Journal of The Korea Society of Computer and Information
Vol. 23 No. 8, pp. 75-84, August 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.08.075

A Steganographic Data Hiding Method in Timestamps by Bit Correction Technique for Anti-Forensics

Gyu-Sang Cho*

Abstract

In this research, a bit correction technique of data hiding method in timestamp of MFT entry in NTFS file system is proposed. This method is proposed in two ways, depending on the number of bytes of data to hide. A basic data hiding method using a bit correction technique to solve the problems of the conventional 2-byte technique is proposed. In order to increase the capacity of the data, a 3-byte data hiding method using an extended bit correction technique is proposed. The data hiding method in the timestamps is based on the fact that is not revealed in the Windows explorer window and the command prompt window even if any data is hidden in the timestamp area of less than one second. It is shown that the validity of the proposed method through the experimental two cases of the basic data hiding method by the bit correction method and the 3-byte data hiding method by the extended bit correction method.

▸Keyword: Data hiding, Timestamps, Timestamp Correction Technique, File System, NTFS

I. Introduction

데이터 숨기기는 나중에 사용을 위해서 접근하기 어렵게 만드는 것이다. 이런 목적을 위해서 사용하는 방법은 암호화, 스테가노그래피 등 다양한 방법이 사용된다. 파일시스템도 데이터 숨기기를 위한 하나의 방법으로 사용된다. N. A. Hassan and R. Hijazi[1]등의 저서에서는 ADS, 볼륨 섀도잉, 복원점, 레지스트리, 파일슬랙, 은닉파티션, 디스크 배드블록, HPA와 DCO이용하기, 가상메모리, 프리페치 등 Windows에서 사용할 수 있는 다양한 데이터 숨기기 방법을 소개하고 있다.

파일시스템의 메타데이터를 이용한 방법들에 관한 여러 연구가 있었다. Huebner등[2]은 NTFS 파일시스템의 데이터 구조의 특성을 이용하여 안티포렌식을 목적으로 $Boot 파일, $BadClus, 파일과 디렉토리의 $DATA 속성, ADS, 볼륨, 파일슬랙 등을 활용하여 숨겨진 데이터를 탐지와 복구 방법을 소개하였다. Cho의 연구[3]에서는 NTFS의 디렉토리 인덱스에서 사용하는 B-tree의 데이터 구조를 이용하여 데이터를 숨길 수 있는 방법을 연구하였다. 이것은 인덱스 엔트리의 파일명을 숨길 데이터로 사용하는데 인덱스 레코드의 빈 영역이 안전하게 사용될 수 있는 방법을 발견하여 데이터 숨기기에 응용한 방법이다. Fu-Hau등[4]은 WinXP의 NTFS파일시스템에서 MFT 레코드의 정보를 변조하는 Concealer와 숨겨진 파일의 데이터 섹터의 순서를 뒤바꾸는 Obfuscator의 기능을 채용하여 안티바이러스나 데이터 복구 소프트웨어로 데이터가 숨겨진 사실이 잘 드러나지 않도록 하는 방법을 제안하였다. 또한, 안드로이드 OS에서 반송파 신호나 민감한 개인의료정보 등 여러 종류의 데이터를 이용하여 데이터를 숨기는 방법이 소개되었다[15].

최근, 타임스탬프의 시간 표현방식을 이용한 데이터 숨기기를 하는 안티포렌식 방법이 몇 가지 소개되었다. Cho[5]의 연구에서는 100나노초 단위로 타임스탬프의 시간을 저장하는 NTFS파일시스템의 탐색기 창이나 명령프롬프트 창에서 타임스탬프에 1초미만의 시간은 표기가 되지 않는다는 점에 착안하여 이곳에 데이터를 숨기

• 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. 13, Revised: 2018. 08. 20, Accepted: 2018. 08. 27.
• 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)

는 방법을 제안하였다. 안전하게 데이터를 저장하기 위하여 10,000,000나노초가 1초로 표현되어 저장되는 많은 데이터 비트를 숨기기 공간으로 활용하는 방법으로 파일시스템의 구조에 데이터를 숨기는 다른 기법에 비하여 숨긴 데이터가 안전하게 보전되고 노출될 염려가 매우 적은 스테가노그래피적 방법이다.

거의 동시에 발표된 S. Neuner등[6]의 연구에서는 타임스탬프의 여유공간에 대량의 정보를 저장할 수 있는 가능성에 대하여 연구하였다. 이 연구에서 제안한 방법인 TOMS는 은닉성, 강인성, 넓은 응용성을 제공하는 스테가노그래픽 시스템이다. 정상적인 환경에서는 잘 사용되지 않는 NTFS 타임스탬프를 활용하여 1초미만의 정보저장부분에 데이터를 숨기는 방법이다. 파일이 생성된 후에 변경이 잘 안되는 생성시간과 접근시간을 데이터 숨기기 장소로 사용하는 것은 다르지만 기본적인 개념의 출발은 Cho[5]의 방법과 같다. Cho의 방법은 $STANDARD_INFORMATION과 $FILE_NAME속성의 각각 4개의 타임스탬프에 2바이트씩 저장한다는 점이 다르다. 이 타임스탬프에 데이터를 숨기는 이 두 방법은 같은 시기에 발표된 최초의 시도이다.

그 후에 ext4 파일시스템의 타임스탬프를 스테가노그래픽 채널로 사용하기 위한 데이터 숨기기 방법의 연구가 발표되었다[7]. 안티포렌식을 위한 기법으로 개발되었으며 숨겨진 데이터의 비밀성, 사용자들이 전문지식 없이 쓸 수 있는 쉬운 사용성을 목표로 개발된 것이다. 이것은 이전에 발표된 연구[6]를 모체로 하고 있으며 NTFS에 적용된 방식과 같은 방법을 ext4에 적용한 것이다.

이 논문에서는 Cho[5]가 제안한 2바이트 데이터를 숨기는 방법에서 타임스탬프의 하위 2바이트(16비트)를 숨길 때 발생하는 문제점을 제기하고 이것에 대한 개선책을 비트교정 기법으로 제안한다. 또한, 데이터를 더 많이 숨기기 위한 3바이트 데이터 숨기기 방법과 이에 필요한 확장 비트교정 기법을 제안한다.

2장에서는 기존의 방법에 대한 문제제기를 한다. 이 방법을 구현하기 위해서 필요한 정보와 사용되는 도구에 대하여 3장에서 기술하기로 한다. 4장에서는 제안된 비트교정 기법이 적용된 데이터 숨기기 방법에 대한 설명과 플로우챠트를 보인다. 5장에서는 두 가지 사례를 통해서 제안한 방법이 타당하게 동작함을 보이고 6장에서 결론을 맺는다.

II. Problem Statements

1. Data Hiding in Lower 16bits of Timestamps

Cho[5]의 연구에서는 NTFS의 파일의 정보를 나타내는 MFT 엔트리에 들어있는 $STANDARD_INFORMATION 속성과 $FILE_NAME 속성의 각각 4가지 타임스탬프 즉, 생성시간, 수정시간, MFT엔트리 수정시간, 접근시간 등에 각각 1초 미만의 저장공간에 2바이트를 데이터 숨기는 공간으로 사용하였다. 그러나 이 방법은 2바이트를 데이터 숨기기에 할당하는 과정에서 오류를 포함하고 있다.

FILETIME 데이터 구조에서는 1초 미만의 시간을 밀리초(mili-second)와 나노초(nano-second)값 mmm:nnnn형식으로 10진 7자리를 사용하는데 이것은 정수값 0,000,000 ~ 9,999,999까지를 사용하는 것이다. 즉, 정수값 1이 100나노초이기 때문에 1초는 10,000,000나노초로 표현된다[10].

1초 미만의 자리의 7자리 10진 정수 값을 16진수 값으로 변환하면 0~0x98967F가 된다. 24비트로 표현가능한 숫자이다. 이 안에 데이터를 숨기려고 한다. 8바이트 코드 3개를 사용하여 24비트를 사용한다면 상위의 한 비트에서 겹치는 부분이 발생하기 때문에 8바이트 2개의 데이터를 사용하여 안전하게 타임스탬프 안에 데이터를 숨기는 방법을 사용한 것이다.

그러나 타임스탬프를 표현하는 숫자는 10진 정수 단위로 계산이 이루어진다. 100ns단위마다 1씩 증가하는 정수값을 표현하기 때문에 1초이상의 시간을 나타내는데 하위 16비트 내에 유의미한 비트값이 포함될 수 있다. 비트단위의 경계가 완벽하게 나눠지지 않는다는 점에서 문제가 발생한다.

1초 미만의 값이 모두 0으로 되어 있는 경우를 살펴보면 2018-08-31 03:34:56(UTC)의 경우에 정수값으로 131,801,600,960,000,000 표현된다. 이것을 16진수로 표현하면 0x01D440DB960C7000이 된다. 하위 16비트는 “7000”으로 표시되기 때문에 이것을 임의로 변경하게 되면 초단위에서 1초의 변동이 발생한다. 2018-04-28 10:35:58(UTC)의 경우도 이런 문제가 발생한다. 이 시간은 10진수로는 131,693,853,580,000,000로 표현되고 16진수로는 0x01D3DEDCB1BCFB00로 표시된다. 하위 16비트에 데이터 숨기려면 모두 “0x0000”인 상태이어야 하는데 실제로는“0xFB00”의 유의미한 값이 들어 있다. 원래 갖고 있던 값에서 1초의 오차가 발생하는데 작은 시간이지만 이것을 무시할 수는 없다.

반면에 어떤 타임스탬프 2018-08-03 04:51:56의 경우는 131,777,455,167,596,554이고 0x01D42AE5B4AB680A이다. 이것의 하위 16비트를 “0x0000”으로 변경하면 2018-08-03 04:51:56가 되고 131,777,455,167,569,920가 된다. 하위자리가 131,777,455,160,000,000로 하위자리가 “0,000,000”이 될 것을 기대하였으나 실제로는 그렇게 되지 않는다. 그러나 시간은 변함없이 2018-08-03 04:51:56이다. 여기에 하위 16비트에 “680A”가 들어 있음에도 하위비트에 0x41(‘A’)와 0x42(‘B’)를 숨겨 넣는다면 131,777,455,167,586,881가 되고 시간은 2018-08-03 04:51:56로 역시 변함없다. 유의미한 데이터 일지라도 어떤 경우는 1초 미만의 시간변화만 갖기 때문에 실제로 “AB”두 글자를 숨겨도 시간이 변경이 되지 않은 것이어서 안전하다고 생각할 수 있었던 것으로 오해될 수 있다. 그러나 스테가노그래피 방식으로 숨기는 경우에는 데이터가 숨겨져 있다는 사실조차 알려지지 않아야하기 때문에 사소한 변화라도 포착되지 않도록 해야 한다.

이 연구에서는 기존의 방법은 타임스탬프의 하위 16비트를 데이터를 숨기기 위한 장소로 사용할 때 발생하는 부작용 문제를 해결하기 위한 방안으로 숨긴 데이터의 영향으로 타임스탬프의 시분초 표현에서의 변화가 생기지 않도록 상위비트 17번째 비트의 값에

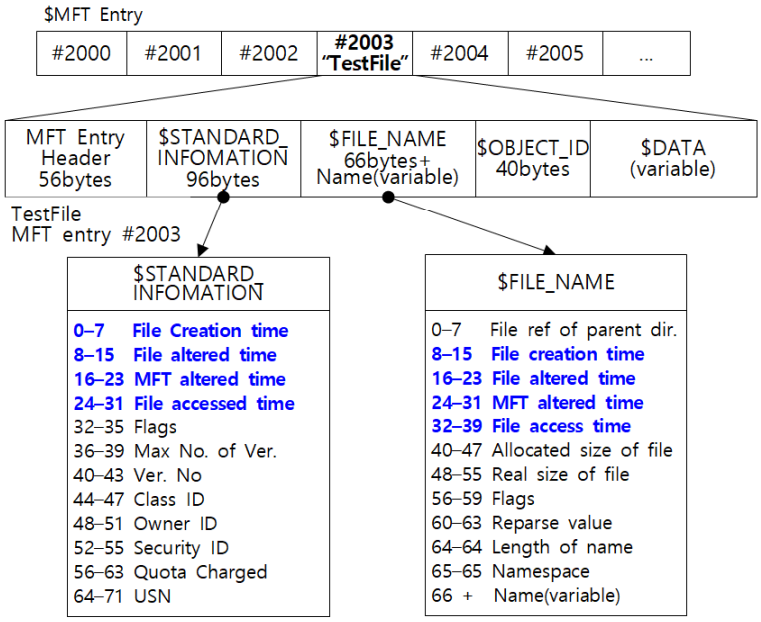

Fig. 1. MFT entry and $STANDARD_INFORMATION and $FILE_NAME structure

가감연산을 수행하여 BitCorrection 기능을 수행하는 비트보정 기법을 통해서 구현한다.

그리고 기존의 2바이트 저장 공간에서 더 많은 공간을 확보하기 위하여 데이터를 3바이트로 확장하여 저장할 수 있는 방법을 제안하고 이에 수반되는 비트보정 기법을 제안하기로 한다. 3바이트를 저장하는 경우 24비트에서 간혹 데이터가 넘치거나 줄어드는 현상이 발생하여 1초가 늘어나거나 1초가 줄어들어서 타임스탬프의 무결성 문제가 야기 될 수 있는 것을 24번째 비트와 25번째 비트의 값을 가감 연산하여 비트보정을 수행하는 3바이트 데이터 숨기기를 위한 비트보정(Bit Correction) 기법을 적용하기로 한다.

III. FILETIME Format and Timestamp Manipulating Tools

1. FILETIME Format

1.1 MFT Entry Attribute

Fig. 1은 MFT엔트리에 들어 있는 속성들과 그 속성 안에 들어 있는 타임스탬프와의 관계를 나타낸 것이다. "TestFile"이라는 파일의 정보는 엔트리 번호 #2003을 갖고 있고 이것은 1KB크기의 MFT엔트리 공간에 저장되어 있다. 이 안에는 56바이트의 MFT엔트리헤더, 96바이트의 $STANDARD_INFORMATION속성($SI), 66바이트의 고정된 요소값과 파일명의 길이가 1~255자까지 가변적인 길이를 갖는$FILE_NAME($FN)속성이 온다. 그 후에는 $OBJECT_ID와 파일내용이 저장되는 $DATA로 구성된다[3,8,16].

$SI속성에는 파일이나 디렉토리의 기본적인 메타정보가 저장되어 있다. 이 안에는 4개의 시간정보도 담겨있다. 즉, 생성시간, 수정시간, MFT엔트리 변경시간, 접근시간 등의 4가지 타임스탬프가 저장된다. 생성시간은 파일이 만들어질 때의 시간이 기록된다. 수정시간에는 파일이 갱신되는 시점의 시간이 기록된다. MFT엔트리 변경시간에는 파일과 관련된 메타정보(MFT엔트리 관련 정보)가 저장된다. 마지막으로 접근시간에는 파일이 읽혀졌을 때의 시간이 기록된다[3,8,16]

$FN 속성에도 $SI과 마찬가지로 4개의 타임스탬프가 들어 있다. 의미와 내용은 같은 방식이지만 시간값이 동일하지는 않다. 저장되는 시간정보는 파일 명령과 관련이 있다[11,16].

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 Reparse 정보(각 4바이트)가 저장된다. 파일명 길이(64바이트 오프셋), 파일명 네임스페이스(65바이트 오프셋)이 저장되고 가변적인 길이는 갖는 파일명이 66바이트 오프셋에서부터 저장된다[3,8,16]. 파일명은 2바이트 유니코드를 사용한다. 1~255자까지의 파일명을 사용할 수 있다. $FN속성에는 파일명만 저장되지만 파일명의 길이를 계산할 때는 드라이브문자와 경로를 포함한 문자의 길이가 255자 이하이어야 한다. 경로명이 긴 경우는 255자에서 경로명 길이를 뺀 길이만큼만 파일명으로 사용할 수 있다는 의미다[3].

1.2 FILETIME Structure

FILETIME 데이터 구조는 64bit로 구성된다. 이것은 1601년 1월 1일 0시 부터 100나노초 정밀도(100ns granularity)로 시간을 표시한다. UTC형식의 타임스탬프를 사용한다. 현지 시간을 저장하는 것이 아닌 표준시간을 저장하는 방식이어서 시간대(time zone)나 일광절약시간(daylight saving time)에 영향 없이 어떤 순간에 해당하는 값에 대한 표현은 유일하게 표현하는 방법이다[10].

FILETIME 데이터 구조는 두 개의 32 비트 값을 구조체로 아래와

같이 정의된다. 여기서 dwLowDateTime는 FILETIME의 32비트 하위부 값이고 dwHighDateTime은 32비트 상위부 값이다[10].

```
typedef struct _FILETIME {
  DWORD dwLowDateTime;
  DWORD dwHighDateTime;
} FILETIME;
```

이 방식은 1초 미만의 값은 10진 정수 7자리로 표시된다. 정수 1이 100나노초이므로 1초는 10,000,000이 된다. 그러므로 1초미만의 수의 표현범위는 0~9,999,999까지가 된다. 이것을 16진수로 표시하면 0x98967F이다. 이 값에 해당하는 비트수만큼 데이터를 숨길 수 있는 의미이다. 최대수를 넘지 않는 범위에서 비트로 활용가능한 경우를 나타내면 23비트까지 안전하게 사용할 수 있다. 이것을 16진수로 표현하면 0x7FFFF까지 사용하면 데이터를 다루기 쉽다. 약간의 사용하지 않는 범위 (8,388,608 ~9,999,999)가 발생하지만 데이터를 안전하고 편리하게 다룬다는 관점에서는 이 정도는 여분으로 남겨두어도 좋다는 판단을 할 수 있다.

2. Timestamp Manipulating Tools

2.1 FileTouch.exe

SetFileTime() 함수는 생성시간, 수정시간, 접근시간 등의 3가지 타임스탬프만 수정이 가능하다. MFT 엔트리 수정시간에 대해서는 접근할 수 있는 기능을 갖고 있지 않지만 탐색기 창이나 명령프롬프트 창에는 이 타임스탬프를 표시하지 않기 때문에 타임스탬프의 변조를 시도할 수 있다. MFT엔트리 수정시간에는 현재의 작업시간이 기록된다[11]. 아래의 도구들은 이 함수를 사용한다.

FileTouch.exe(http://www.softtreetech.com/)는 다음의 옵션으로 사용된다. /W는 수정시간, /A는 접근시간, /C는 생성시간, /S하위폴더, /D는 mm-dd-yyyy형식의 날짜, /T는 hh:mm:ss형식의 시간을 지정하고 filemask는 "*.txt"와 같은 형태를 지정한다.

옵션에서 선택할 수 있는 경우 중에서 MFT엔트리 수정시간이 없다는 점과 시간포맷에서 초미만의 값을 지정하는 표현이 불가능하다.

예: FileTouch [/W][/A][/C][/S][/D date][/T time] filemask

이와 유사한 기능을 갖고 있는 명령행 방식의 프로그램은 chtime.exe(https://github.com/Loadmaster/chtime-win32)가 있고 eXpress TimeStamp Toucher(xtst.exe, http://www.irnis.net)는 GUI방식의 프로그램이다.

2.2 Timestomp

Timestomp는 James C. Foster and Vincent Lie가 제작한 도구로써 타임스탬프를 변조하거나 삭제할 수 있는 기능을 갖추고 있다[13]. 이것은 MFT 엔트리 수정시간을 변경할 수 있는 기능을 갖고 있어 유용한 도구이다. 그러나 2장의 문제제기에서 언급한 바와 같이 $FN 속성의 타임스탬프를 직접 변경시킬 수 있는 기능을 갖고 있지 않다. 이 속성을 변경시키기 위해서는 $SI 속성의 타임스탬프를 변경한 후에 파일이동 명령을 수행하면 $SI 속성 타임스탬프가 타임스탬프로 복사된다. 이 방법을 이용하면 $SI 속성과 $FN 속성을 의도한 대로 바꿀 수 있다[12].

timestomp.exe의 또 다른 특징은 초단위까지만 변경이 가능하다는 것이다. 1초미만의 ms, ns의 시간설정 기능을 갖추고 있지 않다. 다음과 같이 명령행에서 프로그램을 실행시킬 수 있다. 인수는 3개를 가진다. 첫 번째 인수는 경로를 포함한 파일명을 지정하는 것이다. 두 번째 인수에는 변경할 타임스탬프의 종류를 지정한다. -m 은 쓰기시간, -a는 접근시간, -c는 생성시간, -e는 MFT엔트리 수정시간, -z는 4가지 모든 속성 모두, -f는 지정한 파일에 있는 타임스탬프, -v는 타임스탬프 보이기 등의 옵션을 사용할 수 있다. 세 번째 인수에는 “요일 월/일/년 시:분:초 AM/PM"형식으로 날짜와 시간을 지정한다[13].

예: c:₩>timestomp.exe c:₩test.txt -z "Saturday 08/01/2018 12:34:56 PM“

2.3 SetMace

SetMACE의 가장 최신 버전은 2014년에 Ver. 1.0.0.16이다. 이것은 파일시스템을 경유하지 않고 물리적인 디스크를 직접 접근하여 타임스탬프에 쓰기 작업을 수행하는 도구라서 기존의 파일시스템 관련 API를 사용하는 프로그램과 동작하는 방식이 다르다. timestomp.exe에서 갖추지 못한 $FILE_NAME 속성에 직접 쓰기 기능을 수행할 수 있다는 장점이 있다. 또한 시간을 YYYY:MM:DD:HH:MM:SS:MSMSMS:NSNSNSNS 형태로 입력하여 년:월:일:시:분:초:밀리초:나노초 단위까지 직접 입력할 수 있는 기능을 갖추고 있다. 타임스탬프 변조도구로는 유일하게 밀리초:나노초를 설정할 수 있다[14].

인수는 네 종류이다. 첫 번째 인수는 경로를 포함한 파일명을 사용한다. 두 번째 인수에 사용하는 옵션으로는 "-m" (수정시간), "-a" (접근시간), "-c" (생성시간), "-e" (MFT엔트리 수정시간), "-z" (4종류 타임스탬프 모두), and "-d" (기존 타임스탬프를 덤프) 등을 사용할 수 있다. 세 번째 인수에는 "2018:08:01:12:34:56:000:0000"의 형태로 시간을 입력한다네 번째 인수에는 $STANDARD_INFORMATION에는 “-si", $FILE_NAME에는 ”-fn"의 속성을 대상으로 지정하고 둘다 지정할 때는 “-x'를 사용한다[14].

예:c:₩>setmace.exe c:₩file.txt -e
"2018:08:01:12:34:56:789:1234" -fn

IV. Data Hiding Method in Timestamps by Bit Correction Technique

1. Overview

이 연구에서는 NTFS 파일시스템에서 MFT엔트리의 속성

$STANDARD_INFORMATION속성과 $FILE_NAME속성의 각기 4개의 타임스탬프 즉, 생성시간, 수정시간, MFT엔트리 수정시간, 접근시간 등의 타임스탬프(각 8바이트)에 저장된 FILETIME 데이터 구조의 타임스탬프 값에서 1초미만의 하위의 값들은 탐색기 창이나 명령프롬프트 창에서 표시되지 않는다는 점에 착안하여 그 안에 데이터를 감추려고 한다.

2장의 문제제기에 따라 기존에 제안된 방법에 문제점을 해결하고 더 많은 데이터를 숨기는 방법을 고안하는 것이 이 연구의 목적이다.

제기된 문제점은 하위비트 16비트에 데이터를 숨기는 기존의 방법에서 1초의 타임스탬프의 변동이 발생하는 문제를 해결하는 방법으로 비트보정(Bit Correnction)기법을 제안한다. 많은 데이터를 저장하기 위하여 3바이트 저장기법을 제안하였고 여기서 발생하는 비트보정에 대한 문제는 확장 비트보정 기법을 적용하는 방법을 제안한다.

2. Data Hiding Method in Timestamps by Bit Correction Method

기존의 방법은 타임스탬프의 하위 16비트를 데이터를 숨기기 위한 장소로 사용하는데 있어서 생기는 부작용이 발생함을 알았다. 스테가노그래피 형태의 숨기기 방법을 위해서는 표면적으로 변화가 발생하지 않는 방법을 사용해야 한다. 이 연구에서는 이 부분에 대한 해결방법으로 비트보정 기법을 제안하기로 한다. 다음은 이 방법의 절차이다. Fig. 2의 플로우챠트에는 아래 절차의 번호를 해당하는 부분에 표기한다.

절차① 초기화 준비과정. 한 파일에 저장할 문자수를 정하는 과정이다. 기본 2바이트이고 확장형 방식에는 3바이트 할당한다. arraySize는 한 파일의 타임스탬프에 저장할 크기 2바이트씩 $SI, $FN에 저장하므로 전체 16바이트가 된다. 확장형은 3바이트씩 저장하므로 24바이트가 된다. $FN의 경우 긴 이름 형식과 8.3포맷의 짧은 이름의 두 형식을 동시에 사용하는 경우에는 16바이트의 저장공간이 더 늘어 난다. 아래 설명은 8.3 포맷을 사용하지 않는 경우에 해당한다.

byteAllocation(arrayFile, arraySize) (1)
arraySize=16 (2)
arraySizeExt=24 (3)

절차② 전체 숨길 데이터가 저장된 공간을 할당한다. totalData는 전체 숨기려는 데이터의 수이다. type은 파일에서 입력 또는 키보드에서 입력하는 경우를 말한다.

mAllocation[totalData]=inputData(type) (4)

절차③ 저장할 전체 문자수를 결정하고 문자를 숨길 파일을 선택한다.

numOfFiles=INT(totalData/arraySize)+1 (5)

절차④ 데이터 크기 totalData와 한 파일에 숨길 문자수 arraySize에 따라 결정된 데이터를 숨길 파일을 선택한다. n=1~numOfFiles.

selectFile2Hide(n) (6)

절차⑤ 파일에서 $SI와 $FN의 각기 4개의 타임스탬프 전체 8개의 타임스탬프를 구한다.

getTimestampsOfFile(fileName) (7)

절차⑥ $SI의 4개 타임스탬프 $T_{\$SI}^{C}$, $T_{\$SI}^{M}$, $T_{\$SI}^{E}$, $T_{\$SI}^{A}$와 $FN의 4개 타임스탬프 $T_{\$FN}^{C}$, $T_{\$FN}^{M}$, $T_{\$FN}^{E}$, $T_{\$FN}^{A}$에 대하여 하위 16비트에 데이터를 저장한다.

이 절차는 숨길 데이터 공간을 확보하기 위한 방법이다. 각 타임스탬프는 다음과 같이 한 개의 타임스탬프 T는 8개의 바이트로 상위 B7로부터 하위 B0까지 나타낸다.

$$T = B_7B_6B_5B_4B_3B_2B_1B_0 \quad (8)$$

여기에 숨길 공간을 확보하기 위하여 다음의 bitwise연산을 연속하여 수행한다.

T = T >> Bits2Hide
T = T << Bits2Hide (9)

여기서 Bits2Hide=16이다. 이 연산을 마치고 난 후의 T는 0_1=0000 0000는 0_0=0000 0000로 각각 0으로 비트가 채워진다.

$$T_{bitsoff} = B_7B_6B_5B_4B_3B_2\underline{0_1 0_0} \quad (10)$$

절차⑦ 숨기려는 데이터는 H_1와 H_0로 표현된다. 이 두 바이트 데이터에는 각각 ASCII 문자 2개가 저장되고 Unicode인 경우 상위바이트는 H_1에 저장되고 H_0에 하위바이트가 저장된다. 유니코드를 사용하면 타임스탬프 당 한 개의 코드가 저장되므로 한 파일의 타임스탬프에는 8문자가 저장된다. 과정6의 $T_{bitsoff}$와 숨기려는 2바이트 H_1H_0를 합산하면 다음과 같이 구할 수 있다.

$$T_{hide} = B_7B_6B_5B_4B_3B_2\underline{H_1H_0} \quad (11)$$

절차⑧ T_{hide}를 타임스탬프에 기록한다.

saveMaceSI($T_{\$SI}^{C}$, $T_{\$SI}^{M}$, $T_{\$SI}^{E}$, $T_{\$SI}^{A}$)
saveMaceFN($T_{\$FN}^{C}$, $T_{\$FN}^{M}$, $T_{\$FN}^{E}$, $T_{\$FN}^{A}$) (12)

절차⑨ 데이터를 숨기기 전의 타임스탬프와 숨기고 난 후의 타임스탬프간에 차이점에 발생하면 비트보정 기법(Bit Correction Technique)이 동작한다. 숨기기 전후의 타임스탬프의 비교는 compareTimestamp()에 의해서 수행된다. 시간의 차이가 발견되면 correctTimestamp()기능이 동작한다. 데이터 숨기기 전의 타임스탬프를 기준으로 타임스탬프의 시간이

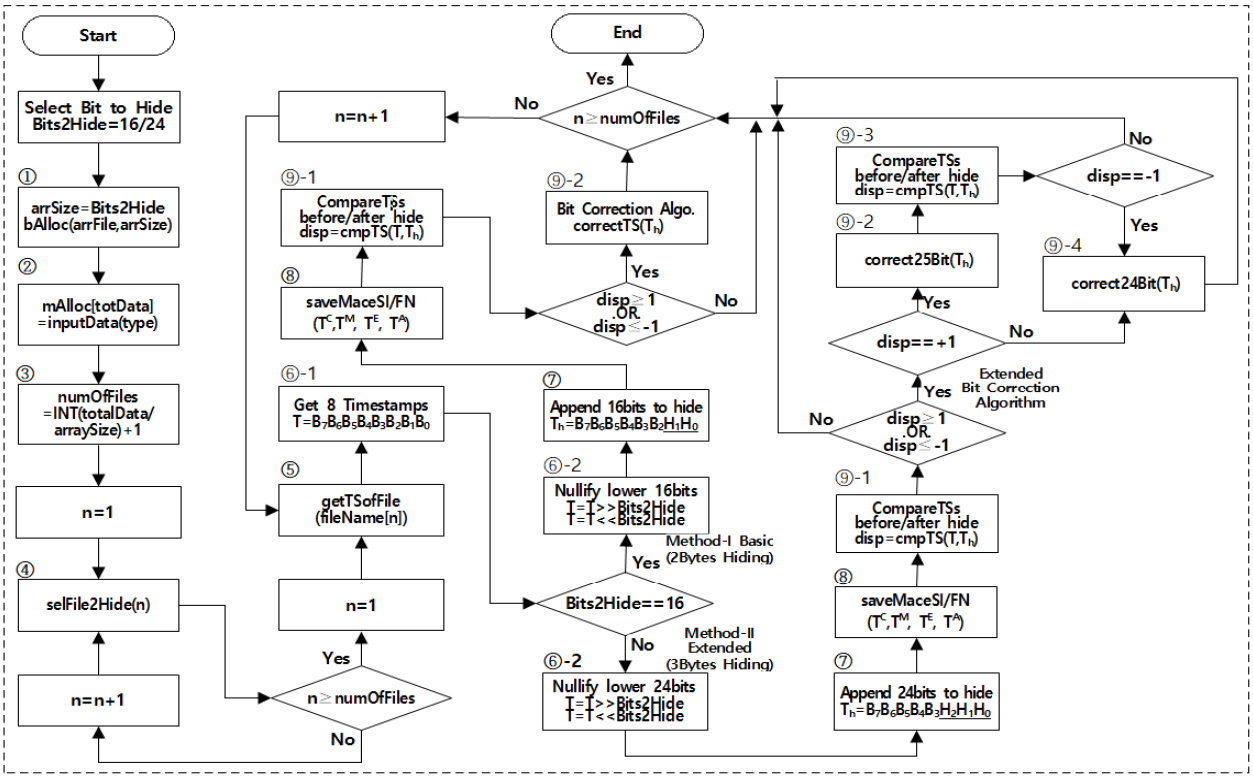

Fig. 2. Procedure of File Data Hiding in an Index Record

증가되면 증가된 시간만큼 감소시키고, 시간이 감소하면 감소된 시간만큼 증가시키는 기능을 수행한다. 사용되는 영역이 1초미만의 영역이기 때문에 시간의 증가와 감소의 최대치는 각각 1초이다. 상위비트 17번째 비트의 값에 가감연산을 수행하여 BitCorrection 기능을 수행하는 것이다. disparity가 +1이거나 -1이면 다음 기능을 수행한다.

$$\text{disparity}=\text{compareTimestamp}(T,\ T_{hide}) \quad (13)$$

$$\text{correctTimestamp}(T_{hide}) \quad (14)$$

모든 숨기려는 파일에 대하여 compareTimestamp()를 수행하고 난 뒤 과정10으로 이동한다.

절차⑩ 모든 데이터 숨기기 과정을 종료한다.

3. 3-byte Data Hiding Method by Extended Bit Correction Technique

1개의 타임스탬프에 2바이트를 숨기는 방식은 16비트만 사용하므로 23비트 공간에서 여유 비트가 생긴다. 이것을 활용하기 위한 방법으로 3바이트를 저장하는 방식을 제안한다. 3바이트를 저장하면 24비트에서 약간의 데이터 넘침현상이 발생한다. 이것의 영향으로 데이터 숨기기 연산을 한 후에 1초가 가해지거나 1초가 감해지는 경우가 발생한다. 이 문제를 비트보정 기법을 도입하여 3바이트 데이터 숨기기를 위한 확장 비트보정(Bit Correction) 기법을 제안한다.

2절의 기본 알고리즘과 절차는 유사하지만 한 개의 타임스탬프에 숨기려는 데이터의 크기가 3바이트와 비트보정 기법에 차이가 있다. 한 개의 파일에 숨길수 있는 문자의 수는 최대 24바이트이다. 즉, arraySize=24가 된다. 다음의 절차는 기본 알고리즘과 절차①~⑤는 동일하다. 차이가 나는 부분 절차 절차⑥~⑨에 대해서만 설명하기로 한다.

절차⑥ $SI의 4개 타임스탬프 $T_{\$SI}^{C}$, $T_{\$SI}^{M}$, $T_{\$SI}^{E}$, $T_{\$SI}^{A}$와 $FN의 4개 타임스탬프 $T_{\$FN}^{C}$, $T_{\$FN}^{M}$, $T_{\$FN}^{E}$, $T_{\$FN}^{A}$에 대하여 하위 24비트에 데이터를 저장한다.

숨길 데이터 공간 확보 방법은 다음과 같다.

$$T = B_7B_6B_5B_4B_3B_2B_1B_0 \quad (15)$$

여기에 숨길 공간을 확보하기 위하여 다음의 비트 연산을 연속하여 수행한다.

$$T = T >> \text{Bits2Hide}$$
$$T = T << \text{Bits2Hide} \quad (16)$$

여기서 Bits2Hide=24이다. 이 연산을 마치고 난 후의 T는 0_2=0000 0000, 0_1=0000 0000, 0_0=0000 0000는 각각 0으로 비트가 채워진다

$$T_{bitsoff} = B_7B_6B_5B_4B_3\underline{0_20_10_0} \quad (17)$$

절차⑦ 숨기려는 데이터는 바이트 단위로 H_2, H_1, H_0로 표현된다. 이 세 바이트 데이터에는 각각 ASCII 문자 3개가 저장

된다. 절차⑥의 $T_{bitsoff}$와 숨기려는 3바이트 $H_2H_1H_0$를 합산하면 다음과 같이 구할 수 있다.

$$T_{hide} = B_7B_6B_5B_4B_3H_2H_1H_0 \quad (18)$$

절차⑧ Thide를 8개의 타임스탬프에 수행한 후 기록한다.

saveMaceSI($T_{\$SI}^{C}$, $T_{\$SI}^{M}$, $T_{\$SI}^{E}$, $T_{\$SI}^{A}$)
saveMaceFN($T_{\$FN}^{C}$, $T_{\$FN}^{M}$, $T_{\$FN}^{E}$, $T_{\$FN}^{A}$) (19)

절차⑨ 3바이트 비트보정 기법은 상위비트 24번째 비트와 25번째 비트의 값을 가감 연산하여 비트보정 기능을 수행한다. 데이터 감추기 이전과 이후의 타임스탬프의 비교는compareTimestamp()에 의하여 수행되고 시간차이 값 disparity를 구한다. 이 값에 따라서 correct24Bit() 그리고/또는 correct25Bit()를 수행하여 보정을 한다(Algorithm 1).

Algorithm 1. Bit Correction Method for Extended Method

```
1  disparity=compareTimestamp(T, Thide)          ⑨-1
2  if(disparity==+1)
3  |  correct25Bit(Thide);                       ⑨-2
4  |  disparity=compareTimestamp(T, Thide);      ⑨-3
5  |  if(disparity==-1)
6  |  |  correct24Bit(Thide);                    ⑨-4
7  |  else
8  |  |  goto END;
9  |  end
10 else
11 |  if(disparity==-1)
12 |  |  correct24Bit(Thide);                    ⑨-4
13 |  else
14 |  |  goto END;
15 |  end
16 end
```

25번째 비트의 값은 16진수 0x1000000으로 표현되고 10진수는 16,777,216이다. 이 비트 값을 보정하는 기능은 correct25Bit()이다. 24번째의 비트의 값은 16진수 0x800000으로 표현되고 10진수는 8,388,608이다. 이 비트값을 보정하는 기능은 correct24Bit()이다. 이것은 24번째 비트를 0→1 기능만 가능하다. 그러므로 correct25Bit()의 감소 기능과 함께 사용하여 비트보정을 하게된다.

IV. Experiments

1. Experimental Environment

이 연구 사례들은 다음 테스트 환경에서 작업이 수행된다.

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
Disk Editing Tool : WinHex Ver. 18.4
Timestamp Manipulating Tools: timestomp.exe
SetMace.exe

2. Case I: Method I-Basic Method

다음의 사례 1은 타임스탬프에 2바이트 데이터 숨기기 작업이다.
숨길 데이터 : DataHidingMethod
대상파일 : d:₩SetMace.exe

변경 전의 타임스탬프의 값은 다음과 같다. Fig.3은 이것의 스크린 캡처이다. 네모 칸의 위치가 데이터를 숨길 2바이트(16비트)이다. 여기서의 데이터들은 리틀엔디언(Little Endian)방식으로 저장되기 때문에 8바이트 중의 작은 값이 먼저 저장된다.

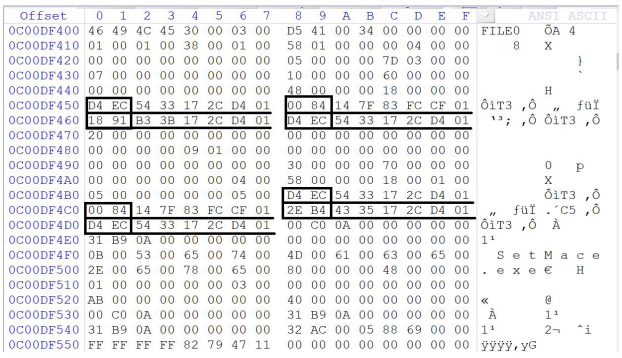

Fig. 3. Case I: Timestamps Before Data Hiding

Fig. 4. Case I: Timestamps After Data Hiding

데이터 감추기 후의 타임스탬프의 상태를 나타낸 것이다. Fig.4는 이것의 스크린 캡처이다. 각 타임스탬프마다 2바이트 공간에 ASCII문자의 코드를 입력하였다. 해당 문자들은 다음의 의미로 저장되었다.

$SI속성의 “15 61 74”에서와 $FN속성의 “15 65 4D”에는 비트보정 기법이 적용되었다. 보정 전에는 ‘14’이었으나 보정 후에 15’로 변경된 것이다. “2014년11월10일 01:13:12”이 원

래의 것이지만 데이터 감추기 후 “01:13:11”이 되어서 17번째 비트에서 올림하여 원래의 시간으로 보정된 것이다.

Fig. 5는 데이터 숨기기를 수행한 후의 “SetMace.exe“파일의 속성정보이다. 기존의 시간정보에 변함이 없다.

Table 1. Case I: Timestamps Before/After Data Hiding

Time-stamps		Before Data Hiding	After Data Hiding	Hide Chars
$SI	C	01 D4 2C 17 33 54 EC D4	01 D4 2C 17 33 54 61 44	a D
	M	01 CF FC 83 7F 14 84 00	01 CF FC 83 7F 15 61 74	a t
	E	01 D4 2C 17 3B B3 91 18	01 D4 2C 17 3B B3 69 68	i H
	A	01 D1 73 05 31 6D DC 79	01 D1 73 05 31 6D 69 64	i d
$FN	C	01 D4 2C 17 33 54 EC D4	01 D4 2C 17 33 54 67 6E	g n
	M	01 CF FC 83 7F 14 84 00	01 CF FC 83 7F 15 65 4D	e M
	E	01 D4 2C 17 35 43 B4 2E	01 D4 2C 17 35 43 68 74	h t
	A	01 D4 2C 17 33 54 EC D4	01 D4 2C 17 33 54 64 6F	d o

Fig. 5. Properties of “SeMace.exe“ After Data Hiding

3. Case II: Method II-Extended Method

다음의 사례2는 확장된 데이터 숨기기 방법이 적용된 것이다.

숨길 데이터

: Steganographic data hiding in timestamps

대상파일 : d:₩dataHiding₩targetFile.txt

데이터를 숨기기 전의 타임스탬프의 값은 표2의 좌측편에 나열되어 있고, 데이터 숨긴 후의 타임스탬프는 우측편에 나열되어 있다. 이 경우는 사례1의 경우와 달리 $FILE_NAME의 타임스탬프가 두 번 나타난다. 하나는 8.3 포맷의 짧은 파일명이고 다른 하나는 255자까지 사용할 수 있는 긴 파일명이다. 8.3 포맷의 사용 여부는 선택하여 설정할 수 있다.

이것은 레지스트리에서 다음과 같이 설정을 변경할 수 있다.

HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Control₩FileSystem에서 NtfsDisable8dot3NameCreation 값을 0또는 1로 설정하거나 명령프롬프트에서 “c:₩>fsutil behavior set Disable8dot3 0”을 실행하면 설정가능하다.

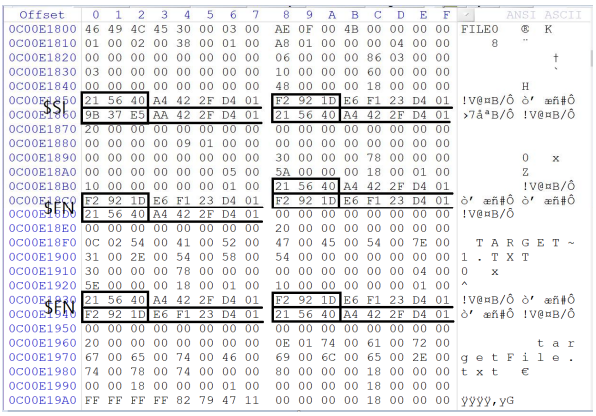

Fig. 6. Case 2: Timestamps Before Data Hiding

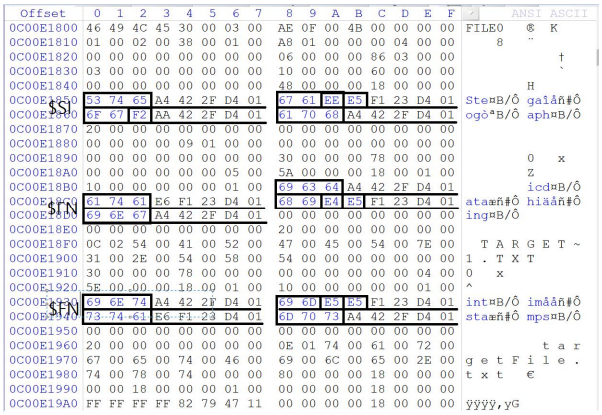

Fig. 7. Case 2: Timestamps After Data Hiding

Fig.6은 데이터를 숨기기 전의 타임스탬프의 스크린 캡처이다. 네모 칸의 위치가 데이터를 숨길 3바이트를 나타낸 것이다. Fig.7은 데이터를 감춘 후의 타임스탬프의 스크린 캡처이다. 각 타임스탬프마다 Table 2의 “Hide Chars”의 문자들을 숨긴 것이다.

비트보정 기법이 4개의 타임스탬프에 적용되었다. $SI속성의 수정시간에 데이터를 숨기고 보정하면 “01 D4 23 F1 E6 1D 92 F2”→“01 D4 23 F1 E6 6E 61 67”→“01 D4 23 F1 E5 EE 61 67”로 변경된다. 데이터 숨기고 나면 “2018/07/25 08:31:35”→“08:31:36”으로 변경되어서 보정이 필요하다. 25번째 비트를 0→1로 바꾸면 “E6”→“E5”로 변경된다. 이 때 “08:31:34”로 바뀌는데 원래의 시간과 차이가 나므로 “6E”의 상위비트(24번째 비트)를 0→1로 변경하면 “6E”→“EE”로 변경되면서 원래의 “08:31:35”로 보정된다.

$SI속성의 MFT엔트리 수정시간 “01 D4 2F 42 AA E5 37 9B”→“01 D4 2F 42 AA 72 67 6F”→“01 D4 2F 42 AA F2 67 6F”로 변경된다. 데이터를 숨기고 나면 “2018/08/08 18:07:27”→“18:07:28”로 변경된다. 이것을 보정하기 위해서 “72”의 상위비트를 1로 변경하여 “F2”로 바꾸면 “18:07:28"

→"18:07:27"로 보정된다. 이 경우는 24번째 비트만 1로 변경만 필요하고 25번째 비트에 대한 보정은 필요치 않다.

$FN(short)의 MFT엔트리 수정시간은 "01 D4 23 F1 E6 1D 92 F2"→"01 D4 23 F1 E6 64 69 68"→"01 D4 23 F1 E5 E4 69 68"로 변경된다. 데이터를 숨기고 나면 "2018/07/25 08:31:35"→"08:31:36"로 바뀐다. 이것에 대한 보정이 필요하다. 25번째 비트를 1→0로 바꾸면 "E6"→"E5"로 변경된다. 이 때 "08:31:34"로 바뀌는데 "64"의 상위비트(24번째 비트)를 0→1로 변경하면 "64"→"E4"로 변경되면서 원래의 "08:31:35"로 보정된다.

$FN(long)의 수정시간은 "01 D4 23 F1 E6 1D 92 F2"→"01 D4 23 F1 E6 65 6D 69"→"01 D4 23 F1 E5 E5 6D 69"로 변경된다. 데이터를 숨기고 나면 "2018/07/25 08:31:35"→"08:31:36"로 바뀐다. 이것에 대한 보정이 필요하다. 25번째 비트를 1→0로 바꾸면 "E6"→"E5"로 변경된다. 이 때 "08:31:34"로 바뀐다. "65"의 상위비트(24번째 비트)를 0→1로 변경하면 "64"→"E4"로 변경되면서 원래의 "08:31:35"로 보정된다.

Table 2. Case 2: Timestamps Before/After Data Hiding

Time-stamps		Before Data Hiding	After Data Hiding	Hide Chars
$SI	C	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 65 74 53	etS
	M	01 D4 23 F1 E6 1D 92 F2	01 D4 23 F1 E5 EE 61 67	nag
	E	01 D4 2F 42 AA E5 37 9B	01 D4 2F 42 AA F2 67 6F	rgo
	A	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 68 70 61	hpa
$FN short	C	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 64 63 69	dci
	M	01 D4 23 F1 E6 1D 92 F2	01 D4 23 F1 E6 67 6E 69	ata
	E	01 D4 23 F1 E6 1D 92 F2	01 D4 23 F1 E5 E4 69 68	dih
	A	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 67 6E 69	gni
$FN long	C	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 74 6E 69	tni
	M	01 D4 23 F1 E6 1D 92 F2	01 D4 23 F1 E5 E5 6D 69	emi
	E	01 D4 23 F1 E6 1D 92 F2	01 D4 23 F1 E6 61 74 73	ats
	A	01 D4 2F 42 A4 40 56 21	01 D4 2F 42 A4 73 70 6D	spm

V. Conclusion

이 연구에서는 기존의 타임스탬프에 데이터숨기기 방법에서 발견된 문제점을 해결하기 위한 비트보정 방법을 제안하였다. 1초 미만의 시간을 나타내는데 사용하는 공간에 2바이트의 데이터를 숨길 수 있지만 간혹 1초의 시간이 잘못 표시되는 문제가 발생하는 점에 대한 비트보정 기법을 제안하였다. 또한 숨길 데이터의 용량을 증가시키기 위해 3 바이트 데이터 숨기기 방법과 함께 확장 비트보정 기법을 제안하였다. 이 두가지 방법을 적용한 사례들에 대한 적용예를 통해서 제안된 방법의 타당성을 입증하고 데이터가 숨겨진 사실이 노출되지 않는 스테가노그래피적 방식의 데이터 숨기기 방법이라는 것을 입증하였다.

이 연구는 사례 적용은 타임스탬프 변조도구 툴과 디스크 편집 도구를 사용하여 수행되었다. 이 방법은 시간이 많이 소요되기 때문에 많은 데이터를 숨기기 위한 방법으로는 적절치 않다. 후속과제로써 타임스탬프에 데이터 숨기기 도구를 개발하는 과제가 수행되어야 하고 대용량의 데이터를 숨길 수 있는 방법에 대한 연구가 수반되어야 한다. 또한, 이 연구에서는 2바이트와 3바이트 데이터 숨기기를 위한 비트보정 기법을 제안하였는데 2바이트 숨기기 방법에서의 발생 횟수는 매우 적었지만 3바이트 기법에서는 좀 더 자주 발생한다는 사실을 경험적으로 알고 있다. 얼마나 많은 횟수 또는 확률로 이 기법이 적용되는지 분석을 하지 못한 부분도 추후의 연구과제로 제시한다.

REFERENCES

[1] N. A. Hassan and R. Hijazi, "*Data Hiding Techniques in Windows OS*", Elsevier, 2017.
[2] Ewa Huebner, Derek Bem and Cheong Kai Wee, "Data hiding in the NTFS file system,", Digital Investigation, Vol. 3, Issue 4, pp. 211-226, Dec. 2006.
[3] 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.
[4] 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.
[5] 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.
[6] Neuner, S. et al., "Time is on my side: steganography in filesystem metadata," Digital Investigation, Vol. 18, Supplement 7, pp. S76-S86, Aug. 2016,
[7] Thomas Gobel and Harald Baier, "Anti-forensics in ext4: On secrecy and usability of timestamp-based data hiding," Digital Investigation, Vol. 24, pp. s111-s120, Mar. 2018.
[8] B. Carrier, "*File System Forensic Analysis*", Addison-Wesley, pp. 273-396, Mar. 2005
[10] INFO: Working with the FILETIME Structure, https://support.microsoft.com/en-us/help/188768/info-working-with-the-filetime-structure
[11] G.-S. Cho, "A Computer Forensic Method for Detecting Timestamp Forgery in NTFS," Computer & Security, Vol. 34, pp. 36-46. May. 2013.
[12] Wicher Minnaard, "Timestomping NTFS," IMSc final research project report, University of Amsterdam,

Faculty of Natural Sciences, Mathematics and Computer Science, Jul. 2014.
[13] Metasploit Anti Forensics Project, http://www.metasploit.com/research/projects/antiforensics/
[14] SetMace, “https://github.com/jschicht/SetMace”
[15] Chu Luo et al., "A Data Hiding Approach for Sensitive Smartphone Data," DOI: 10.1145/2971648. 2971686, Sep. 2016.
[16] T. Knutsson, "Filesystem Timestamps: What Makes Them Tick?," SANS Institute InfoSec Reading Room, pp. 1-27, Mar. 2016.

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.