유니코드 변환이 적용된 NTFS 인덱스 레코드에 데이터를 숨기기 위한 안티포렌식 기법★

조규상*

요 약

윈도우즈 NTFS 파일시스템에서 인덱스 레코드에 데이터를 숨기기 위한 기법은 파일명을 이용하여 메시지를 숨기는 방법이다. 윈도우즈 NTFS의 파일명 규칙에서 일부 ASCII 문자는 파일명으로 사용할 수 없는 문제가 있다. 영문과 함께 한글, 기호 문자가 함께 입력이 될 때와 바이너리 형태의 데이터들이 입력될 때 인덱스 레코드에 데이터 숨기기 방법 수행 시에 파일생성 에러 문제가 발생하는 것을 해결하기 위한 방법으로 유니코드의 특정 영역으로 변환하는 방법을 제안한다. 에러가 발생하는 문자들을 한글과 영문 영역이 아닌 유니코드로 변환하고. 바이너리 형태의 데이터인 경우는 확장 유니코드 영역과 아스키 코드의 영역이 아닌 유니코드의 영역으로 256개의 코드 전체를 변환하는 방식을 적용한다. 영문과 함께 한글이 사용된 경우에 제안한 방식이 적용된 사례의 결과를 보이고. 바이너리의 경우는 PNG이미지 파일의 바이너리 코드를 유니코드로 변환한 사례를 통해서 제안한 방법이 타당함을 보인다.

An Anti-Forensic Technique for Hiding Data in NTFS Index Record with a Unicode Transformation

Gyu-Sang Cho*

ABSTRACT

In an "NTFS Index Record Data Hiding" method messages are hidden by using file names. Windows NTFS file naming convention has some forbidden ASCII characters for a file name. When inputting Hangul with the Roman alphabet, if the forbidden characters for the file name and binary data are used, the codes are convert to a designated unicode point to avoid a file creation error due to unsuitable characters. In this paper, the problem of a file creation error due to non-admittable characters for the file name is fixed, which is used in the index record data hiding method. Using Hangul with Roman alphabet the characters cause a file creation error are converted to an arbitrary unicode point except Hangul and Roman alphabet area. When it comes to binary data, all 256 codes are converted to designated unicode area except an extended unicode(surrogate pairs) and ASCII code area. The results of the two cases, i.e. the Hangul with Roman alphabet case and the binary case, show the applicability of the proposed method.

Key words : Data Hiding, Directory Index, Digital Forensics, Windows NTFS, B-tree

접수일(2015년 12월 7일), 수정일(1차: 2015년 12월 26일)
게재확정일(2015년 12월 29일)

*동양대학교/컴퓨터정보전학과

★ 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2013R1A1A2064426)

1. 서 론

데이터를 감추기 위한 많은 방법들이 소개되었는데 그 중에 한 가지는 파일시스템의 구조를 이용하여 데이터를 감추는 방법이다. 이 방법은 다양한 파일시스템에 적용되지만 그 중에서 Windows와 Unix/Linux에 관련된 방법이 많이 소개되었다.

Raggo와 Hosmer의 저서[1]에서 운영체제에 데이터 감추기 기법에서 윈도우즈와 리눅스 운영체제에 관련된 데이터 감추기는 방법을 소개하고 있다. 윈도우즈의 방법은 ADS(Alternate Data Stream)과 Stealth ADS, Volume Shadowing방법을 소개하고 있고, 리눅스의 경우는 Linux Filename Trickery, Extended File System Data Hiding, TrueCrypt등 잘 알려진 방법을 소개하고 있다. Carvey의 저서[2]에서도 운영체제와 관련된 데이터 감추기 방법을 소개하고 있다. ADS, Registry, Office 문서, OLE structured storage에 데이터를 감추는 방법을 소개하고 있다.

데이터를 감추기 위한 오픈소스 도구들도 개발이 되어 실제로 사용되고 있다. Metasploits의 Slacker[3]는 NTFS file system에서 파일을 생성할 때 실제로 필요한 것보다 더 큰 영역을 할당하는 점을 이용하여 데이터를 숨기는 기능을 수행한다. Thompson등이 개발한 FragFS[4]는 NTFS MFT(Master File Table) 영역 안에 데이터를 숨기는 도구로 안티포렌식을 위한 도구로 알려져 있다. 잘 사용하지 않는 MFT엔트리에 16바이트 단위로 데이터를 저장하는 방식으로 구현되었다.

S. Piper등[5]은 Ext2/Ext3 파일시스템에서 안티포렌식을 하기 위해 숨겨진 데이터를 찾아내는 방법에 관한 연구를 수행하였다. 파일시스템의 예약된 영역에 숨겨진 데이터를 검색하는 방법을 제안하였다.

Huebner등[6]은 파일시스템 안에 숨겨진 데이터를 탐지하고 복구하는 방법들을 소개하였다. 이 연구에서는 파일시스템의 데이터구조의 특성만을 이용한 여러 가지 방식의 데이터 숨김 방식을 소개하고 있다. 파일을 숨기기 위하여 암호화나 난독화 등이 아닌 파일시스템의 구조만을 이용하는 방법을 다루고 있다. 소개된 방법들 중에서 메타 파일($BadClus 파일, $DATA 속성, $Boot 파일)의 구조를 이용하는 방법과 데이터 파일을 이용하는 방법(ADS, 디렉토리 $DATA속성, 추가된 클러스터) 그리고 디스크의 슬랙 공간(볼륨, 파일시스템, 파일의 슬랙)을 이용하는 데이터를 숨기는 방법들을 소개하고 있다.

최근 Cho의 연구[7]에서 이론적인 B-tree가 NTFS에 구현된 방식에 대한 조사를 수행하였고 파일의 수가 증가함에 따라 인덱스 엔트리가 확장되는 방법을 분석하였다. 다른 연구[8]에서는 디렉토리의 인덱스 레코드에 데이터를 감추는 새로운 방법을 제안하였다. 파일이 삭제될 때 인덱스 레코드 안의 인덱스 엔트리들이 재정렬하면서 배치되는 방식을 이용하여 인덱스 레코드에 데이터를 숨기는 방법이다.

이 연구에서는 앞서서 발표된 인덱스 레코드에 데이터를 감추는 방법에서 데이터를 저장할 때 파일명을 이용하는 방식을 사용하는데 파일명에 사용할 수 없는 문자들로 인해서 파일에러가 생기는 문제에 대한 개선된 방법을 제안한다. 이 방법은 기존의 연구[8]에서 파일명으로 사용할 수 없는 문자들, 아스키 문자 중 0x00-0x1F, 0x80~0xA0 코드와 9개 문자들을 사용할 때 파일생성 에러가 발생하여 메시지 숨김 처리를 할 수 없는 것을 단점을 개선하고 바이너리 형태의 데이터를 감추기 위해 유니코드의 특정영역으로 변환하여 파일명을 이용하여 감추는 방법을 제안한다.

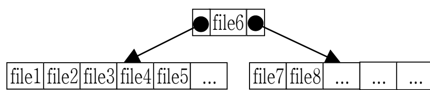
2. NTFS의 디렉토리 인덱스

2.1 NTFS의 B-tree의 동작 특성

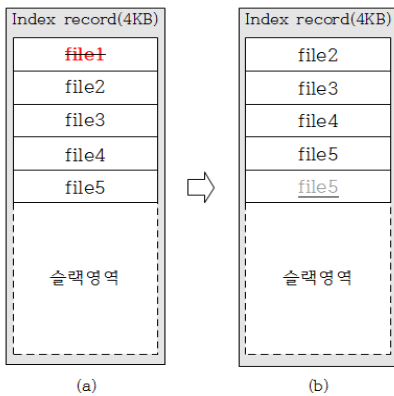
NTFS에서는 디렉토리 안에 들어 있는 파일들의 목록을 관리하기 위한 방법으로 B-tree를 사용한다. 자료구조의 이론에서 잘 알려진 이 방식은 여러 파일들을 다루는데 편리한 방법이고 파일의 수에 비하여 트리의 깊이가 상대적으로 낮게 유지되는 균형트리 방식이다. 파일에 대한 검색도 빠르게 수행할 수 있다. (그림 1)은 일반적인 B-tree의 간단한 예를 나타낸 것이다. 이 그림에서 왼쪽 자식들은 부모노드의 키 값보다 작은 값들로 구성되어 있고 오른쪽 자식들은 부모노드의 키 값보다 큰 값들로 구성된다.

NTFS에서는 B-tree의 노드를 인덱스 레코드라 부른다. 각 노드들 안에는 여러 개의 엔트리를 갖는다.

엔트리가 다 채워지면 두 개의 노드로 나뉘게 된다. NTFS에서 루트 노드 역할은 $INDEX_ROOT속성이 담당한다. 이것은 MFT 엔트리의 레지던트 속성이기 때문에 기록되는 목록의 수가 한정적이다. 이 공간의 크기는 대략 700바이트정도 사용이 가능하다. 8.3형식의 파일명인 경우는 보통 5~6개 정도밖에 기록되지 않는다. 공간이 부족하면 $INDEX_ALLOCATION에 들어있는 Run-length를 이용하여 외부 노드에 디렉토리의 인덱스 엔트리가 저장된다. 바로 외부 노드의 엔트리



(그림 1) B-tree의 예



(그림 2) 인덱스 레코드에서 목록의 삭제

저장 공간이 인덱스 레코드이다. 이 크기는 4KB이다. 인덱스 레코드 안에는 많은 디렉토리 목록들이 저장된다. 이것들은 항상 대소문자 구분없이 아스키 코드순으로 소팅되어 저장된다.

(그림 1)의 모든 노드들은 (그림 2)에서와 같은 인덱스 엔트리로 구성되는데 부모노드이건 리프 노드이건 상관없이 모두 4KB의 크기를 갖는다. 인덱스 엔트리 안에는 (그림 2)의 (a)와 같이 엔트리들이 소팅되어 저장된다. 만일에 첫 번째 엔트리 “file1”이 삭제된다면 두 번째 엔트리가 첫 번째 엔트리 자리로 이동하고 순차적으로 나머지 엔트리들도 같은 방식으로 전체의 엔트리가 재구성된다. 맨 끝 엔트리가 들어 있던 곳에는 흔적이 남게 된다. 이렇게 흔적이 남게 되는 것을 이용하여 이 연구에서 인덱스 레코드에 메시지를 숨기는 방법이 적용된다.

3. 인덱스 레코드에 데이터 숨기기

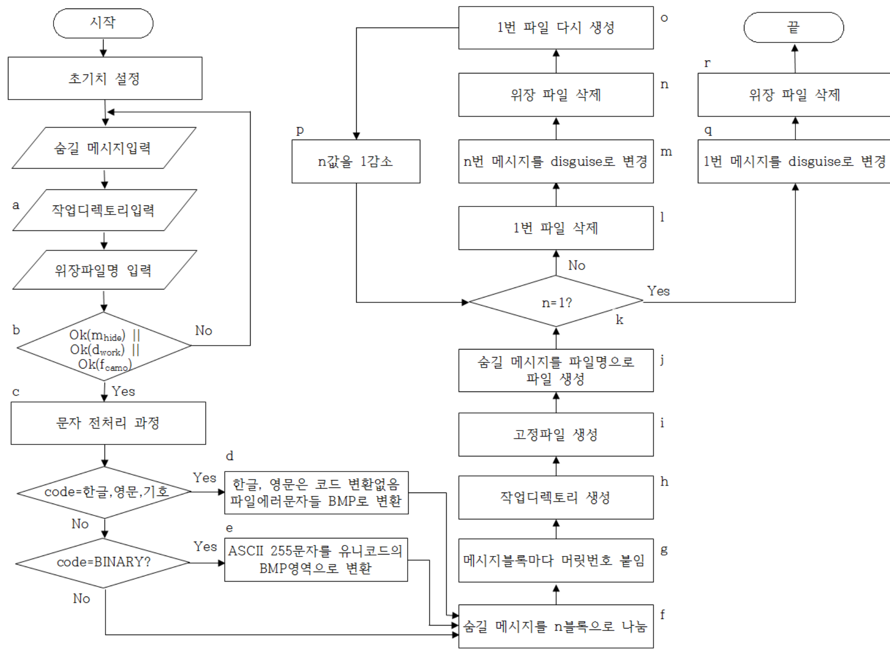
3.1 알고리즘 설명

이 알고리즘은 NTFS 인덱스 레코드에 데이터를 숨기기 위한 방법이다. 이 방법의 기초는 참고문헌[8]에서 두고 있다. 기존의 방법은 사용할 수 있는 문자의 제약이 있다. 사용 가능한 문자들은 키보드로 입력할 수 있는 문 자중에서 파일명에 사용할 수 없는 9개의 문자를 포함하여 ASCII 코드의 경우는 0x00~0x1F와 0x80-0x9F까지의 코드는 사용할 수 없다. 이런 제약을 극복하기 위한 방법으로 (그림 3)의 플로 차트와 알고리즘 1,2에 개선된 방법을 제안한다.

(그림 3)의 ”a”는 메시지, 작업 디렉토리명, 위장파일명을 입력하는 부분이다. “b”에서 입력이 제대로 이루어졌는지 확인하게 된다. “c”의 문자 전처리 과정에서는 마지막 메시지를 처리하고 난 후에 위장파일에 의하여 덮여지는 1번 메시지부분을 위하여 메시지의 선두부분에 빈 문자를 넣는 과정이다.

Windows NTFS에서는 UTF-16 엔코딩 방식이 적용된다. 문자를 16비트(2바이트)로 저장하게 된다. 문자체계는 유니코드를 사용한다. 그러므로 각 문자들은 유니코드 문자체계를 사용하면서 16비트로 파일시스템에 기록된다. “d”부분에서 입력된 문자의 코드가 한글, 영문, 기호문자에 해당한다면 코드는 파일명에 사용할 수 없는 문자들이 변환된다. “e”부분에서 만일 코드가 바이너리 코드에 해당한다면 메시지로 사용할 수 없는 문자가 포함되어 있기 때문에 특정 유니코드의 BMP영역으로 전체 코드를 변경한다.

”f”부분에서는 메시지를 n개의 블록으로 나누는 절차이다. ”g”부분에서는 각 블록의 선두에 머릿번호를 붙인다. 여러 블록으로 나뉜 메시지들의 순서가 변경



(그림 3) 인덱스 레코드에 메시지를 숨기기 위한 개선된 알고리즘

되지 않도록 하기 위함이다. "h"부분은 작업디렉토리를 생성하는 과정이다. 이 작업 디렉토리의 인덱스에 메시지 숨김을 한다. "i"부분은 고정파일을 생성하는 과정이다. 인덱스 레코드는 한번 할당되면 디렉토리의 파일의 수가 줄어들어도 동적으로 해제되지 않는 특성이 있다. 그러나 최소 디렉토리의 목록이 한 개가 유지되어야 할당한 모든 인덱스 레코드가 유지된다. 이것을 위해서 고정파일이 필요하다. "j"부분에 n 블록으로 나뉜 숨길 메시지들을 파일명 지정하여 파일들을 생성한다.

이전까지의 과정들에서 메시지를 숨기기 위한 준비를 한 것이다. 실질적인 숨기기 절차는 "l"부분부터 시작된다. n개의 메시지 블록 중에서 첫 번째 위치에 있는 1번 파일 삭제한다. 인덱스 레코드 내에서 n-1개의 목록들이 1번 파일의 인덱스 엔트리가 삭제된 자리를 2번 파일의 인덱스가 채워지고 그 후로 차례로 나머지 인덱스 엔트리들이 재정렬하게 된다. 마지막 자리에 있던 인덱스 엔트리가 한 자리 위로 이동하면서 그것의 흔적인 목록의 맨 끝에 남는다. "m"부분에서 n번 메시지 파일명을 위장파일명으로 변경한다. MFT 엔트리에 기록된 n번 메시지 파일명이 위장파일명으로 기록된다. 인덱스 엔트리에도 위장파일명으로 기록된다. 다음 단계 "n"에서 위장파일을 삭제한다. 파일명이 기록되는 두 군데(MFT 엔트리, 인덱스 엔트리) 모두 위장파일명이 기록되어 있다가 삭제된 위장 파일의 흔적만 남는다. "o"부분에서 1번 파일을 다시 생성하면서 한 절차를 마무리한다. "p"부분은 n-1번 메시지로 처리 메시지 번호를 조정하는 과정이다. 처리할 메시지의 번호가 1번에 도달하면 마지막 한 개의 남은 1번 메시지를 처리하기 위한 "q"와 "r"부분으로 이동한다. 1번 파일 한 개만 남으면 1번 파일을 삭제하는 과정과 1번 파일을 다시 생성하는 과정을 수행하지 않고 위장파일명으로 변경하고 삭제하는 과정만 수행하면서 전체 과정을 마친다.

인덱스 레코드 내에서는 항상 자동적으로 인덱스 엔트리들이 아스키 코드 순으로 첫 엔트리부터 마지막 엔트리까지 정렬된 상태를 유지하며 기록된다는 특징이 있다. 이런 특징으로 인해서 만일 중간의 어느 엔트리가 삭제된다면 그 후의 목록들이 차례로 지워진 자리를 채우면서 재정렬하게 된다.

3.2 파일명으로 금지된 문자들

키보드 상의 문자중에서 9개의 문자는 파일명에 들어가면 파일생성 에러를 일으킨다. 9개의 문자는 “ (34, double quote), * (42, esterisk), / (47, slash), : (58, colon), < (60, greater than), > (62, less than), ? (63 question mark), \ (92, back slash), | (124, vertical bar)는 파일명에 사용할 수 없다[9].

ASCII 코드 집합 중에서 0(0x00)~31(0x1F)까지의 문자들은 출력할 수 없는 문자들이다. 주로 주변기기나 프린터를 제어하기 위해서 만들어진 코드들이다. 이 문자들이 포함된 파일명 생성에러가 발생한다[9].

확장 ASCII 코드 중에서 128(0x80)~160(0xA0) 사이의 문자들(표 1의 2번 행)도 파일명에 사용할 수 없다[9]. 이 문자들은 다른 그룹의 문자들과 함께 파일명에 사용되는 경우에는 파일에러를 일으키지는 않고 파일명에서 이 문자들만 제거되는 특징이 있다.

<표 1> ASCII와 한글의 유니코드에서 사용영역

순번	코드 종류		코드 범위
1	ASCII 코드 영역	Basic Latin	0000-007F
2		C1 Control	0080-009F
3		Latin-1 Supplement	00A0-00FF
4	한글 영역	한글자모(256개)	1100-11FF
5		호환용한글자모(96개)	3130-318F
6		한글자모확장(32)	A960-A97F
7		한글소리마디(11184)	AC00-D7AF
8		한글자모확장(80)	D7B0 - D7FF
9	확장 unicode	Surrogate pair(High)	D800 - DBFF
10		Surrogate pair(Low)	DC00 - DFFF

4. 안티포렌식 데이터 숨김 기법을 위한 파일명 유니코드 문자변환

4.1 한글/영문자를 사용하는 경우를 위한 방법 제안 : BMP영역 코드매핑

UTF-16(16-bit Unicode Transformation Format)은 유니코드 문자체계에서 인코딩하기 위한 여러 방식 중의 하나이다. 유니코드를 표현하기 위해서 16비트를 사용하는 방식이며 주로 사용되는 기본 다국어 평면(BMP, Basic multilingual plane)에 속하는 문자들은 코드값 그대로 16비트로 인코딩된다. 그 이상의 문자들의 경우는 확장 유니코드(Surrogate Pair) 방식의 32비트로 인코딩이 된다[10].

이 절에서는 영문자와 한글을 함께 사용하는 경우에 대한 처리방법을 제안한다. Basic Latin 영역에 포함되어 있는 출력할 수 없는 문자들과 0(0x00)~31(0x1F), C1 Control 문자들 128(0x80) ~ 160(0xA0), 그리고 9개의 파일명 사용금지 문자들(“,*, /, :, <, >, ?, \, |)에 대한 처리 방법이다.

알고리즘 1에서 1번 행의 getMsgLen()은 전체 입력된 메시지의 글자의 수를 계산하는 함수이다. 2번 행과 14번 행 사이의 while문은 입력된 문자들의 코드의 범위에 따라 유니코드 내에서의 변환을 수행하는 함수들이다. 4번 행의 if(c≤0x1F)는 ASCII의 Basic Latin 영역 중 출력할 수 없는 문자들을 의미한다. 이 문자들의 경우에 chgCode(BMPbasis1, c)함수에 의하여 BMPbasis1에는 BMP영역 중에서 한글과 ASCII의 영역이 아닌 곳의 코드 값으로 임의로 지정한 위치값을 사용한다.

6번 행의 if(c≥0x80 AND c≤0xA0)는 C1 Control 문자를 의미한다. 이 문자들이 사용되면 파일 생성 에러를 일으키지는 않지만 파일명 내에 해당 문자들이 기록이 되지 않는 특성을 보인다. chgCode(BMPbasis2, c)함수의 인수 BMPbasis2의 코드 값에 따라 코드가 변환된다. 이 코드 값도 한글과 ASCII영역이 아닌 곳이면 어떤 선택도 가능하다.

8번 행의 if(compare9char(c))는 키보드에서 입력가능한 문자 중에서 파일명에 사용하면 에러를 일으키는 9개 문자여부를 체크하는 함수이다. chgCode(BMPbasis3, c)의 BMPbasis3 인수값의 지정에 따라 코드가 변환된다. 9개의 코드의 순서에 따라 차례로 BMPbasis를 기준으로 각문자의 순서 값에 따라 코드가 부여된다. 즉, BMPbasis+{(″)→0, (*)→1, (/)→2,(:)→3, (<)→4, (>)→5, (?)→6, (\)→7, (|)→8}로 변환된다.

10번 행의 else문의 실행 함수 doAlphaHangul(c)는 한글과 알파벳대소문자와 숫자를 처리하는 함수이다.

원래의 코드를 변환 없이 그대로 사용한다.

4.2 바이너리 파일의 코드처리 방법 제안

이 절에서는 바이너리 코드 형태의 파일을 처리하기 위한 방법의 제안이다. 일반적인 키보드 입력가능한 문자의 처리 방법과는 달리 바이너리 파일의 내용들은 아스키코드의 0번에서부터 255번까지 고루 사용된다는 특징이 있다. 실행파일 형태나 암호화된 형태 등을 모두 이 방법으로 처리할 수 있다

<알고리즘 1>유니코드 한글/영문 입력문자의 변환

```
1  MsgSize=getMsgLen()
2  while i<MsgSize do
3    c=getcharMsg(i)
4    if(c≤0x1F)then
5      | chgCode(BMPbasis1, c)
6    else if(c≥0x80 AND c≤0xA0 ) then
7      | chgCode(BMPbasis2, c)
8    else if(compare9char(c)) then
9      | chgCode(BMPbasis3, c)
10   else then
11     | doAlphaHangul(c)
12   end
13   i=i+1
14 end
```

<알고리즘 2>바이너리 문자의 변환

```
1  MsgSize=getMsgLen()
2  setCodeBasis(basis)
3  if(!isSurrogate(basis))then
4    | doSetCodeBasis(basis)
5  else then
6    | redoSetCodeBasis(basis)
7  end
8  while i ≤ MsgSize do
9    ch=getcharMsg(i)
10   c = ch + getCodeBasis()
11   if(isSurrogate(c) OR isASCII(c)) then
12     | showMsgOverBMPError()
13   end
14   i=i+1
15 end
```

이 방법은 아스키 문자 하나에 해당하는 것을 유니코드의 한 문자로 치환하는 것이다. Windows의 NTFS 파일시스템에서는 UTF-16 인코딩 방식을 사용하고 있어서 2바이트짜리 유니코드로 변환하는 것이 적절한 선택이다. 유니코드의 기본 다국어 평면 (BMP, Basic multilingual plane)영역 중에서 한글과 Latin영역을 제외한 영역으로 대응되는 코드포인트를 지정한다. 가능한 경우 수는 매우 많으므로 다양한 선택을 할 수 있다. 숨길 메시지는 디렉토리 인덱스의 슬랙영역에 사용될 것이기 때문에 화면에 디스플레이할 필요성이 없다. 그러므로 해당문자에 대한 폰트의 지원 여부를 확인할 필요가 없다.

1번 행에서 입력된 메시지의 문자의 개수를 구한다. 2번 행의 setCodeBasis(basis)는 인수로 지정한 코드를 기준으로 설정하는데 사용하는 함수이다. 이 값이 지정되면 바이너리 코드를 지정한 코드값을 기준으로 원래의 바이너리 값을 더하여 변환하게 된다. 3번 행의 isSurrogate(basis)함수는 basis코드로 변환하려는 값이 확장유니코드 영역의 코드인지를 체크하는 기능을 수행한다. 확장유니코드의 경우는 상위 16비트와 하위 16비트의 surrogate pair를 사용하여 32비트 코드를 표기하는 방식이다. 설정한 basis 코드값이 확장유니코드 영역의 것이 아니면 doSetCodeBasis(basis)함수에 의하여 basis코드의 사용을 허가한다. 만일 확장유니코드의 영역이라면 redoSetCodeBasis(basis) 함수에 의하여 basis코드를 새로 정한다. 이 연구에서는 BMP영역에서만 코드 변환을 사용하기 때문에 확장 유니코드 영역의 코드의 사용을 막기 위해 사용하는 방법이다.

8번 행에서 14번 행까지의 문장은 입력된 문자를 BMP영역 중의 어느 부분으로 코드를 변환하는 과정이다. 10번 행의 문장에서 getCodeBasis()로 basis코드값을 구하고 변환하려는 문자에 더하여 코드를 생성한다. 11~13번 행에 의하여 확장코드영역으로 변환하는 것과 ASCII영역으로 변환하는 것을 방지한다.

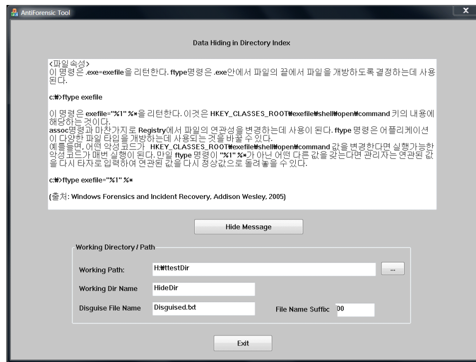
5. 사례 적용

5.1 한글, 영문, 기호문자의 입력 사례

영문과 한글, 기호문자가 혼재된 문장으로 구성된 데이터를 키보드로 입력하여 인덱스 레코드에 데이터 숨기기를 하려고 한다. 이 사례는 다음의 시스템 환경 조건에서 수행된 것이다.

OS : Windows 7 Ultimate K Service Pack 1
개발도구 : Visual Studio 2013
개발언어 : C/C++, MFC 클래스
어플리케이션 타입 : Windows dialog based
NTFS버전 : NTFS v3.1
저장매체 : 1TB 외장 usb 드라이브
작업 디렉토리 : h:\ttest\HideDir
위장 파일 : Disguised-__.txt
고정 파일 : $fixedfile.txt

이 사례에서 chgCode(BMPbasis1, c)에 의해서 변환하기 위해서 사용되는 변환 기준값은 U+0x2460로 정한다. 이 값으로 시작하는 유니코드문자들은 ①②…⑳등의 원문자와 ⑴⑵…⒇등의 괄호문자 코드이다. BMPbasis2 코드의 값은 U+0x0x24b0로 정하였으나 예제의 문자들은 이 영역에 해당하는 문자들이 존재하지 않아서 이 코드들은 결과에 반영되지 않았다. 파일명에 사용할 수 없는 9개의 문자들은 (")→U+0x2660, (*)→U+0x2661, (/)→U+0x2662,(:)→U+0x2663, (<)→U+0x2664, (>)→U+0x2665, (?)→U+0x2666, (\)→U+0x2667, (|)→U+0x2668로 변환된다. 문자변환에 선택된 코드들은 잘 사용되지 않는 영역으로 선택한다. 한글은 변환 없이 원래의 유니코드를 그대로 사용한다.



(그림 4) 한글, 영문, 특수문자 혼합 입력문

전체 문자의 개수는 598개이다. 블록의 크기는 117로 정하였다. 그러므로 메시지의 개수는 6개가 된다. 블록의 크기는 117로 작게 설정한 이유는 스크린 샷에 여러 메시지를 담기 위한 조치이다.

(그림 3)의 알고리즘의 데이터 숨김 알고리즘과 알고리즘 1의 유니코드 한글/영문 입력문자의 변환문자 변환을 적용한 결과를 (그림 5)에 나타냈다. (그림 5)에서 "a"는 인덱스 레코드의 헤더이다. "b"는 고정파일 $fixedFile.txt의 인덱스 엔트리 정보이다. "c"는 마지막 메시지를 숨김 처리할 때 사용된 Disguised00.txt 파일의 삭제된 후의 흔적이다. 첫 번째 메시지의 앞부분을 덮어 쓰고 있어서 겹치는 이 부분의 문자들을 입력된 전체 문자에서 고려하여야 한다. 이 인덱스 엔트리들이 삭제되고 난 후의 슬랙 영역이다. 이 부분에 들어 있는 메시지들의 흔적은 MFT엔트리의 정보를 통해서는 접근 할 수 없는 부분이다. 여기에 기록된 내용들은 숨겨진 메시지들이다.

```
0BD1A5000 a 49 4E 44 58 28 00 09 00  43 45 00 06 00 00 00 00
0BD1A5010   00 00 00 00 00 00 00 00  28 00 00 00 A8 00 00 00
0BD1A5020   E8 0F 00 00 00 00 00 00  05 00 00 00 85 C7 00 00
0BD1A5030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0BD1A5040 b AC 01 00 00 00 00 14 00  70 00 5E 00 00 00 00 00
0BD1A5050   6B 01 00 00 00 00 36 00  C2 35 D3 9E 56 2F D1 01
0BD1A5060   86 F3 10 30 7B 2F D1 01  86 F3 10 30 7B 2F D1 01
0BD1A5070   C2 35 D3 9E 56 2F D1 01  10 00 00 00 00 00 00 00
0BD1A5080   0D 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00
0BD1A5090   0E 00 24 00 66 00 69 00  78 00 65 00 64 00 46 00
0BD1A50A0   69 00 6C 00 65 00 2E 00  74 00 78 00 74 00 31 00
0BD1A50B0   00 00 00 00 00 00 00 00  10 00 00 00 02 00 00 00
0BD1A50C0 c 6B 01 00 00 00 00 36 00  D1 86 B6 2A 7B 2F D1 01
0BD1A50D0   AC FD 45 30 7B 2F D1 01  CD 21 4D 30 7B 2F D1 01
0BD1A50E0   4C 9C 43 30 7B 2F D1 01  10 00 00 00 00 00 00 00
0BD1A50F0   0D 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00
0BD1A5100   0F 00 44 00 69 00 73 00  67 00 75 00 69 00 73 00
0BD1A5110   65 00 64 00 30 00 30 00  2E 00 74 00 78 00 74 00
0BD1A5120   00 00 00 00 00 00 00 00  10 00 00 00 02 00 00 00
0BD1A5130 d 30 00 2D 00 64 26 0C D3  7C C7 8D C1 31 C1 65 26
0BD1A5140   6A 24 74 C7 20 00 85 BA  39 B8 40 C7 2E 00 65 00
0BD1A5150   78 00 65 00 20 00 3D 00  20 00 65 00 78 00 65 00
0BD1A5160   66 00 69 00 6C 00 65 00  44 C7 20 00 AC B9 34 D1
0BD1A5170   5C D5 E4 B2 2E 00 20 00  66 00 74 00 79 00 70 00
0BD1A5180   65 00 85 BA 39 B8 40 C7  20 00 2E 00 65 00 78 00
0BD1A5190   65 00 48 C5 D0 C5 1C C1  20 00 0C D3 7C C7 58 C7
0BD1A51A0   20 00 5D B0 D0 C5 1C C1  20 00 0C D3 7C C7 44 C7
0BD1A51B0   20 00 1C AC 29 BC 58 D5  C4 B3 5D B8 20 00 B0 AC
0BD1A51C0   15 C8 58 D5 94 B2 70 B3  20 00 AC C0 A9 C6 1C B4
0BD1A51D0   E4 B2 2E 00 6A 24 6A 24  63 00 63 26 65 26 66 00
0BD1A51E0   74 00 79 00 70 00 65 00  20 00 65 00 78 00 65 00
0BD1A51F0   00 00 00 00 00 00 00 00  10 00 00 00 02 00 05 00
0BD1A5200   6B 01 00 00 00 00 36 00  69 74 7F 2B 7B 2F D1 01
0BD1A5210   A9 81 82 2B 7B 2F D1 01  A9 81 82 2B 7B 2F D1 01
0BD1A5220   69 74 7F 2B 7B 2F D1 01  10 00 00 00 00 00 00 00
0BD1A5230   0D 00 00 00 00 00 00 00  20 00 00 00 00 00 00 00
0BD1A5240 e 77 00 31 00 2D 00 66 00  69 00 6C 00 65 00 6A 24
0BD1A5250   6A 24 74 C7 20 00 85 BA  39 B8 40 C7 20 00 65 00
0BD1A5260   78 00 65 00 66 00 69 00  6C 00 65 00 20 00 3D 00
0BD1A5270   20 00 60 26 25 00 31 00  60 26 20 00 25 00 61 26
0BD1A5280   44 C7 20 00 AC B9 34 D1  5C D5 E4 B2 2E 00 20 00
0BD1A5290   74 C7 83 AC 40 C7 20 00  48 00 4B 00 45 00 59 00
0BD1A52A0   5F 00 43 00 4C 00 41 00  53 00 53 00 45 00 53 00
0BD1A52B0   5F 00 52 00 4F 00 4F 00  54 00 67 26 65 00 78 00
0BD1A52C0   65 00 66 00 69 00 6C 00  65 00 67 26 73 00 68 00
0BD1A52D0   65 00 6C 00 6C 00 67 26  6F 00 70 00 65 00 6E 00
0BD1A52E0   67 26 63 00 6F 00 6D 00  6D 00 61 00 6E 00 64 00
0BD1A52F0   20 00 A4 D0 58 C7 20 00  B4 B0 A9 C6 D0 C5 20 00
0BD1A5300   74 D5 F9 B2 58 D5 94 B2  20 00 83 AC 74 C7 E4 B2
```

(그림 5) 숨겨진 메시지의 앞 부분

"d"부분은 첫 번째 숨겨진 메시지이다. 각 코드들은 리틀엔디언 방식으로 저장되어 있다. 이 부분에 들어있는 7개의 변환된 코드들은 차례대로 (<)→U+0x2664, (>)→U+0x2665, (\n)→U+0x246A, (\n)→U+0x246A,(:)→U+0x2663, (\n)→U+0x246A, (:)→U+0x2663, (>)→U+0x2665이다.

"e"는 두 번째 메시지가 숨겨진 부분이다. 이 그림 내에서는 9개의 코드가 변환되었다. 차례대로 (\n)→U+0x246A, (\n)→U+0x246A, (")→U+0x2660, (")→U+0x2660, (*)→U+0x2661, (\)→U+0x2667, (\)→U+0x2667, (\)→U+0x2667, (\)→U+0x2667이 변환된 것이다.

5.2 바이너리 파일

이 사례에서는 바이너리 파일로 저장된 데이터를 디렉토리 인덱스에 숨김 처리를 하려고 한다. (그림 6)에 나타낸 크기가 675자로 구성된 PNG 그림파일이다. 사례 1과 마찬가지로 블록의 크기는 117로 정하였다. 메시지의 개수는 6개가 된다. 이 사례에서 setCodeBasis(basis)에 의하여 설정되는 변환코드 영역은 basis=0xAC00이다. 즉, 한글소리마디 영역으로 할당하였다. 바이너리 코드들은 0x00→U+0xAC00~0xFF→U+0xACFF 범위 내에서 변환된다.

(그림 6) 바이너리 데이터 입력(PNG 그림파일)

(그림 7)은 바이너리 데이터를 basis코드를 기준으로 변환한 후에 인덱스 레코드에 숨기기를 마친 후의 모습이다. 한글의 모든 문자는 파일명으로 사용할 수 있기 때문에 변환된 유니코드 문자의 의한 메시지 숨기기 알고리즘을 적용하는데 파일생성 에러 없이 처리할 수 있게 된다. (그림 7)의 "a"부분은 첫 번째 숨긴 메시지이고 "b"는 두 번째 숨긴 메시지이다. "c"는 맨 끝 숨긴 메시지를 나타낸 것이다. 중간의 숨긴 메시지들은 생략하였다. "a"부분에서 밑줄 친 0xAC89 코드는 ASCII의 "0x89", 0xAC50 코드는 0x50("P"), 0xAC4E 코드는 0x4E("N"), 0xAC47 코드는 0x47("G")가 각각 변환된 것이다. 그 후의 문자들도 모두 같은 방식으로 변환된 것이다. 바이너리 코드의 범위는 0x00~0xFF까지의 256종류이므로 변환된 코드는 0xAC00~0xACFF까지 문자범위 내에서만 변환된다.

5.3 제안한 방법의 평가

5.1과 5.2절에서 적용한 사례는 제작한 툴에 의해 실행된 결과이다. 이 도구를 사용하면 모두 성공적으로 수행되지만 반드시 다음의 조건을 전제로 한다.

1) 숨김 메시지의 길이는 경로의 문자 길이를 포함하여 255자를 넘지 않아야 한다.

2) 변환하려는 유니코드는 0x0000-0x00FF의 아스키 코드 제외한 영역이어야 한다.

3) 확장유니코드 영역 0xD800 - DBFF(Surrogate pair high)와 0xDC00 - DFFF(surrogate pair low)은 이론적으로 사용이 가능하지만 개발된 툴에서 변환된 코드의 부작용을 고려하지 않았다.

4) 숨길 메시지와 관련된 디렉토리 내에 제안한 방법과 상관없는 파일은 들어 있지 않아야 한다.

5) 고정파일명은 반드시 목록 중에서 가장 선두에 놓이는 이름이어야 한다.

(그림 7) 숨겨진 메시지(첫 부분과 끝부분)

본 연구에서 제안한 방법은 기존의 방식과는 다른 새로운 기법이 적용된 것이다. 참고문헌 [6]에 소개된 여러 가지 방법 중에서 슬랙 공간(볼륨, 파일시스템, 파일의 슬랙)을 이용하는 방법이 소개되었지만 인덱스 레코드를 이용하는 본 연구의 방법과는 다른 방법이다. 또한, 실제로 필요한 것보다 더 큰 파일을 만들어서 슬랙을 이용하는 방법인 Metasploits의 Slacker [3]과 NTFS의 MFT 영역 안에 데이터를 숨기는 도구인 Thompson등의 FragFS[4]도 제안한 방법과는 기본적으로 다른 방법이다.

6. 결 론

윈도우즈의 NTFS 파일시스템에서 인덱스 레코드에 데이터를 숨기기 위한 기법에서는 파일명을 이용하여 메시지를 숨긴다. 아스키 코드들 중의 일부 문자들은 파일명으로 사용할 수 없다는 문제점이 있다. 영문과 한글이 입력이 될 때 파일에 사용할 수 없는 문자가 입력되거나 바이너리 형태의 데이터(예를들면 암호화된 내용과 프로그램 실행코드)들에는 파일명으로 사용할 수 없는 문자들이 포함되어 있어서 데이터 자체를 변환 없이 온전한 형태로 메시지 파일명에 담아서 데이터 숨김을 할 수 없다.

이 연구에서의 제안 방법은 영문과 한글이 입력되는 경우는 에러가 발생하는 코드들을 부분적인 유니코드 중에서 한글과 영문이 아닌 다른 부분에 코드를 매핑하는 방식을 적용하였다. 바이너리 데이터들은 확장 유니코드 영역과 아스키 코드의 영역이 아닌 유니코드의 영역으로 코드 전체를 변환하는 방식을 적용하였다. 한글과 영문이 사용된 사례에 제안한 방식을 적용하여 수행된 결과를 보였고 바이너리의 경우는 PNG 그림 파일의 바이너리 코드를 유니코드로 변환한 결과를 통해서 제안한 방법이 타당함을 보였다.

UTF-16 인코딩은 2바이트 형태이므로 바이너리의 한 바이트를 2바이트로 변환하는 형태가 된다. 저장공간의 효율성을 고려한 문자변환 방식에 대한 고려의 필요성을 추후과제로 생각해볼 수 있다. 또한 현재 구현된 도구는 쓰기 기능만을 갖추고 있다. 읽기 기능과 변경 기능을 갖춘 도구를 개발하면 안티포렌식을 위한 데이터 감추기 도구로 활용도를 높일 수 있을 것이다. 이 연구에서 개발된 안티포렌식 기법이 기존의 포렌식 도구들이 디렉토리 인덱스에 감춰진 데이터를 좀 더 논리적이고 조직적이며 자동화된 방법으로 찾아 낼 수 있는 기능이 구현되기 위한 촉매로 작용될 수 있기를 바란다.

참고문헌

[1] Michael T. Raggo, Chet HosmerB, 'Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems', Syngress, 2013.

[2] H. Carvey, 'Windows Forensics and Incident Recovery', 2005.

[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, 2006.

[5] S. Piper et.al, "Detecting hidden data in EXT2/EXT3 file systems", Advances in Digital Forensics, pp. 245-256, 2006.

[6] E. Huebner, D. Bem and C. K. Wee, "Data hiding in the NTFS file system", Digital Investigation, Vol. 3, Issue 4, pp. 211-226, 2006.

[7] Gyu-Sang Cho, "NTFS Directory Index Analysis for Computer Forensics", IMIS 2015, Blumenau Brazil, 2015.

[8] 조규상, "새로운 NTFS 디렉토리 인덱스 안티포렌식 기법", 한국정보전자통신기술학회 논문지, 8권, 4호, pp.327-337, 2015.

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

[10] Microsoft MSDN, "Surrogates and Supplementary Characters", https://msdn.microsoft.com/ en-us/library/windows/desktop/dd374069

[저자소개]

조 규 상 (Gyu-Sang Cho)

1986년 2월 한양대학교 학사
1989년 2월 한양대학교 석사
1997년 2월 한양대학교 박사
동양대학교 컴퓨터정보전학과 교수

E-mail : cho@dyu.ac.kr