파일명에 사용할 수 없는 문자들

4.1 개요

디렉토리 안에 파일들이 존재하면 4KB 크기의 인덱스 레코드 안에 인덱스 엔트리가 생성된다. 인덱스 엔트리는 그 안에 들어 있는 정보 중에서 파일명을 기준으로 알파벳 순으로 소팅하여 기록하게 된다. 파일이 삭제되면 인덱스 레코드의 뒷부분에 기록되었던 내용이 흔적으로 남는다. 이 점에 주안하여 3장의 알고리즘은 고안된 것이다.

파일명에 메시지를 기록하고 그 흔적을 이용하기 때문에 파일을 생성하는 문제가 이 방법에서의 가장 중요한 문제이다. 그러나 파일명에 사용하는 문자들 중에는 파일명으로 사용될 수 없는 문자들이 존재한다. 숨기려는 메시지 안에 이런 문자들이 들어 있다면 파일생성 에러가 발생하여 원하는 방식으로 메시지 숨기기를 할 수 없게 된다. 그래서 이 장에서는 파일 생성에러를 일으키는 문자들에 대한 조사를 수행하기로 한다.

4.2 키보드의 문자 중 사용 불가능한 문자들

키보드로 입력가능한 문자들 중에서 표 1의 1 그룹에 나열된 9개 문자는 파일명에 사용할 수 없다. 파일명에 한 개의 문자라도 이에 해당하는 문자가 사용된다면 파일이 생성되지 않는다. 이 문자들은 Windows에서 해당 문자마다 파일 작업에 관련된 어떤 예약된 기능이 부여되어 있기 때문에 사용할 수 없는 경우이다.

4.3 ASCII 코드 중 사용 불가능한 문자들

표 1의 그룹 2의 ASCII 코드 집합 중에서 0번 문자인 NULL 문자부터 1~31번의 문자들은 파일명으로 사용할 수 없다. 이 32자의 문자들은 프린트 할 수 없는 제어 문자(unprintable control codes)라고 불리운다. 주변기기나 프린터를 제어하기 위해서 사용하는 코드들이다.

그룹 3은 확장 ASCII 코드 집합이다. 코드번호 128(0x80) ~ 160(0xA0) 사이의 문자들도 파일명에 사용되면 파일생성 에러가 발생한다. 이 문자들 역시 화면에 표시할 수 없는 문자들이다.

그룹2와 그룹3은 모두 다 파일명으로 기록되지 않는 문자들이다. 그러나 둘은 파일명에 사용될 때 다른 결과를 나타낸다. 그룹 2의 문자들은 파일명에 이 문자들이 속해있으면 파일생성 에러를 일으키며 파일 자체가 생성되지 않는다. 그러나 그룹 3의 경우는 이 문자와 다른 문자들과 함께 파일명에 사용이 되면 이 문자가 파일명에서 제외되고 나머지 문자들로만 파일명이 생성된다. 즉, 파일명에 그룹 3의 문자들이 사용되면 파일은 생성되지만 문자의 흔적은 남지 않게 된다.

4.4 특수 기능의 이름들

표 1의 그룹 4에 열거된 22개의 이름은 DOS 시절부터 사용하던 디바이스를 정의하는 파일명들이다. 이 이름을 사용하여 파일을 만들면 파일 생성에러가 발생한다. 이 이름들이 파일명에 단독으로 사용되면

<표 1> Windows에서 파일명에 사용 불가능한 문자들

그룹	항목	문자들
1	키보드 상의 특수문자들[8]	" (34, 0x22) * (42, 0x2A) / (47, 0x2F) : (58, 0x3A) < (60, 0x3C) > (62, 0x3E) ? (63 0x3F) \ (92, 0x5C) | (124, 0x7C)
2	ASCII code의 키보드에 없는 문자[8]	0(0x00)~31(0x1F)번까지의 문자들
3	확장 ASCII code의 문자들	128(0x80) ~ 160(0xA0)까지의 문자들
4	특수 기능의 이름들[8]	CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9
5	파일명의 끝에 사용할 수 없는 문자	. (46, 0x2E) " "(space, 32, 0x20)

안되지만 부분적으로 속해있는 경우는 가능하다. 예를 들어 "PRN.txt"라고 하면 확장자가 붙어있음에도 불구하고 에러처리를 한다. 그러나 "helloPRN.txt"라고 하면 파일생성이 가능하다. 결론적으로 22개의 예약된 디바이스 파일명은 단독 형태가 아닌 다른 글자들과 복합적으로 사용하는 것은 가능하다는 의미이다.

표 1의 그룹 5에서의 구두점 ". "(46, 0x2E)은 파일명의 중간에는 사용이 가능하지만 파일명의 맨 끝에는 사용이 불가능하다. 예를 들면 "a. b. c.txt"는 가능하지만 "a. b. c. "라고 쓰는 것은 불가능하다. 파일명에 맨 끝에 구두점을 붙이게 되면 자동적으로 삭제되어 직전까지 쓰인 문자까지만 기록된다. 또한 스페이스문자 (32, 0x20) 경우도 마찬가지 방법으로 적용된다. 이것을 파일의 끝에 붙이면 자동적으로 삭제되어 직전까지 쓰인 문자까지만 파일명으로 쓰이게 된다.

Ⅴ. 안티포렌식 기법에 사용하기 위한 파일명에 문자처리 방법

5. 1 키보드 문자들을 사용을 위한 방법 제안

이 절의 내용은 숨길 메시지에서 사용하는 문자의 범위를 키보드로 입력가능한 문자에 대해 처리를 위해 제안된 방법을 설명한다.

키보드로 입력가능한 문자 중에서 파일명으로 사용할 수 없는 문자들은 표1의 그룹1에 나타낸 9개의 문자들이다. 이 문자들이 사용되면 파일이 생성될 수 없기 때문에 파일명에 메시지를 저장하여 디렉토리 인덱스의 슬랙영역을 이용한 메시지 숨기는 안티 포렌식 방법을 사용할 수 없게 된다. 그래서 9개의 문자들에 대한 대체 표현방법 "#"문자를 이용하여 이스케이프(escape)문자들을 정의한다. "#"문자 자체가 이스케이프 기호로 사용되므로 이것에 대한 정의도 필요하다. 그래서 전체 이스케이프 문자는 10개가 정의된다. 이스케이프 기호로 "#"을 사용한 것은 여러 기호 문자들 중에서 평문에 사용되는 빈도가 적고 이스케이프 문자로 사용하기에 시각적으로 적합하다고 판단되어 "#"을 기호를 채택한 것이다.

<표 2> 키보드 상의 특수문자들에 대한 이스케이프 문자 정의

순번	키보드 상의 특수문자들	Escape문자 정의	비고
1	" (34, 0x22)	#d	
2	* (42, 0x2A)	#e	
3	/ (47, 0x2F)	#s	
4	: (58, 0x3A)	#c	
5	< (60, 0x3C)	#l	
6	> (62, 0x3E)	#r	
7	? (63 0x3F)	#q	
8	\ (92, 0x5C)	#b	
9	| (124, 0x7C)	#v	
10	# (35, 0x23)	##	
11	. (dot, 46, 0x2E)	#y	숨길메시지의 끝 문자
12	" "(space, 32, x20)	#z	

알고리즘 1 - 키보드 입력문자의 Escape변환

```
1  EOM=getMessageLength()
2  if EOM ≤4 then
3  |  if checkSpecialName() = TRUE then
4  |    doForbiddenFileName()
5  end
6  i=0
7  while i<EOM do
8  |  ch=getCharFromMessage(i)
9  |  if  ch≤0xFF then          //ASCII
10 |  |  if  (ch≤0x1F )AND
11 |  |  |  (ch≥0x80 AND ch≤0xA0 ) then
12 |  |  |   doForbiddenChar()
13 |  |   end
14 |  |  if ch = ‘ “ ’       then  ch←‘#d’
15 |  |  |  else if ch = ‘ * ’ then  ch←‘#e’
16 |  |  |  else if ch = ‘ / ’ then  ch←‘#s’
17 |  |  |  else if ch = ‘ : ’ then  ch←‘#c’
18 |  |  |  else if ch = ‘ < ’ then  ch←‘#l’
19 |  |  |  else if ch = ‘ > ’ then  ch←‘#r’
20 |  |  |  else if ch = ‘ ? ’ then  ch←‘#q’
21 |  |  |  else if ch = ‘ \ ’ then  ch←‘#b’
22 |  |  |  else if ch = ‘ | ’ then  ch←‘#v’
23 |  |  |  else if ch = ‘ # ’ then  ch←‘##’
24 |  |  end
25 |  end
26 |   i=i+1
27 end
28 if i = EOM then
29 |  if ch = ‘  ’      then ch ← ‘#y’
30 |  else if ch = ‘. ’ then ch ← ‘#z’
31 end
```

5.2 의사코드 알고리즘

다음의 알고리즘 1은 표 2의 문자들이 입력되는 경우를 이스케이프 문자로 처리하는 과정과 아스키 코드 중 키보드에 없는 문자와 확장아스키 코드 영역의 문자들, 그리고 파일명의 끝 문자로 사용할 수 없는 경우에 대한 알고리즘을 의사코드(pseudo code)형식으로 표현한 것이다. 이 부분은 3.1절의 알고리즘의 2단계에 해당한다.

알고리즘에서 1번 행은 입력된 메시지의 길이를 구하는 함수이다. 2~6번 행은 표 1의 5그룹을 처리한 부분이다. 처리하려는 메시지의 문자의 수가 4이하일 때 예약된 특수 기능의 파일명과 같은 문자열이라면 메시지로 사용할 수 없는 경우라서 에러처리 기능을 갖춘 doForbiddenFileName()함수에 의해서 처리된다.

7번~27번 행의 while문은 전체 메시지에 들어 있는 9개의 특수문자를 처리하는 부분이다. 14번 행에서부터 23번 행까지의 문장은 파일명에 사용할 수 없는 문자(표 1의 1번 그룹)이스케이프 문자로 처리하는 과정이다. 10번~13번 행의 두 개의 조건은 키보드로 입력할 수 없고 파일명에 사용할 수 없는 아스키 코드에 대한 처리 부분이다. 표 1의 2, 3번 그룹에 해당하는 문자를 처리하는 과정이다.

28번~31번 행의 문장은 표 1의 5번 그룹을 처리하는 과정이다. 메시지의 맨 끝 문자로 공백 문자나 마침표 문자가 할당되면 이스케이프 문자로 각각 “#y”와 “#z”로 변환하는 과정이다.

VI. 적용 사례

6.1 프로그램 개발환경

제안된 방법은 다음의 환경에서 프로그램이 제작되었다.

개발환경:

OS : Windows 7 Ultimate K Service Pack 1

개발도구 : Visual Studio 2013

개발언어 : C/C++, MFC 클래스

어플리케이션 타입 : Windows 다이얼로그 프로그램

실행환경:

디스크 포맷 : NTFS v3.1

저장매체 : 외장 usb 드라이브

저장공간 : 1TB

작업 디렉토리 : HideDir

위장 파일 : disguised-??.txt

고정 파일 : $fixedFile.txt

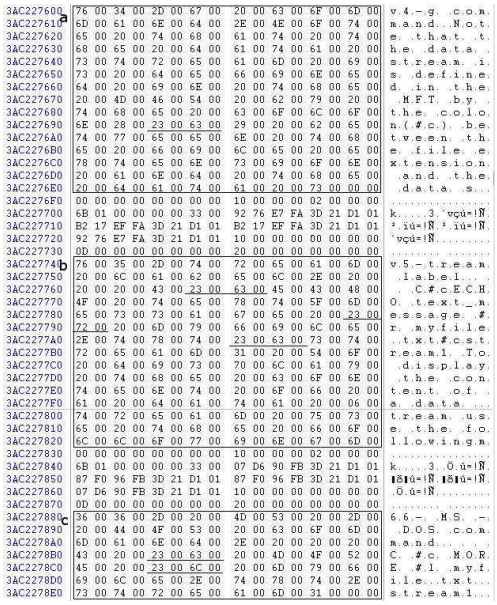
6.2 적용 사례 1

"NTFS Data Stream" - Of interest to the computing investigator and forensic examiner are multiple data streams. Data can be appended to existing files when you are examining a disk. Data stream can be obscure valuable evidentiary data, either intentionally or by coincidence.
In NTFS, a data stream becomes an additional data attribute of a file. From a Windows NT, 2000, or XP DOS shell, you can create a data stream by using the following command. Note that the data stream is defined in the MFT by the colon (:) between the file extension and the data stream label.
C:\ECHO text_message > myfile.txt:stream1
To display the content of a data stream use the following MS-DOS command.
C:\MORE < myfile.txt.stream1

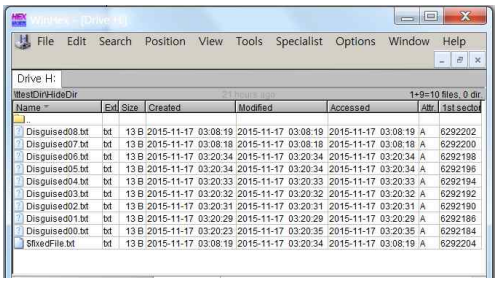
<그림 2> 사례1: 숨길데이터-일반텍스트에 특수문자가 포함되어 있는 경우

그림 2는 숨길 데이터이다. 데이터의 전체의 길이는 공백을 포함하여 710자로 구성되어 있다. 한 블록의 크기는 116자의 크기로 설정한다. 머릿번호는 2문자로 구성된다. 전체 7개의 블록으로 나뉘며 블록 크기의 배수에 맞지 않는 끝 블록은 빈 영역을 채우지 않고 문자의 실제 길이만 사용한다.

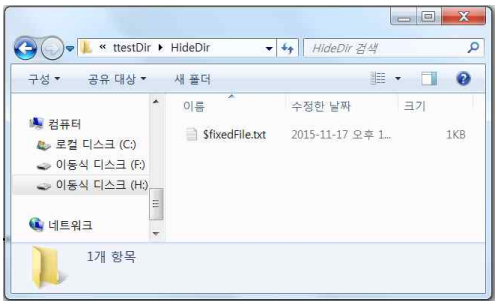
그림 3은 숨긴 데이터들의 일부를 나타낸 것이다. 변환된 문자에 대한 설명을 위하여 앞쪽에 숨겨진 "0-"~"3-"데이터 부분 생략하고 "4-"~"6-"부분만 사각형으로 표시하였다. "a"부분의 "23 00 63 00"은



<그림 3> 사례1: 숨긴 데이터-특수문자들의 변환



<그림 4> 사례1: 위장 파일들



<그림 5> 사례1: 작업 마친 후의 디렉토리

"#c"-(:)을 변환한 것이다. NTFS의 ASCII코드는 2바이트 유니코드를 사용하기 때문에 "23 00"과 "63 00"

으로 리틀엔디언(little endian)방식의 2바이트로 저장된다.

"b"부분에서 밑줄을 표시한 문자 "23 00 63 00"은 "#c"-(:), "23 00 72 00"은 "#r"-(>), "23 00 63 00"은 "#c"-(:)을 변환한 것이다.

"c"부분은 앞의 두 메시지에 비하여 길이가 짧다. 맨 끝 메시지이며 길이가 블록의 배수가 아니기 때문에 짧게 저장된 것이다. 두 개의 변환문자가 저장되었는데 하나는 "23 00 63 00"은 "#c"-(:)이고 다른 하나는 "23 00 6C 00"은 "#l"-(<)이 변환된 것이다.

그림 4는 삭제된 위장파일들을 나타낸 것이다. 디스크 툴에 의하여 삭제된 목록을 찾아서 표시한 것이다. 이 목록에서 알 수 있듯이 숨길 데이터로 파일명을 만들었던 것들은 보이지 않고 위장파일로만 표시e된다. 그림 5에는 고정파일만 남아있고 작업에 사용된 모든 파일들은 삭제 후의 디렉토리의 모습이다. 데이터로 사용된 파일들과 위장파일들은 모두 삭제된 후라서 디렉토리 목록에는 나타나지 않는다.

6.3 적용 사례 2

사례 2에서 다루는 데이터는 C/C++ 프로그램 코드이다(그림 6). 파일명에 사용할 수 없는 특수문자들이 이 코드안에는 다양하게 분포하고 있어서 본 논문에서 제안한 방법을 적용하기에 최선의 대상이다.

전체 문자는 공백문자를 포함하여 338자이다. 블록의 크기는 84문자 크기이다. 머릿번호는 2문자로 구성된다. 전체 블록의 수는 5개이다. 블록의 크기를 작게 설정한 것은 그림 7에 여러 블록이 표시되도록 의도적으로 설정한 것이다. 그림 7에서 "a"는 고정파일의 인덱스 엔트리 정보를 나타내고 있다. "b"는 마지막에 삭제된 위장파일의 인덱스 엔트리 흔적이다. 이 파일의 영향으로 숨긴 데이터의 첫 번째 데이터의 앞에서 일부분이 위장파일이 덮어쓰게 된다. 그러므로

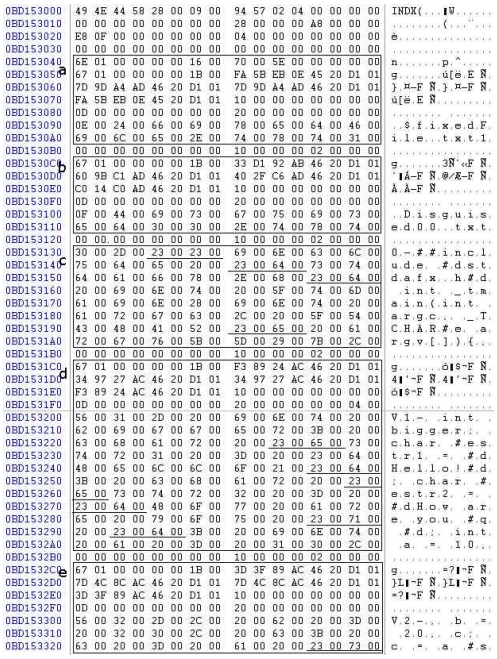
```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int bigger;
    char *str1 = "Hello!";
    char *str2 = "How are you?";

    int a = 10, b = 20, c;
    c = a / b;
    printf("a/b=%d₩n", c);

    bigger = (str1 > str2) ? 0 : 1;
    switch (bigger){
        case 0: printf("%s > %s₩n", str1, str2);
            break;
        case 1: printf("%s < %s₩n", str1, str2);
            break;
    }

    return 0;
}
```

<그림 6> 사례2: 숨길데이터-프로그램 코드

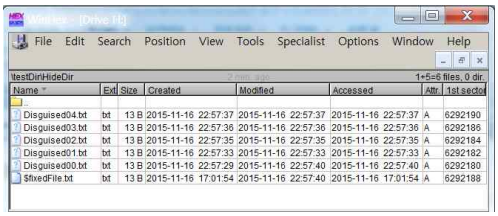
감출 데이터에는 위장파일의 파일명 길이만큼 여분을 두어야 한다.

"c"는 첫 번째 숨긴 데이터, "d"는 두 번째 데이터, "e"는 세 번째 데이터이다. 지면 부족으로 그림에는 3개의 데이터만 표시되었다. 각 데이터에는 파일명으로 사용될 수 없는 문자가 들어 있는데 각 부분에 밑줄로 표시된 부분이 본 논문에서 제안한 방법으로 적용된 이스케이프 문자로 변환된 문자들을 나타낸다. "c"부분의 첫째줄에서 "23 00 23 00"은 "##" 두 문자의 변환이다. 입력데이터의 "#incldue"의 "#"부분을 변환한 것이다. 둘째 줄과 셋째 줄의 "23 00 64 00"은 "#d"를 나타낸다. 이것은 겹따옴표(")의 변환이다. 일곱째 줄의 "23 00 65 00"는 "#e"의 변환이고 "_TCHAR*"의 "*" 부분을 변환한 것이다. "d"부분과 "e"부분의 밑줄 부분들도 "c"부분에서의 방법과 유사하게 방법으로 변환된 것이다.

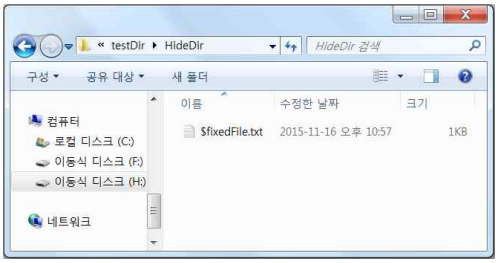
그림 8은 위장파일이 삭제된 흔적을 나타낸다. WinHex 프로그램을 사용하여 삭제된 파일을 표시한 것이다. 숨긴 데이터 파일은 삭제된 파일의 목록에서 찾을 수 없다. 그림 9는 데이터 숨김을 마친 후에 디



<그림 7> 사례2: 숨긴 데이터-특수문자들의 변환



<그림 8> 사례2: 위장 파일들



<그림 9> 사례2: 작업 마친 후의 디렉토리

렉토리 안의 고정파일 $fixedFile.txt파일을 나타낸 것이다. 작업에 관련된 다른 파일들은 모두 삭제되어 보이지 않고 유일하게 이 고정파일만 남게 된다.

Ⅶ. 결론

이 연구에서는 윈도우즈의 NTFS 파일시스템에서 디렉토리 인덱스 레코드에 데이터를 숨기기 위한 기법에서 파일명에 사용할 수 없는 문자 문제에 대한 해결 방법을 제안하였다. 데이터 숨김 기법을 구현하는 과정에서 DOS시절부터 사용하던 파일명 규칙이 Windows에서도 적용되는데 키보드로 입력이 가능한 문자들 중 몇 문자들에 특별한 기능을 부여하고 있어서 파일명으로는 사용할 수 없도록 정하고 있다는데에 문제가 있음을 발견하여 이 논문에서는 그 문제에 대한 해결방안으로 5개 그룹으로 분류된 파일명으로 사용불가능한 문자에 대한 처리방안을 키보드 입력 문자의 전처리 알고리즘을 제시하였다.

이 연구에서 제안한 방법은 디렉토리 인덱스 레코드에 데이터를 숨기는 방법의 기능향상을 위한 것이다. 문자들을 키보드로 입력하는 방법을 전제로 구현된 방법이다. 이 방법과 더불어 파일에서 바이너리 형태의 데이터를 처리할 수 있도록 기능이 확장되면 더 많은 활용이 가능할 것이다.

이 연구에서의 방법으로 안티포렌식 도구의 개발하려는 목적은 일반 포렌식 도구의 기능을 향상시키기 위함이다. 의도적으로 숨겨놓은 데이터에 대해서 단순히 디스크에 대한 키워드 검색만으로 포렌식을 수행할 수 있을 것이라는 기대는 하지 않도록 해야 하며 고급 기능이 구현된 포렌식 도구들이 개발되어야 한다.

참고문헌

[1] Wikipedia. org, “NTFS - Features - Scalability,” http://en. wikipedia. org/wiki/NTFS#Features

[2] Microsoft TechNet, “How NTFS Works,” https://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx.

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

[4] William Ballenthin, “NTFS INDX Attribute Parsing,” http://www.williballenthin.com/forensics/indx/index.html.

[5] Chad Tilbury, “NTFS $I30 Index Attributes: Evidence of Deleted and Overwritten Files,” SANS Digital Forensics and Incident Response Blog, http://digital-forensics.sans.org.

[6] William Ballenthin and Jeff Hamm, “Incident Response with NTFS INDX Buffers - Parts 1, 2, 3 and 4,” https://www.mandiant.com/blog/author/willi-ballenthin/

[7] 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.

[8] 조규상, “타임스탬프 변화패턴을 근거로 한 평가함수에 의한 디지털 포렌식 방법,” 디지털산업정보학회 논문지, 10권, 2호, 2014, pp.91-105.

[9] 조규상, “Windows 파일시스템의 디렉토리에 대한 디지털 포렌식 분석,” 디지털산업정보학회 논문지, 제11권, 제2호, 2015, pp. 73-90.

[10] Gyu-Sang Cho, “NTFS Directory Index Analysis for Computer Forensics,” IMIS 2015(the 9-th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing), July 8th-10th, Blumenau Brazil, 2015.

[11] 조규상, “새로운 NTFS 디렉토리 인덱스 안티포렌식 기법,” 한국정보전자통신기술학회논문지, 8권, 4호, 2015, pp. 327-337.

[12] Microsoft MSDN, “Naming Files, Paths, and Namespaces”, https://msdn.microsoft.com/en-us/library/aa365247

■ 저자소개 ■

조 규 상
Cho Gyusang

1996년 3월~현재
동양대학교 컴퓨터정보전학과 교수
2010년 9월~2011년 8월
미국 Purdue대학교, Dept. of Computer Information Technology, Cyber Forensic Lab, Visiting scholar
1997년 2월 한양대학교 전자공학과 (공학박사)

관심분야 : 디지털 포렌식, 시스템보안
E-mail : cho@dyu.ac.kr

논문접수일: 2015년 10월 29일
수 정 일: 2015년 월 일
게재확정일: 2015년 12월 4일