한국정보전자통신기술학회논문지(jkiiect)'15-08, Vol.8 No.4
논문 15-08-04-327 http://dx.doi.org/10.17661/jkiiect.2015.8.4.327

새로운 NTFS 디렉토리 인덱스 안티포렌식 기법

조규상*

A New NTFS Anti-Forensic Technique for NTFS Index Entry

Gyu-Sang Cho*

요 약 이 논문에서는 윈도우즈 NTFS 파일시스템에서 디렉토리의 인덱스에 메시지를 숨기기 위한 새로운 안티 포렌식 방법을 제안한다. 인덱스 엔트리 관리를 위하여 채택하고 있는 B-tree 구조의 특징을 이용하여 인덱스 레코드의 슬랙 영역에 숨길 메시지를 저장한다. 안티포렌식을 위해 숨길 메시지가 노출되지 않게 하기 위해서 위장 파일을 사용하여 삭제된 파일이름의 정보가 MFT 엔트리에 남지 않도록 한다. 이 기법의 핵심 아이디어의 이해하기 위해서 B-tree방식의 인덱스 레코드의 운영방법을 소개하고 이 연구에서 제안된 알고리즘을 설명한다. 제작된 소프트웨어를 사용한 메시지를 숨긴 사례를 들어서 이 방법이 실질적인 기법이라는 것을 보인다.

Abstract This work provides new forensic techinque to a hide message on a directory index in Windows NTFS file system. Behavior characteristics of B-tree, which is apoted to manage an index entry, is utilized for hiding message in slack space of an index record. For hidden message not to be exposured, we use a disguised file in order not to be left in a file name attribute of a MFT entry. To understand of key idea of the proposed technique, we describe B-tree indexing method and the proposed of this work. We show the proposed technique is practical for anti-forensic usage with a real message hiding case using a developed software tool.

Key Words : Anti-forensic technique, Data hiding, Directory index, B-tree, NTFS file system

1. 서론

Windows에서의 기본 파일시스템은 NTFS이다. 이것은 Windows NT에서 부터 사용하기 시작하여 Windows 8, 10에 이르는 동안까지 오랜 기간동안 사용되고 있다[1]. 마이크로소프트의 공식사이트인 Microsoft Technet[2]에는 NTFS에 관한 중요한 정보들이 잘 기술되어 있다. 여기에는 NTFS가 무엇인지, NTFS가 어떻게 동작하는지, NTFS에 대한 도구와 설정방법 등이 어떻게 사용되는지에 대해서 정보를 제공하고 있지만 포렌식을 위해서 정작 필요한 자세한 자료구조를 얻을 수는 없다. NTFS 파일시스템에 기술적인 부분과 자료구조 등은 Carrier[3]의 저서에 상세하게 소개되어 있어 NTFS에 대한 좋은 참고자료로 활용되고 있지만 다루고 있는 자료구조에 대한 한계가 있다. 현재까지 잘 알려져 있는 구조만 다루고 있기 때문이다. 디렉토리 인덱스의 데이터 구조는 일반적으로 잘 알려져 있지만 B-tree[4]가 NTFS에서 어떻게 적용되었는지 구체적으로 설명하고 있는 문헌을 발견하기 어렵다.

디지털 포렌식에 관해서는 블로그들에서 실제적인 많은 정보를 얻을 수 있다. 디렉토리의 인덱스에 관해서 Python으로 작성된 프로그램을

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2013R1A1A2064426)
*Corresponding Author : Department of Computer Information, Dongyang University(email: cho@dyu.ac.kr)
Received August 3, 2015 Revised August 8, 2015 Accepted August 13, 2015

William Ballenthin의 블로그[5]에서 찾을 수 있다. 이것을 활용하여 디렉토리에 들어있는 파일의 이름과 파일의 크기, 타임스탬프에 관한 정보등을 얻을 수 있다. Chad Tilbury의 블로그[6]에는 NTFS 인덱스의 할당된 영역에 기록되어 있는 엔트리들의 흔적 정보들을 통해서 어떤 파일과 연관성이 있는지를 여러 유명 포렌식 툴을 사용하여 인덱스에 대한 분석결과를 제시하고 있다.

디지털 포렌식에 관한 연구의 사례 중에서 NTFS에 관련된 의미있는 연구가 수행되었는데 그 중에서 Sameer H. Mahant[7] 등은 삭제된 파일의 복원 문제를 다루었다. Ewa Huebner[8]등은 NTFS 파일 시스템에서 데이터를 숨기는 방법, 감지하는 방법, 복원하는 방법 등에 대한 연구를 수행하였다. Christopher Lees[9]는 웹브라우저를 사용한 내역에 대해서 안티포렌식 방법이 적용된 경우에 $UsnJrnl(USN Journal) 파일에서 특이한 흔적을 찾아내서 분석하는 방법에 관한 연구를 수행하였다.

이 연구와 관련된 Cho의 연구[10]는 파일의 명령들에 따라 타임스탬프의 변화 패턴이 각기 다른 형태를 나타내는 것을 조사하여 타임스탬프를 조작한 경우에 대한 탐지 방법에 응용한 연구를 하였다. 이 후에 타임스탬프의 변화 패턴에 대한 평가함수를 만들어서 어떤 파일 명령이 실행되었는지 판단할 수 있는 논리적인 방법의 구현에 대한 연구를 하였다[11]. 최근 연구[12]에서는 디렉토리 인덱스를 관리하는데 파일의 수가 늘어 날 때 B-tree가 어떤 방식으로 확장되고 어떻게 동작하는지 포렌식 관점에서 얻을 수 있는 정보가 어떤 것인지를 기술한 내용을 발표하였고 그 후의 연구[13]에는 디렉토리 안에 들어 있는 파일 명령 실행 후 디렉토리의 MFT 엔트리내의 타임스탬프에 어떤 변화가 생기는지 조사하고, 디렉토리 전체가 삭제된 상황을 알 수 있는 포렌식 분석 방법을 소개하였다.

이 연구에서는 Windows의 파일시스템인 NTFS에서 사용하고 있는 B-tree구조를 이용하여 인덱스 레코드의 슬랙 영역에 인덱스 엔트리의 내용에 숨길 메시지를 저장하여 데이터 숨김(data hiding)을 하여 포렌식을 방어할 수 있는 새로운 기법을 제안한다. 이 기법은 기존에 알려지지 않은 새로운 방법으로 안티포렌식을 위한 기법이다. 2장에서 이 기법에서 필요한 NTFS의 인덱스에 관한 기본지식에 대해서 살펴보기로 한다. B-tree가 적용된 방법에 대한 것과 인덱스 엔트리의 데이터 구조에 관하여 살펴보기로 한다. 또한 3장에서는 이 연구에서 새롭게 제안한 디렉토리 안티포렌식 기법에 대한 개요 및 구체적인 알고리즘에 대해서 설명하기로 한다. 4장에서는 이 기법에서 사용한 개발된 툴에 대한 설명과 함께 이 기법이 사용된 예를 들어보이기로 한다. 5장에서는 이 연구의 의미와 함께 후속 연구에서 다뤄야 할 부분에 대한 논의로 결론을 맺기로 한다.

2. NTFS 디렉토리 인덱스

2.1 NTFS에서의 B-tree의 이해

2.1.1 일반적인 B-tree

NTFS는 디렉토리 인덱스를 관리하기 위하여 B-트리구조를 사용한다. 일반적으로 B-tree는 인덱스를 빠르게 검색할 수 있고 키의 갯수가 늘어나는 정도에 비하여 트리의 깊이(depth)가 상대적으로 덜 증가하는 좌우 대칭 형태인 균형트리(balanced tree) 방식이다[4]. 루트 노드는 두 개 이상의 하위 노드를 가지며 자식 노드에 대한 포인터와 키 값을 갖는다. 자식 노드를 가리키는 포인터는 키 값의 크기를 비교하여 작은 경우는 왼쪽 자식 노드, 큰 경우는 오른쪽은 자식 노드를 가리키도록 구성된다. [그림 1]은 B-tree의 한 예를 나타낸 것이다.

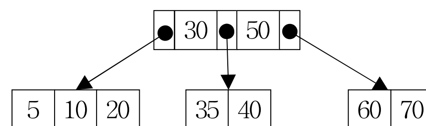

그림 1. 간단한 B-tree의 구조 예
Fig. 1. Diagram of a simple B-tree structure

NTFS가 B-tree 방식을 사용하고 있다는 사실[2,3,4]은 알려져 있지만 이 자료구조를 어떻게 운영하고 있는지에 대해서 상술하고 있는 문헌을 찾기 어렵다. 최근 참고문헌[12, 13]에서는 NTFS의 디렉토리 인덱스의 B-tree의 디렉토리 관리 방법을 실험적인 분석을 수행하여 포렌식에 활용하기 위한 연구가 있었다. B-tree가 파일시스템에서는 실제로 어떻게 구현되고 어떤 방식으로 운영되고 있는지를 분석한 것이다.

2.1.2 $INDEX_ROOT 속성

B-tree의 루트 역할은 $INDEX_ROOT속성이이 하고 있다. 루트 노드 안에 저장되는 인덱스들의 수는 가변적이다. 인덱스 엔트리의 구성요소 중에서 파일명만이 가변적이고 다른 요소들의 크기는 고정되어 있다. 그러므로 파일명의 길이에 따라 이 속성안에 저장될 수 있는 인덱스 엔트리의 수가 달라지게 된다. 기본적인 MFT 엔트리에서 필요한 정보를 저장하고 나면 남는 공간 최대 768바이트이다. 이 공간이 내주 속성인 $INDEX_ROOT에게 허용된 최대공간이다[3, 12, 13].

2.1.3 $INDEX_ALLOCATION 속성

인덱스 엔트리수가 늘어나서 확장을 하게 되는 경우는 외주(Non-resident) 속성인 $INDEX_ALLOCATION에 관련 정보가 기록된다. 다른 외주 속성과 마찬가지로 확장되는 인덱스 레코드들에 대한 정보는 시작되는 인덱스 레코드의 클러스터 번호와 이어지는 레코드들에 대한 Run-length에 정보를 갖는다. 각 인덱스 레코드의 크기는 4KB이다. 이 안에는 여러 개의 인덱스 엔트리가 저장된다.

[그림 2]에서의 내주 속성인 $INDEX_ROOT 안에 여러 개의 인덱스 레코드가 들어 있는 상황을 나타낸 것이다. 이 속성의 공간은 한정적이어서 8.3포맷으로 파일명이 저장된다면 5~6개 정도의 인덱스 엔트리가 저장가능하다[13].

$MFT내의 디렉토리의 MFT entry

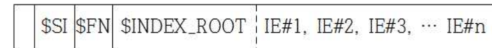

그림 2. $INDEX_ROOT만 존재하는 경우
Fig. 2. Index entries exists in only $INDEX_ROOT

[그림 3]은 인덱스 엔트리가 $INDEX_ROOT에 들어 있기는 공간이 좁아서 Index Record #1을 할당받은 상황을 나타낸 것이다. 이 공간은 4KB 크기이므로 여러 개의 인덱스 엔트리를 기록할 수 있다. 이 때의 $INDEX_ROOT 속성에는 키 인덱스 엔트리가 들어 있지 않고 단지 자식노드로 존재하는 Index Record #1에 대한 포인터 정보만 들어있다. $INDEX_ALLOCATION 속성에는 Index Record #1을 가리키는 클러스터 번호를 갖고 있고 여러 개의 클러스터를 사용하는 경우에 대한 정보를 Run-length정보로 갖고 있다. B-tree에서는 이런 형식의 자식을 갖지 않고 데이터만 들어 있는 노드를 리프노드라고 부른다.

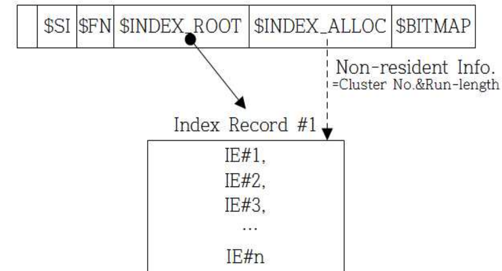

그림 3. 인덱스 엔트리가 한 인덱스 레코드에만 들어 있는 경우
Fig. 3. Index entr exists only in a index reocrd.

[그림 4]는 인덱스 엔트리의 숫자가 늘어나서 여러 개의 인덱스 레코드가 필요하게 된 경우를 나타낸다. 그림 상에서 Index Record #1에 두 개의 키가 존재하는데 키 값 IE#6는 왼쪽 아래의 Index Record #2를 가리키고 키 값 IE#9는 오른쪽 아래의 자식 Index Record #4를 가리킨다. 키 값 IE#6와 IE#9 사이의 포인터는 Index Record

#3을 가리킨다. 이 안에는 두 개의 인덱스 엔트리가 있는 것으로 묘사되어 있다. 인덱스 레코드마다 상황에 따라 들어 있는 인덱스 엔트리의 수는 다를 수 있다. 어떤 경우는 비어 있는 경우도 존재한다.

인덱스 엔트리의 숫자가 계속 늘어서 인덱스 레코드의 수가 늘어 나면 Index Record #1의 키 값들의 수도 증가한다. 계속 늘어나면 이 공간도 부족해지는데 그 때는 새로운 인덱스 레코드를 할당 받아서 반씩 나눠서 저장한 후에 중간의 인덱스 레코드가 $INDEX_ROOT에 저장하게 되며 키 인덱스 엔트리로 사용된다. 이 안에는 두 두 개의 자식 인덱스 레코드를 가리키는 포인터가 들어 있다. 인덱스 엔트리의 수가 지속적으로 더 늘어나게 되어 $INDEX_ROOT의 저장공간마저 부족해지면 그 때는 새로운 인덱스 레코드가 만들어지며 B-tree의 깊이(depth)가 한 층 더 키워지게 된다.

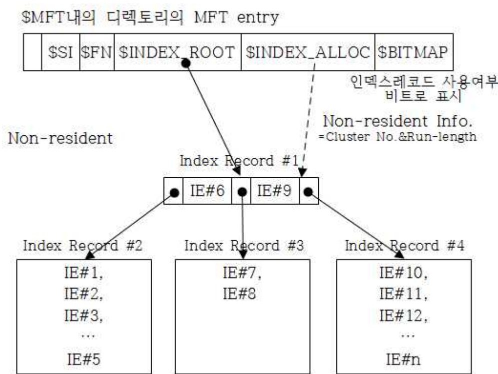

그림 4. 인덱스 엔트리가 여러 인덱스 레코드에 들어있는 경우.

Fig. 4 Index entries exists in several index records

2.1.4 $BITMAP 속성

$BITMAP속성에 4KB 크기의 인덱스 레코드가 여러 개 할당되어 있을 때 인덱스 엔트리가 할당영역에 한 개라도 있으면 1로 표시가 되고 한 개도 없는 빈 경우라면 0으로 표시된다. 이 비트맵 정보를 이용하면 할당되어 있기는 하지만 인덱스 엔트리가 없어서 비어있는 상태의 인덱스 레코드인지를 빠르게 판단할 수 있다.

2.1.5 긴 파일이름과 짧은 파일이름

NTFS는 두가지 형식의 파일이름 포맷을 지원한다[14]. 한가지는 255자까지 사용할 수 있는 긴 이름을 위한 포맷이고 다른 하나는 DOS시절부터 사용하던 형식인 8.3 포맷의 짧은 이름 포맷이다. 기본적으로 NTFS의 MFT엔트리와 디렉토리 인덱스에는 두가지 형식을 모두 사용할 수 있도록 되어 있다.

MFT엔트리에는 하나의 엔트리 안에 긴 파일이름과 짧은 파일이름을 동시에 기록하고 있다. 즉, $FILE_NAME속성(0x30)을 두 번 사용하여 하나는 짧은 파일이름을 위한 속성, 다른 하나는 긴 파일이름을 위한 속성을 동시에 갖고 있다. 그러나 인덱스 엔트리에는 짧은 파일명의 엔트리와 긴 파일명의 엔트리가 마치 서로 독립적인 파일인 것처럼 기록된다. 한 파일에서 비롯된 파일이름이지만 서로 달리 관리되고 있다는 점이 특이하다. 8.3포맷으로 저장되는 경우는 모두 대문자로 변환한 후에 파일명이 중복되지 않도록 짧게 만든 파일명과 확장자 사이에 “~1”과 같은 숫자를 생성하여 이름을 만들고 있다. 유사한 파일의 이름이 많으면 파일이름 뒤에 붙는 숫자가 다양하게 생성된다.

인덱스 엔트리는 파일 이름을 항상 알파벳 순으로 소팅되어 저장하게 된다. 긴 파일이름을 가진 경우라면 짧은 파일이름으로 저장하기 위해서 대문자로 변환되고 짧게 표시된다. 원래의 긴 파일이름도 따로 저장된다. 결과적으로 하나의 파일명이 두 개로 따로 저장되고 다른 파일의 인덱스와 서로 섞이고 순서도 바뀔 수 있다. 파일이름을 소팅할 때는 대소문자를 구분하지는 않는다.

이 연구에서는 인덱스 엔트리에 긴 파일이름을 이용하여 메시지를 숨기려고 한다. 인덱스 엔트리에 짧은 이름이 생성되면 메시지의 순서가 뒤바뀌고 이를 관리하기 어렵게 된다. NTFS에는 이런 파일이름을 저장하는 기록하기 위한 여

러 가지 방법을 제공하고 있다. 이 연구에서 이루고자하는 방법을 구현하기 위해서는 8.3 포맷의 짧은 이름을 저장하지 않는 것이 유리하다. 이를 구현하기 위해서는 8.3 포맷을 사용하지 않기 위해서 NTFS 파일시스템에 다음과 식 (1)과 같이 fsutil을 사용하여 기능을 설정해야 한다.

fsutil behavoir set disable8dot3 1 (1)

이 설정은 시스템 내의 모든 NTFS 파일시스템을 사용하는 드라이브에서 8.3포맷의 사용을 불가능하게 만드는 옵션 설정이다. 옵션의 값이 "0"이면 모든 볼륨에서 8.3포맷을 사용한다는 의미이고, "1"이면 모든 볼륨에서 8.3포맷을 사용 불가능하게 한다는 의미이다. "2"이면 지정한 볼륨에서 8.3포맷을 사용한다는 의미이고, "3"이면 시스템 볼륨을 제외하고 8.3포맷을 사용하지 않는다는 의미이다[15].

2.2 디렉토리에 메시지 숨김을 위해 필요한 인덱스 엔트리 동작에 대한 지식

B-tree 자료구조를 사용하여 NTFS의 인덱스 엔트리가 생성되거나 삭제되면서 보이는 동작 특성을 다음과 같이 정리할 수 있다[12].

1. 인덱스 엔트리는 항상 알파벳 순으로 소팅된다. 대소문자는 구분하지 않는다.

2. 인덱스 레코드는 인덱스 엔트리 들로 꽉 채워져 더 이상 저장할 수 없을 때에는 새로운 인덱스 레코드를 생성하고 두 개의 인덱스 레코드로 나뉘게 된다. 중간 순서의 인덱스 엔트리는 중간 키 역할을 하며 이것을 중심으로 낮은 쪽의 인덱스 엔트리들은 왼쪽 기존의 인덱스 레코드 쪽에 남고 높은 쪽의 인덱스 엔트리들은 오른쪽 인덱스 레코드로 이동하며 중간 키 인덱스 엔트리는 부모노드로 보내진다.

3. 인덱스 레코드가 할당되면 파일이 삭제되어 빈 인덱스 레코드가 생기더라도 해제되지 않는다. 단, 디렉토리에서 최소한 한 개라도 남아 있어야 그 동안 확장했던 모든 인덱스 레코드들이 그대로 유지된다.

4. 인덱스 엔트리의 흔적이 모두 같은 내용으로 채워져 있으면 마지막 엔트리보다 앞에 있는 파일들이 먼저 삭제가 된 것이다.

5. 인덱스 엔트리의 흔적이 서로 다른 것으로 채워져 있으면 인덱스 엔트리의 맨 끝에 있는 파일들이 순차적으로 삭제 된 것이다.

이상에서 알아본 바와 같이 인덱스 엔트리의 생성과 삭제에 따른 동작의 특징들이 메시지를 숨기기 위한 기법의 핵심 아이디어이다.

3. NTFS 디렉토리 안티포렌식 기법

3.1 디렉토리 안티포렌식 기법의 개요

이 논문에서 제안한 디렉토리 안티포렌식을 위한 기법은 [그림 5]에서의 기술한 바와 같은 5가지의 절차로 구성된다.

첫 번째 절차는 "초기화 및 입력"의 과정이다. 숨길 메시지 mhide, 작업 디렉토리 dwork, 위장파일 fcamo 등의 초기 설정을 위한 과정이다.

두 번째 절차는 "숨길 메시지 mhide를 n블록으로 나눔"을 수행하는 절차이다. 이 단계에서는 숨길 메시지의 문장을 블록의 크기에 알맞게 나누는 작업을 한다.

세 번째 절차는 "머릿번호 hpre붙임"을 하는 과정이다. 여러 메시지들을 숨길 때 서로 순서가 섞이지 않도록 하기 위한 방법을 적용하는 절차이다.

네 번째 절차는 "작업 디렉토리 dwork 및 숨길 파일 fhide의 생성"이다. 이 절차는 지정한 작업디렉토리 이름을 가진 디렉토리를 생성하고 이 디렉토리의 인덱스에 메시지를 숨긴다. 숨길 파

일들의 이름에 메시지를 기록하는 과정이다.

다섯 번째 절차는 “인덱스 엔트리를 슬랙영역에 숨김, MFT 엔트리에 위장 파일 fcamo기록” 절차이다. 이 절차가 메시지를 숨기는 핵심적인 과정으로 인덱스 레코드의 슬랙영역을 고의적으로 생성시키며 그 안에 메시지를 숨기고 MFT 엔트리에는 삭제된 파일의 이름으로 위장된 파일 이름이 기록되는 과정을 수행한다.

3.2 디렉토리 안티포렌식 알고리즘

디렉토리 인덱스를 저장하기 위한 인덱스 레코드 공간에 파일의 생성과 삭제를 수행하여 슬랙공간이 생기도록하고 그 슬랙공간에 파일이름을 기록하는 자리에 메시지를 기록한다. 인덱스 엔트리의 슬랙에 메시지를 기록한 후에 위장파일로 파일명을 변경하여 MFT엔트리의 삭제 파일 정보에는 위장파일의 이름이 기록되도록 하여 숨긴 메시지가 노출되지 않도록 하는 것이 이 기법에 대한 설명이다. 다음의 내용은 이 기법을 5가지 절차로 상술한다.

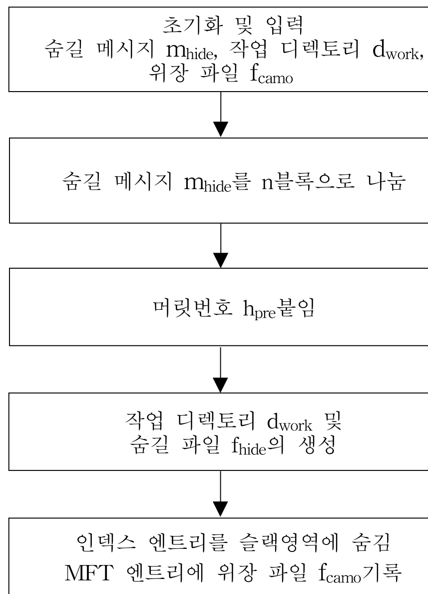

그림 5. 디렉토리 안티포렌식 기법의 절차

Fig. 5. Procedures of directory anti-forensic technique

1. 초기화 및 입력

- 작업 디렉토리 dwork의 이름을 입력받는다.
- 숨길 메시지 mhide를 입력한다.
- 숨길 메시지 mhide의 블록 크기 b를 설정한다.

각 변수를 초기화한다.

2. 숨길 메시지의 n 등분

- 숨길 메시지를 일정한 크기의 블록(파일명 길이만큼, 239자)으로 등분한다.
- 이 때 n의 배수가 되지 않아서 마지막 메시지 블록의 뒷부분에 빈 부분이 생기는 것을 방지하기 위하여 bogus문자를 삽입한다.

3. 블록 순서지정 위한 머릿번호 hpre삽입

인덱스 엔트리는 알파벳 크기 순으로 자동으로 팅된다.

- 숨길 메시지 mhide가 여러 블록으로 나뉠 때 블록의 선두에 붙여서 항상 순서가 유지되도록 한다.
- 숫자 또는 문자를 머릿번호로 지정한다.
- 메시지 블록의 개수에 따라서 자릿수를 정하다. 10개 미만이면 “0-, 1-, 2-, ...”등으로 붙이거나, 알파벳과 숫자를 혼용하여 “a1, a2, a3, ...”과 같이 붙인다.
- 이것을 메시지의 선두부에 합성하여 각 메시지 배열에 저장한다.

4. 숨길 파일 fhide 및 작업 디렉토리 dwork생성

숨김 파일들이 들어갈 디렉토리를 생성한다.

- 숨길 메시지로 파일이름으로 정하고 n개의 파일을 생성한다.

5. 숨길 파일 인덱스 엔트리 슬랙영역에 기록

인덱스 레코드의 슬랙에 기록을 남긴 후에는 파일이름을 변경한 후에 삭제하여 마지막 MFT 엔트리에 삭제된 파일은 위장 파일 fcamo이름으로 기록하게 한다.

작업 디렉토리 dwork에는 최소 한 개의 파일을 남겨두어야 하므로 마지막까지 남길 파일

frem의 이름을 임의로 정한다(예: sample.txt).

남길 파일의 이름은 헤더 문자보다 알파벳 순으로 앞선 문자로 정한다. 인덱스 엔트리는 항상 알파벳 순으로 저장하기 때문에 숨길 메시지 보다 항상 인덱스 엔트리에서 앞에 올 수 있도록 파일이름을 정한다.

숨길 메시지 중의 첫 번째 블록(파일명)은 인덱스 엔트리 상에서 남겨둘 파일 바로 뒤에 위치한다.

첫 숨길 파일을 삭제한 후에 맨 끝 숨길 파일의 이름을 위장파일이름으로 변경한다. 변경한 후에 바로 삭제한다.

첫 숨길 파일을 다시 생성한다. 그러면 최종적으로 맨 끝의 숨길 파일이 삭제되었고 인덱스 엔트리에는 맨 끝 숨길 파일은 기록은 슬랙영역에 남게 된다. MFT 엔트리에는 숨길 파일에 대한 정보는 남지 않고 위장 파일의 정보가 삭제된 파일로 남아있게 된다.

모든 숨길 파일에 위의 과정을 반복한다. 최종적으로 디렉토리 안에는 남길 파일만 남아 있게 된다.

3.3 용어, 기호의 의미

작업 디렉토리 dwork: 숨길 파일이 들어 있는 디렉토리. 이 디렉토리의 인덱스에 인덱스 엔트리안의 파일명 위치에 숨길 메시지가 들어 있다.

남길 파일 frem: 인덱스 레코드 내에서 인덱스 엔트리가 모두 없어지면 MFT 엔트리의 $INDEX_ALLOCATION속성에 기록되어 있는 인덱스 레코드 페이지를 위한 run-length의 기록이 지워지기 때문에 할당된 인덱스 레코드 영역을 유지하기 위해서는 최소한 인덱스 엔트리에 최소한 한 개의 엔트리가 남아 있도록 사용하는 파일이다.

블록 크기 b: 숨길 메시지 mhide가를 저장하기 위하여 사용하는 파일명의 길이. 이 길이는 파일명의 길이로써 최대 255자까지 가능[ ]하다고 알려져 있지만, 이 연구에서 Visual Studio 2012로 작성된 프로그램으로 파일을 생성할 때 실제로는 239자까지만 가능한 것으로 조사된다 (Win7 환경).

숨길 파일 fhide : 숨길 메시지가 파일이름으로 기록된 파일

숨길 메시지 mhide: 숨길 파일명과 같은 의미로 메시지를 파일명으로 사용하여 파일을 생성한다.

위장 파일 fcamo: 숨길 메시지로 파일이름을 생성한 파일의 이름을 다른 이름으로 변경하여 MFT엔트리에 삭제된 상태로 최종의 파일이름으로 남게 만들어 숨길 메시지가 노출되지 않도록 위장하기 위해서 사용하는 파일이름이다.

머릿번호 hpre: 인덱스 엔트리에서는 항상 파일이름을 기준으로 소팅하여 저장하므로 숨기려는 메시지의 순서가 뒤바뀌지 않도록 순서를 유지하기 위하여 메시지의 앞에 붙이는 숫자 또는 문자를 말한다. 저장될 메시지 블록의 개수에 따라서 필요한 만큼 머릿번호를 붙인다. 예를 들면 “a1, a2, a3, ...”등과 같이 붙이거나, “1-, 2-, 3-, ...” 등과 같이 붙인다.

4. 도구 개발 및 사례

4.1 개발 환경 및 프로그램 구현

이 연구에서 작성한 안티포렌식 프로그램 도구는 다음과 같은 개발도구와 실행환경은 다음과 같다.

개발환경:
OS : Windows 7 Ultimate K Service Pack 1
개발도구 : Visual Studio 2012
개발언어 : C/C++, MFC
어플리케이션 타입 : Windows 다이얼로그 프로그램
사용 클래스 : MFC 클래스 및 사용자 클래스

실행환경:
디스크 포맷 : NTFS v3.1
저장매체 : 외장 usb 드라이브
저장공간 : 1TB
디스크 할당 클러스터 크기 : 4,096 바이트
작업 디렉토리 : HideDir
위장 파일 : .sample.txt

이 프로그램은 다음과 같은 기능으로 구성되어 있다. [그림 6]은 제작된 안티포렌식 도구의 실행화면을 나타낸 것이다.

기능의 구성:
작업디렉토리 이름 입력
작업디렉토리 경로 선택
위장 파일이름의 입력
위장파일의 시작 일련번호지정
숨길 메시지의 입력
메시지 숨기기 기능실행
프로그램 종료

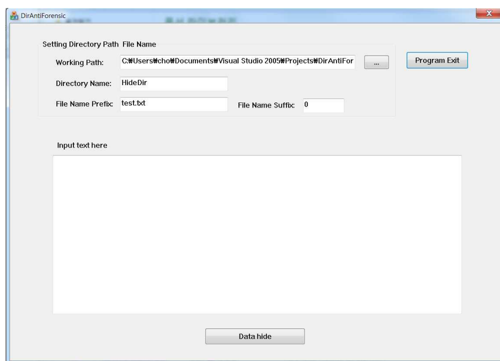

그림 6. 개발된 안티포렌식 도구 프로그램의 실행화면
Fig. 6. Screen shot of the developed anti-forensic program

4.2 사례 적용

개발된 안티-포렌식 도구를 사용하여 “Sound of silence”의 가사를 숨길 메시지로 선택한 후에 실행 한 결과를 나타낸 것이다. 숨길 메시지의 전체의 길이는 공백을 포함하여 전체 1,167자로 구성되어 있다. 한 인덱스 엔트리에 최대 239자를 숨길 수 있다. 숨길 메시지의 헤더에 2문자를 할당하여 실제로는 237자의 문자가 저장된다. 전체 4블록으로 구성되며 블록 크기의 배수에 맞지 않는 끝 블록은 빈 영역을 채우지 않고 문자의 실제 길이만 사용하였다.

[그림 7]은 숨길 메시지가 “HideDir” 디렉토리의 인덱스에 들어 있는 디렉토리의 목록을 나타낸 것이다. 탐색기 창에 표시된 파일은 “.sample.txt” 파일만이 보인다. 표면적으로는 이 안에 메시지가 숨겨있는지 알 수 없다.

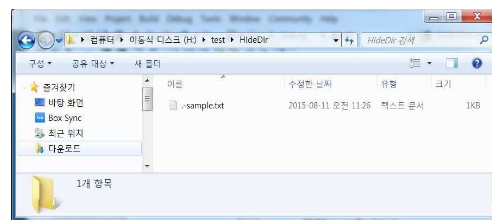

그림 7. 탐색기 창-프로그램 실행 후 “HideDir” 디렉토리 내의 파일의 목록.
Fig. 7. Explorer window-“HideDir” direcotory list after execution of the program

[그림 8]은 디스크 편집 도구인 WinHex로 “HideDir” 디렉토리의 내에 들어 있는 삭제된 파일의 목록을 포착한 것이다. 숨길 메시지가 노출되어 있지 않다. 이것은 MFT 엔트리의 최종 목록에 기록된 내용을 나타낸 것이다. 삭제된 파일의 이름이 “sampleXX.txt”로 표시되어 있어서 메시지를 숨긴 것이 드러나지 않는다.

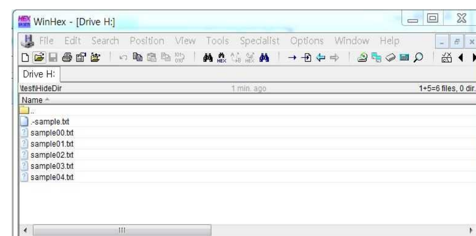

그림 8. 디스크 편집 프로그램-프로그램 실행 후 “HideDir” 디렉토리 내의 위장된 삭제된 파일 목록
Fig. 8. Disk edit program-Disguised delete file list in “HideDir” direcotory after execution of the program

그림 9. 인덱스 레코드-남길 파일 “.sample.txt”와 첫 번째 숨긴 메시지

Fig. 9. Index record- the remianing file “.sample.txt” and the first hidden message

[그림 9]는 첫 번째 메시지를 숨기고 난 후의 인덱스 레코드의 상황을 나타낸 것이다. 이 그림에서 “a” 부분은 인덱스 레코드의 헤더이다. “b” 부분에 기록된 첫 번째 파일은 남길 파일 “.sample.txt”에 관한 정보이다. 이 파일은 디렉토리 내에 현재 존재하고 있는 파일이다. “c” 부분은 인덱스 엔트리의 끝을 알리는 정보이다. “0x02” 값은 목록의 끝임을 나타낸다. 만일 이것이 “0x03”으로 되어 있으면 목록의 끝이고 자식노드가 있음을 나타낸다. 즉, 자식노드로 다른 인덱스 레코드가 존재함을 의미하게 된다. “d”부분은 삭제된 인덱스 엔트리의 정보이며 인덱스 엔트리의 관점에서는 “c”부분까지가 파일시스템이 관장하는 영역이므로 이 부분은 인덱스 레코드의 슬랙영역에 해당한다. 그러나 인덱스 엔트리의 정보가 온전하게 기록되어 있음을 알 수 있다. “0xEF”는 파일이름의 길이를 나타내는 정보이다. $FILE_NAME속성형식에서 파일이름 앞에 붙이는 파일길이를 나타내는 정보다. 이 값은 10진수로 239를 나타낸다. 0x“30 00 2D 00”은 “0-” 문자의 유니코드를 나타낸다. 이것은 숨길 메시지의 머릿번호이다. “e” 부분은 앞의 “c”부분과 마찬가지로 인덱스 엔트리의 끝을 알리는 정보이다. 슬랙에 남아있는 정보이기 때문의 현재의 정보가 아닌 과거의 정보이다. “d” 부분의 파일이 존재하고 있을 때 인덱스 엔트리의 끝을 나타냈던 것이다. 숨길 파일이 삭제되면서 “e”부분은 유효하지 않고 “c”부분이 유효한 인덱스 엔트리의 끝 표식으로 사용되고 있는 것이다.

[그림 10]은 마지막 메시지를 숨기고 난 후의 인덱스 엔트리의 흔적을 나타내고 있다. 중간의 2,3,4번째의 메시지에 대한 설명은 비슷한 구성으로 되어 있고 지면이 부족한 관계로 생략하였다. 이 그림에서 “a” 부분은 인덱스 엔트리의 헤더이다. “b” 부분은 파일이름을 저장한 정보이다. 이 안에 끝 메시지를 숨기고 있다. 끝 메시지는 “0xDD”는 10진수로 221이고 저장된 파일명이 221자임을 나타낸다. 그러므로 숨긴 메시지의 길이는 머릿번호 0x“34 00 2D 00” 즉 “4-”의 2글자를 제외하고 219자의 문자를 저장한 것이다.

그림 10. 인덱스 레코드-남길 파일 “.sample.txt”와 마지막 숨긴 메시지

Fig. 10. Index record- the remianing file “.sample.txt” and the last hidden message

5. 결론

이 연구에서는 Windows의 NTFS 파일시스템에서 디렉토리의 인덱스 관리를 위하여 채택하고 있는 B-tree구조의 특징을 이용하여 인덱스 레코드의 슬랙 영역에 숨길 메시지를 저장하고 이것이 노출이 되지 않도록 안티 포렌식의 새로운 기법을 제안하였다. 이 기법은 기존에 알려지지 않은 방법으로 NTFS 파일시스템의 구조를 이용한 데이터 숨김(data hiding)을 위한 새로운 방법이다.

이 논문에서는 이 기법에서 필요한 NTFS의 인덱스에 관한 기본지식 및 B-tree구조를 사용한 인덱스 엔트리의 특이점에 대해서 살펴보았다. 이 지식을 활용하여 디렉토리 안티포렌식의 알고리즘을 구현하였고 이것에 대한 절차적인 설명을 기술하였다. 파일이름에 숨길 메시지를 기록하고 앞에 있는 파일을 먼저 삭제하는 방식으로 삭제의 순서를 이용해서 인덱스 엔트리를 슬랙공간을 확보하여 숨길 메시지를 기록하는 방법이다. 삭제된 파일이 MFT엔트리에 파일명에 대한 삭제 흔적이 남아있는 것을 방지하기 위하여 위장 파일을 이용하여 포렌식을 방어하는 기법을 설명하였다. 또한 이 기법을 사용하여 개발된 툴에 대한 설명과 함께 이 기법이 적용된 사례를 통하여 이 기법이 현실적으로 사용될 수 있는 안티포렌식을 위한 방법이라는 것을 보였다.

그러나 이 논문에서 다루지 못한 부분이 몇가지 있다. 한 가지는 숨긴 메시지를 수정하여 다시 저장할 수 있는 방법이 적용되지 않았다. 현재는 숨길 메시지가 섞이지 않기 위하여 헤더를 부착하고 있다. 수정하여 내용이 변경된 경우 부분적으로 메시지가 옮겨져서 다른 메시지에 영향을 주는 것을 고려한다면 이 부분에 대한 알고리즘의 수정이 필요하다. 그리고 파일시스템에서 허용하지 않는 문자에 대한 처리 부분이다. “*, >, <, ?”는 파일이름에 사용할 수 없는 기호들의 사용문제이다. 현재에는 이 부분에 대한 처리방법을 적용하고 있지 않아서 이런 문자가 들어있는 경우는 사용할 수 없다는 단점이 있다. 개선책으로는 “\”를 한 개의 문자와 함께 붙여서 C언어의 문자열 처리방법에서 처럼 escape문자 방식으로 사용하는 것이 대안으로 제시될 수 있다. 또 다른 한 가지는 암호화에 대한 고려이다. 메시지 자체가 노출되지 않아야 하는 경우에는 암호화를 반드시 적용해야 한다. 디렉토리 인덱스 슬랙에 들어 있는 내용이 디스크 전수조사나 우연에 의해 발견되더라도 메시지의 내용이 노출 및 분석되지 않도록 하는 방법에 대한 추가적인 연구가 필요하다.

이 기법이 실제로 안티포렌식을 위한 방법으로 사용되는 경우에 대비하여 이 기법의 특징에 대한 분석과 대응 방법 및 포렌식 방법을 개발하여 안티포렌식을 방어하기 위한 새로운 디지털 포렌식 방법 즉, 안티-안티 포렌식(anti-anti forensics)방법으로 발전하기를 기대한다.

REFERENCES

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

[2] Microsoft TechNet, “NTFS Technical Reference”,https://technet.microsoft.com/en-us/library/cc758691(v=ws.10).aspx.

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

[4] Wikipedia, “B-tree”, http://en.wikipedia.org/wiki/B-tree.

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

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

[7] Sameer H. Mahant and B. B. Meshram, “NTFS Deleted Files Recovery: Forensics View”, IRACST(- International Journal of

Computer Science and Information Technology & Security (IJCSITS), Vol. 2, pp. 491-497, No.3, 2012.
[8] Ewa Huebner, Derek Bem and Cheong Kai Wee, "Data hiding in the NTFS file system", Digital Investigation, Vol. 3, Issue 4, pp. 211-226, 2006
[9] Christopher Lees, "Determining removal of forensic artefacts using the USN change journalOriginal", Digital Investigation, Vol. 10, Issue 4, pp. 300-310, 2013.
[10] G.-S. Cho, "A computer forensic method for detecting timestamp forgery in NTFS", Computers & Security, Vol. 34, pp. 36-46, 2013.
[11] Gyu-Sang Cho, A Digital Forensic Method by an Evaluation Function Based on Timestamp Changing Patterns. (2014), Journal of KSDIM(ISSN:1738-6667), Vol. 10, No. 2, pp. 91-105.
[12] G.-S. Cho, "NTFS Directory Index Analysis for Computer Forensics", Proceedings of IMIS 2015, Blumenau Brazil, July 2015.
[13] Gyu-Sang Cho, A Digital Forensic Analysis for Directory in Windows File System. (2015), Journal of KSDIM(ISSN:1738-6667), Vol. 11, No. 2, pp. 73-89.
[14] Microsoft MSDN, "Naming Files, Paths, and Namespace-Short vs. Long Names", http://msdn.microsoft.com.
[15] Microsoft TechNet, Fsutil behavior, "https://technet.microsoft.com/en-us/library/cc785435.aspx"

저자약력

조 규 상(Gyu-Sang Cho) [회원]

- 1997년 2월 : 한양대학교 대학원 전자공학과(공학박사)
- 2010년 9월 ~ 2011년 8월 : 미국 Purdue대학교 방문연구원 Cyber Forensic Lab
- 1996년 3월 ~ 현재 : 동양대학교 컴퓨터정보전학과 교수

<관심분야>
디지털 포렌식, 시스템 보안