Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.2, Apr. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.2.387

삭제되거나 손상된 이벤트 로그(EVTX) 파일 복구 기술에 대한 연구*

신 용 학,[1†] 전 준 영,[2] 김 종 성[1,3‡]
[1]국민대학교 금융정보보안학과, [2]행복마루 컨설팅, [3]국민대학교 수학과

Study on Recovery Techniques for the Deleted or Damaged Event Log(EVTX) Files*

Yonghak Shin,[1†] Junyoung Cheon,[2] Jongsung Kim,[1,3‡]
[1]Dept. of Financial Information Security, Kookmin University,
[2]HM Consultings, [3]Dept. of Mathmatics, Kookmin University

요 약

디지털 기기의 사용이 늘어나면서 저장매체에 남아있는 각종 디지털 정보를 분석하여 범죄 단서를 찾는 디지털 포렌식 기술이 나날이 발전하고 있다. 또한 디지털 포렌식 기술과 더불어 안티 포렌식 기술도 발전하고 있다. 안티 포렌식 기술 중 분석을 어렵게 할 목적으로 사용 흔적 삭제 도구를 이용하여 로그파일 또는 웹브라우저 흔적을 삭제하기도 한다. 만약 사이버 범죄 수사 시 삭제되거나 손상된 데이터가 수사진행에 중요한 단서가 될 수 있다면 삭제되거나 손상된 데이터에 대한 복구는 매우 중요하다. 현재까지 이벤트 로그를 이용하여 다른 파일이나 파일 시스템을 복구하는 방식에 대한 연구는 많이 진행되었으나 이벤트 로그 자체를 복구하는 방법에 대한 연구는 미흡하다. 본 논문에서는 삭제되거나 손상된 이벤트 로그(EVTX) 파일의 복구 알고리즘을 제안하고, 실험을 통해 제안한 알고리즘의 높은 복구율을 확인한다.

ABSTRACT

As the number of people using digital devices has increased, the digital forensic, which aims at finding clues for crimes in digital data, has been developed and become more important especially in court. Together with the development of the digital forensic, the anti-forensic which aims at thwarting the digital forensic has also been developed. As an example, with anti-forensic technology the criminal would delete an digital evidence without which the investigator would be hard to find any clue for crimes. In such a case, recovery techniques on deleted or damaged information will be very important in the field of digital forensic. Until now, even though EVTX(event log)-based recovery techniques on deleted files have been presented, but there has been no study to retrieve event log data itself, In this paper, we propose some recovery algorithms on deleted or damaged event log file and show that our recovery algorithms have high success rate through experiments.

Keywords: Digital Forensic, EVTX(Event Log), Carving, Chunk, Event Record, Recovery Techniques

Received(02. 03. 2016), Modified(03. 10. 2016), Accepted(03. 15. 2016)

* 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2013R1A1A2059864).

† 주저자, sinyoohag@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

I. 서 론

안티 포렌식은 디지털 포렌식 기술에 대응하여 증거물을 훼손하거나 차단하는 일련의 행위로써 탐지 회피, 정보 수집 방해, 디지털 포렌식 도구의 오작동, 실행 흔적을 발견하지 못하도록 기록을 삭제하는 등, 법정 보고서나 증언으로 가치가 없도록 증거를 훼손시키는데 목적을 둔다. 그 중, 사용흔적 삭제 도구는 분석을 어렵게 할 목적으로 웹 브라우저 사용 흔적, 로그파일, 최근 실행 파일을 자동으로 삭제 시키는 도구로써 이러한 도구를 사용할 경우 디지털 포렌식 분석에 차질이 생길 수 있다.

사이버 범죄 수사 시 삭제된 데이터가 개인 및 집단에 의미가 있는 파일이거나 어플리케이션, 실행 정보 등 중요한 정보일 수 있기 때문에 복구에 실패한다면 수사에 난항을 겪을 수 있다. 이에 대비한 데이터 복구 기술은 지속적으로 연구되어 왔지만 이벤트 로그(EVTX)에 대한 복구 기술 연구는 미흡한 상황이다.

EVTX(Windows XML Event Log)는 Microsoft에서 사용하는 Event Log File System으로써 이전버전(MS Windows 2000, XP, 2003)에서는 EVT(Windows Event Log)를 이용한다. EVTX는 이벤트 로그 관리를 하며, Vista 이후에는 전역 로그와 응용 프로그램 및 서비스 로그라는 두 가지 범주의 이벤트 로그가 있다. 전역 로그의 범주에는 이전 버전의 Windows에서 사용할 수 있는 설치, 보안, 시스템 로그가 포함되며, 응용 프로그램 로그에는 응용 프로그램이나 프로그램에서 기록된 이벤트가 포함된다. 각종 이벤트 로그는 컴퓨터를 사용할 때, 시스템 어플리케이션의 작동 상태 및 사용자의 행위에 따라 보안, 설치, 실행, 관리, 작업, 분석, 디버그 할 경우에 남는 기록이다. 본 논문은 파일시스템의 비할당 영역에서 EVTX 파일을 복구하기 위해 파일 구조를 분석하고 파일의 손상 여부에 따라 조각 난 경우 재조립하여 복구하는 기법을 알고리즘과 함께 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 배경 및 관련연구를, 3장에서는 EVTX 파일 구조를 설명한다. 4장에서는 EVTX 검증 방법과 복구 방식에 대해 제안하며, 5장에서는 제안한 복구 알고리즘에 대한 실험 결과를 보이고, 6장에서 결론을 맺는다.

II. 배경 및 관련 연구

사용자의 부주의나 오작동 또는 의도적인 손상으로 인하여 필요한 데이터를 잃는 경우가 발생한다. 이에 대비하여 데이터 복구에 대한 연구가 진행되어 왔다. 전통적인 데이터 복구 방법은 파일 시스템의 구조에 기반하며 메타 데이터를 이용하여 복구하는 방법이다. 이런 방법으로 다양한 파일시스템의 분석이 이루어졌고 복구 알고리즘이 개발되었다. 삭제된 파일의 메타 데이터를 참조하기 때문에 속도도 빠르고 복구율이 높지만 메타 데이터가 덮어 쓰일 경우 복구가 불가능하다.

이러한 단점을 극복할 수 있는 복구 방법으로 Signature 탐지를 하는 카빙 기법이 있다. 카빙(Carving) 기법은 메타 데이터가 없어도 복구가 가능하지만 파일의 종류마다 구조가 다르고 기록되는 데이터의 형태도 다르기 때문에, 유형별 고유한 복구 알고리즘이 필요하다. 또한, 파일의 데이터가 연속적이지 않는 블록이 할당되는 경우 복구가 어려우며 많은 시간이 소요된다는 단점이 있다[1]. 이는 파일의 단편화(Fragmentation)로 인해 발생할 수 있다. 단편화는 파일이 여러 개의 조각으로 나뉘어져 저장되는 것을 의미한다. 단편화의 발생 원인으로는 크기가 고정되지 않은 자원을 배분하는 과정에서 빈번한 자원의 할당과 해제로 인하여 발생한다.

이벤트 로그 파일을 복구하기 위해서는 여러 가지 검증 과정이 필요한데, 그 중 CRC 검증은 매우 중요한 요소로 작용한다. 이벤트 로그에서 사용되는 순환 검사 방법(CRC32)은 데이터 전송 시 데이터에 오류가 있는지를 확인하기 위한 체크 값을 결정하는 방식을 말한다. 데이터를 전송하기 전에 데이터의 값에 따라 CRC 값을 계산하여 데이터에 붙여 전송하고, 받은 데이터의 값으로 다시 CRC 값을 계산하여 두 값을 비교하고, 두 값이 다르면 데이터 전송 과정에서 오류가 발생했다는 것을 알 수 있다[2].

이벤트 로그 파일의 구조는 다양한 연구자에 의해 연구되었다. 2007년 Andreas의 발표[3] 이후, Joachim Metz에 의해 2014년에 발표된 "Windows XML Event Log"[4]에서 Chunk 이외에 이벤트 로그의 실질적인 내용이 들어있는 Record의 XML 구조를 소개하였다. 또한 [3, 4]에서는 이벤트 로그의 구조와 함께 삭제된 이벤트 로그 파일에 대한 복구 방법도 소개하였다. 하지만 이벤트 로그의 복잡한 구조와 다양한 손상방식 때문에 실제 환경에서의 효과적인 복구방식을 제안하지 못하였다. 본 논문에서는 파일시스템에서 이벤트 로그의 손상된 정도에 따른 효과적인 복구 알고리즘을 제안한다. 특히, 기존 연구에서는 비할당 영역 또는 슬랙 공간으로부터 카빙

된 이벤트 레코드에 대한 복구는 어렵다고 주장하였으나, 본 연구에서 제안하는 복구 알고리즘은 비할당 영역 또는 슬랙 공간에서도 이벤트 레코드를 복구할 수 있다.

또한, 2013년 박민수 등은 "Record File Carving Technique for Efficient File Recovery in Digital Forensic Investigation"을 발표하였다[5]. 위 논문에서는 EVTX 파일 카빙 기술을 제시하고, 이에 대한 실험 결과를 보였다. 하지만, 실험환경 및 파일이 공개되지 않아 본 논문에서 제안하는 복구 알고리즘과의 직접적인 비교는 불가하나, 본 논문의 알고리즘은 보다 다양한 검증요소를 포함시킴으로써 더 많은 삭제되거나 손상된 EVTX 파일을 획득할 수 있다.

III. EVTX 파일 구조

이벤트 로그는 컴퓨터를 사용할 때, 시스템 어플리케이션의 작동 상태 및 사용자의 행위에 따라 남는 기록이다. Vista 이후 운영체제에서 사용되는 EVTX(Windows XML Event Log)는 Windows에서 특정 이벤트 발생 시 생성되는 정보를 Event Record에 저장하며, 여러 Event Record가 모여 Chunk를 구성한다. 또한 다수의 Chunk가 모여 하나의 EVTX 파일을 구성한다. 즉, 한 개의 EVTX 파일은 File Header (4096-Byte)와 여러 개의 Chunk (65536-Byte)로 이루어져 있다. 각각의 Chunk 파일은 Chunk Header (512-Byte), 크기가 정해져 있지 않고 가변적인 다수의 Record와 Slack 영역으로 이루어져 있다[4].

3.1 EVTX 파일 헤더(File Header) 구조

파일 헤더의 Signature는 ElfFile(45 6C 66 46 69 6C 65 $00_{(16)}$)이며, First/Last Chunk Number에는 처음과 마지막 Chunk 번호가 저장되어 있다. Header Block Size는 헤더의 크기를 나타내며 Number of Chunk는 해당 EVTX 파일 내 모든 Chunk의 개수를 의미한다. 헤더 내 CRC32 값은 Signature부터 120-Byte를 입력하여 CRC 함수를 거쳐 생성된 값이 저장되어 있다.

이벤트가 발생 시, EVTX 파일은 Chunk에 순차적으로 이벤트 정보를 저장하며, EVTX 파일 헤더에 가장 먼저 저장된 Record가 있는 Chunk의 정보를 가지고 있다. 예를 들어, Fig. 1의 First Chunk Number가 0이고 Last Chunk Number가 99이고

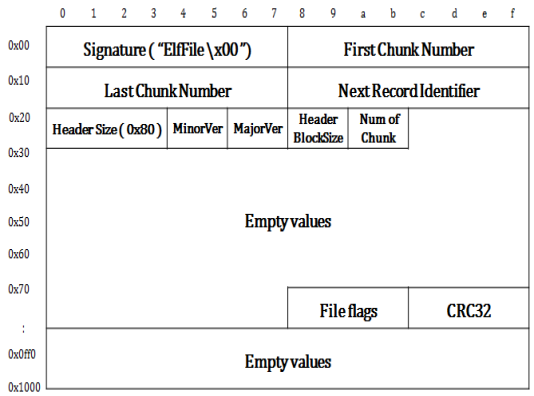


Fig. 1. EVTX File Header Structure

Num of Chunk가 100인 경우 Chunk의 순서는 0부터 시작하며 99까지 순차적으로 저장되어 있다는 것을 의미한다. 또한, First Chunk Number가 5이고 Last Chunk Number가 4라면 Chunk는 5부터 시작해서 Num of Chunk 만큼 순차적으로 저장되고 그 다음 0부터 시작해서 4까지 저장된다.

3.2 Chunk 헤더(Chunk Header)구조

Chunk 헤더의 Signature는 ElfChnk(45 6C 43 68 6E 6B $00_{(16)}$)이다. First/Last Event Record Number는 Event Record의 해당 Chunk 내에서 처음과 마지막 Event Record의 일련번호를 의미하고, Event Record Identifier는 Event Record에 기록된 일련번호이다. Last Event Record Offset은 해당 Chunk에 존재하는 여러 Record 중 마지막

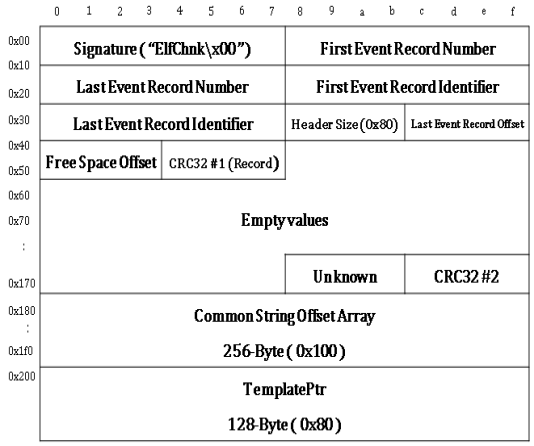


Fig. 2. Chunk Header Structure

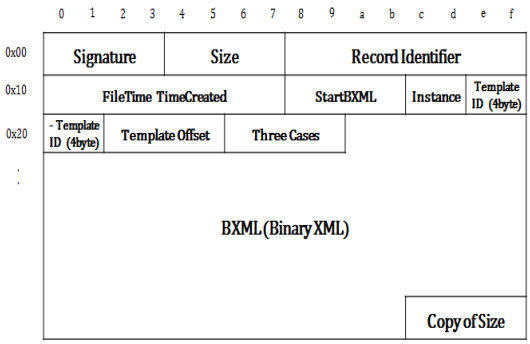


Fig. 3. Record Structure

Event Record의 시작 Offset값이 저장되어 있으며 Free Space Offset은 Slack 공간의 시작 Offset이다.

Chunk 헤더에는 두 개의 CRC32가 존재하여 데이터의 변조, 변경, 손상, 유실에 대한 검증을 수행한다. CRC32#1의 경우 Chunk의 모든 Event Record를 입력하여 생성된 값이며 Event Record의 손상 여부를 검증한다. CRC32#2는 Chunk 헤더의 512-Byte 중 Unknown 부분과 CRC32#2 부분을 제외한 나머지 부분을 입력하여 생성된 값이며, Chunk 헤더의 손상 여부를 검증한다.

3.3 레코드(Record) 구조

Event Record의 Signature는 (2a 2a 00 00 $_{(16)}$)이다. Signature 다음 4-Byte와 Event Record의 마지막 4-Byte는 해당 Event Record 크기를 의미한다. 만약 Record의 Size값이 30 06 00 00 $_{(16)}$이면 Record의 Size는 $0630_{(16)}$-Byte가 된다 (Intel 계열의 CPU는 리틀 엔디안 방식을 사용). 이 경우 Signature 시작점부터 $630_{(16)}$-Byte만큼이 Record의 데이터가 된다. Event Record Identifier는 Event Record의 일련번호가 저장되고 실제 이벤트와 관련된 정보는 StartBXML부터 시작하며 이 부분을 BXML이라 부른다. Fig. 4의 XML값은 Table.1과 같은 규칙으로 Binary 형태로 저장되어 있다. 각각의 BXML은 Template 구조로 되어 있고, 각각 Template마다 ID 값을 할당하여 관리한다.

Record의 BXML은 단편화된 구조로 되어 있다. 다양한 구조로 되어있기 때문에 일정한 규칙을 가지고 관리되는 이벤트 로그는 다음 Fig. 4와 같은 Microsoft Developers Network(MSDN)에서 공개된 이벤트 로그 포맷 형식을 따른다.

```
<Events>
   <Event>
      <System>
         <EventID>1</EventID>
         <TimeCreated SystemTime="2006-10-
           08T09:21:28.415Z"/>
         <EventRecordID>573</EventRecordID>
         …
      </System>
      <EventData>
      …
      </EventData>
   </Event>
</Events>
```

Fig. 4. Structure of an XML Scheme

Table 1. BXML Token Structure

Value	Meaning	Example
0x00	EndofBXmlStream	
0x01 0x41	OpenStartElementTag	〈name〉
0x02	CloseStartElementTag	〈name〉
0x03	CloseEmptyElementTag	〈name/〉
0x04	EndElementTag	**〈/name〉**
0x05 0x45	Value	attribute ="**value**"
0x06 0x46	Attribute	**attribute** ="value"
0x0c	TemplateInstance	
0x0d	NormalSubstitution	
0x0e	OptionalSubstitution	
0x0f	StartOfBXmlStream	

Token은 XML 구조를 표현하기 위하여 사용된다. XML의 Markup을 담당하는 Tag인 시작 태그(Start-Tag), 끝 태그(End-Tag), 빈 엘리먼트 태그(Element Empty-Tag)와 이름/값 짝으로 이루어진다. 시작 태그와 빈 엘리먼트 태그 속에 위치하는 Template의 시작을 나타내는 Token, 구조인 Attribute, 값을 나타내는 Value로 데이터를 정의하고 그 정의된 데이터를 관리하는 Substitution 구조가 두 가지로 표현된다.

데이터를 정의하는 부분은 다음과 같다. Record는 Data를 효율적으로 관리하기 위해, Template가 정의될 때 동시에 해당 문자에 대한 정보를 정의와 Substitution 형태로 아이디를 부여하여 관리한다. 문자에 대한 정의는 Table. 2와 같으며 Name Offset Definition의 값은 Chunk Header의

Common String Offset Array에 랜덤하게 기록된다. Data의 경우 문자의 방식이 Unicode이기 때문에 데이터의 길이는 2 * Length와 Eof를 나타내는 (00 $00_{(16)}$) 2-Byte로 이루어지며 해당 문자의 정보를 기록하기 위해 Substitution과 Template상에서의 문자의 OEM ID를 부여하고 이후 데이터를 표현할 때 해당 ID를 부여한 순서대로 사용한다.

데이터의 정의를 마치면 해당 데이터의 개수를 Count에 기록하고, 이후에 기록된 값만큼의 Size(2-Byte), Value(1-Byte), Eof(1-Byte)가 데이터 정의 부분 순서대로 기록된다. 해당 4-Byte 값이 Count만큼 기록이 되면 순서대로 Data에 해당하는 값이 Size와 Value에 맞게 표현되며 그 구조는 Fig. 5와 같이 표현된다.

이벤트 로그는 Record를 관리할 때 Template 구조로 값을 관리하며 그 형태는 크게 두 가지로 나뉜다. 첫 번째는 Template를 정의하는 부분이며 두 번째는 이전에 정의된 Template를 이용한 부분이다. 첫 번째 정의를 하는 부분은 Template Offset 이후 Three Case 값이 첫 번째 레코드인 경우 (00 00 00 $00_{(16)}$)으로 표현되며 이후 레코드는 2-Byte의 Offset 형태로 표현이 된다. 정의를 하는 부분의 특징은 Three Case 이후 4-Byte에 같은 Template ID가 반복됨으로 해당 Template가 정의되었음을 나타난다.

두 번째 경우인 Template가 이 전에 정의되어 있어 같은 값을 사용하는 경우 데이터 정의 부분은 생략되고 Three Case값에 Count값인 (12 00 00 $00_{(16)}$)이나 (14 00 00 $00_{(16)}$)가 저장된다. 이때 이전 Template에 해당하는 문자의 OEM ID가 부여되어 있으므로 이후 데이터를 표현할 때 순서대로 사용한다.

Table 2. BXML Data Name definition

Offset	Size	Value	Description
0	4		Name Offset Definition
4	4	00	Empty
8	2		Hash
10	2		Length
12			Data
…	1	0xOE	Substitution
…	2		OEM ID
…	1		Type

Data Definition (Name + Substitution)		
Count N		
Size 0 (2-byte)	ValueType 0 (1-byte)	0 (1-byte)
Size 1 (2-byte)	ValueType 1 (1-byte)	0 (1-byte)
…		
Size n-1 (2-byte)	ValueType n-1 (1-byte)	0 (1-byte)
DATA 0 (Size 0 byte)	DATA 1 (Size 1 byte)	
DATA 2 (Size 2 byte)		
…		
DATA n-1 (Size (n-1) byte)		

Fig. 5. Substitution Array Structure

Template가 정의되는 부분 중 Fig. 5의 Data Definition도 크게 두 가지로 나뉠 수 있다. 처음 XML의 Element를 정의하는 경우 해당 값에 대한 정의가 되어야 이벤트 로그 상에서 표현된다. 한 Chunk내에 위와 같은 값이 정의되었다면 또 다시 표현하는 것은 데이터의 낭비이므로 EVTX는 데이터 표현할 때 필요한 값인 Hash 값, 길이, 데이터, Substitution, Oem ID, Type을 처음 정의되는 문자의 Name Offset Definition만 사용하여 표현한다. 즉 Chunk내 첫 번째 Record에는 모든 문자가 정의되어 있다. 이후 Template를 새로 정의하는 Record에는 데이터를 정의할 경우 없는 것은 해당 값에 대한 정의를 하며 이전에 있는 것은 Name Offset Definition만 사용하여 길이를 절약하는 구조이다.

3.4 파일 시스템 상의 이벤트 로그 저장 방식

NTFS 파일시스템에서 이벤트 로그가 저장되는 특징은 이벤트 로그가 생성되는 경우 File Header와 Chunk가 생기고 각각의 이벤트 로그 파일에 해당하는 로그들이 발생할 때 해당 이벤트 Record에 기록이 되는 구조이다. 처음 할당된 EVTX파일은 Header(4096-Byte)와 한 개의 Chunk(65536-Byte)이다. 이후 Record가 누적되어 Chunk의 크기를 초과하면 두 번째 Chunk를 생성될 때 총 16개의 Chunk를 연속적으로 생성하게 된다. 즉, 1개의 Chunk는 16개의 클러스터를 차지하기 때문에 총

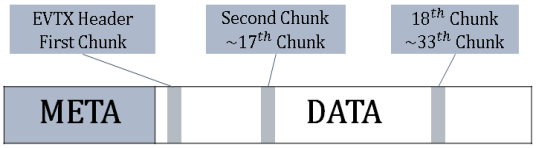


Fig. 6. Event Log Stored on the FileSystem

256개의 클러스터를 연속적으로 할당하여 16개의 연속된 Chunk를 생성한다. 이후 할당된 16개의 Chunk에 Record가 모두 생성되면 새로운 16개의 Chunk가 저장될 수 있는 곳에 Chunk 단위로 할당된다.

IV. EVTX 파일 복구 방법

EVTX 파일 안의 비트 열이 손상되거나 유실된 경우 파일 헤더만 손상되어 Chunk가 남아있거나 의미있는 Record가 이미지 상에 남아있다면 이를 복구하는 것은 매우 중요하다. 해당 영역에서 Signature를 이용하여 EVTX 파일의 복구 가능한 데이터를 선별하고, 결합하여 EVTX 파일을 복구한다. 본 장에서는 Chunk를 이용한 복구 방식과 Record를 이용한 두 가지 방식의 복구 알고리즘을 제안한다.

4.1 Chunk를 이용한 복구 방식

EVTX Header가 손상되어 구조가 완벽하지 않는 경우 조각나지 않은 Record들의 Chunk를 검증하여 찾는다. 검증 방법에는 EVTX의 Chunk 구조, Record의 구조를 검증하며 이후 파일을 복구하고 EVTX 파일을 재조립한다.

4.1.1 Chunk 단위 복구의 검증 방법

Chunk 헤더의 검증 요소는 Chunk 헤더의 Signature(45 6C 66 43 68 6E 6B $00_{(16)}$), Chunk 헤더 CRC32(#2), Chunk 헤더 CRC32(#1-Event Record)가 있다. Event Record의 검증 요소는 Event Record Signature(2a 2a 00 $00_{(16)}$), Size와 Copy of Size를 비교하는 방법이 있다.

4.1.2 적용된 복구 기법

Event Record는 크기가 가변적이므로, Event Record 중간에서 파일 단편화가 일어날 수 있다.

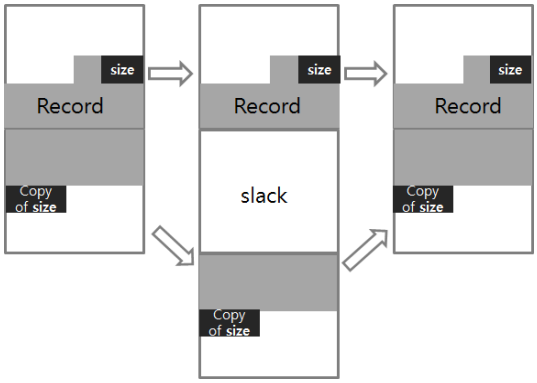


Fig. 7. Technique Recovery for Damaged File

Event Record가 중간에서 조각난 경우 Size와 Copy of Size의 저장 되는 장소가 조각난 부분에 위치할 수 있다. 따라서 Event Record를 재조립하기 위해 클러스터 단위로 조각난 데이터를 찾는다. 클러스터 단위로 조각난 데이터가 포함된 Copy of Size의 위치에 있는 조각난 Record의 Size 값과 동일 여부를 확인한다. 동일한 경우 해당 데이터를 하나의 Event Record로 재조립했을 때의 해당 CRC32 계산 값과 기존 Chunk 헤더에 있는 CRC32#1 값이 동일하면 유효한 데이터로 판단 가능하다.

4.1.3 복구 방법

파일 시스템의 특징에 따라 EVTX 파일이 조각나는 경우 해당 데이터의 복구 방법은 다음과 같다.

이미지 내의 Chunk Signature(45 6C 66 43 68 6E 6B $00_{(16)}$)를 검색한다. 해당 Chunk가 검색되면 그 안에 있는 Record의 손상 여부를 파악한다. Size와 Copy of Size를 비교하여 모은 유효한 Record의 CRC32값과 이전 검색된 Chunk에서 기록된 CRC32#1과 비교하여 같을 경우 다음 과정을 진행하고 다를 경우 Chunk Header에 알맞은 CRC32#1값으로 수정한다. 수정 시 검증 방법으로는 Chunk Header에는 Free Space Offset값이 Last Event Record Offset + Last Event Record Size를 더한 Offset과 일치해야 한다.

CRC32#1이 검증되면 Chunk Header 정보를 이용하여 CRC32#2를 검증하며 다를 경우 다시 계산한다. 이 경우 정상적인 Chunk이므로 EVTX Header를 해당 Chunk 정보에 맞게 값을 수정한다. Chunk 하나에 관한 내용이므로 First/Last

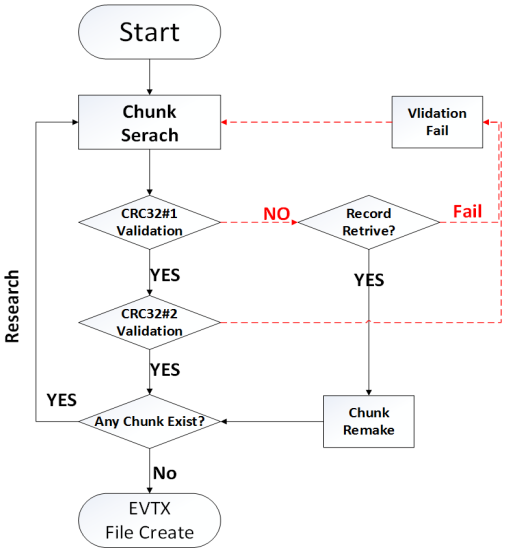


Fig. 8. Chunk Unit Recovery Algorithm

Chunk Number를 $00_{(16)}$으로, Number of Chunk를 $01_{(16)}$로 설정하고 EVTX Header의 CRC32를 구하여 EVTX Header를 구성하면 하나의 해당 Chunk를 복구 가능하다.

4.2 Record를 이용한 복구 방식

EVTX에서 Record는 BXML부분에 실질적인 내용을 담고 있다. 그래서 하나의 Record라도 의미 있는 데이터를 담고 있다면 해당 Record는 복구해야한다. Record 구조를 분석하여 복구 가능한 것만 모아서 Chunk와 File Header를 재조립한다.

4.2.1 Record 단위 복구의 검증 방법

Record 단위 복구의 검증 요소는 Event Record Signature(2a 2a 00 $00_{(16)}$), Size와 Copy of Size를 비교하는 것이다. Record 단위 복구에서 가장 중요한 부분은 한 Chunk 상의 첫 번째 Record를 찾는 것이다. 이유는 Template의 데이터가 한 번 정의되면 해당 값을 이용하기 때문이다. 첫 번째 Record를 찾고 이후 Record들에 대하여 관계 여부를 파악하여 같은 EVTX Chunk인지 비교한다.

Table 3. First 20 bytes in the first record

Offset	Size	Description
0	4	Size
4	2	Hash
6	2	Length
8	12	Data (“xmlns”)

4.2.2 적용된 복구 기법

이벤트 로그의 Chunk에서 첫 번째 Record만 가지고 있는 MagicNumber가 있다. 그 값은 데이터를 정의하는 값 중 xmlns(78 00 6D 00 6C 00 6E 00 73 00 00 $00_{(16)}$)이며 그 길이는 20-Byte이다. 무조건 첫 번째 Record에만 존재하며 다음 Record부터는 이 값을 받아와서 참조하게 된다.

첫 번째 Record의 Template와 두 번째 Record 이후의 Template를 정의하는 방식이 차이가 있어 Record들의 구별이 가능하다. BXML에는 다양한 종류의 Record를 구별할 수 있는 고유 정보를 가지고 있는데 이 값은 Record Signature로부터 0x61을 이동한 후 나오는 4-Byte에 고유정보의 크기가 나오게 된다. 이 부분에 첫 번째 Record에는 xmlns 값이 있고 아닌 경우는 없으므로 해당 값이 20-Byte 차이가 있다면 같은 Chunk라고 볼 수 있다. Three Case에 Count값이 나오는 경우에는 Template가 이 전에 정의되어 있어야 Record를 사용할 수 있으므로 Template ID가 이전 Record에 있는지 확인하여 있는 경우 같은 Chunk이다.

4.2.3 복구 방법

EVTX 파일에 손상되지 않는 Record들이 있는 경우 해당 Record들을 모아서 Chunk 단위로 복구한다. Chunk상의 첫 Record가 모든 정보를 가지고 있다고 해도 무방하므로 첫 Record를 찾는 것이 첫 번째 순서이다. Record를 검색하여 Size를 검증하고 xmlns 여부를 파악하여 첫 번째 Record를 찾는다. 이후 발견되는 Record에 대해 Template를 정의하는 경우에는 고유정보가 같은지 확인하며 Template ID 다음에 Count가 나오는 경우 Template ID가 이전 Record에 존재하는지 확인한다. 이후 성립하는 경우에 한하여 Record이후에 연결하고 다음 Record를 찾는다. 연결한 Record가 Chunk

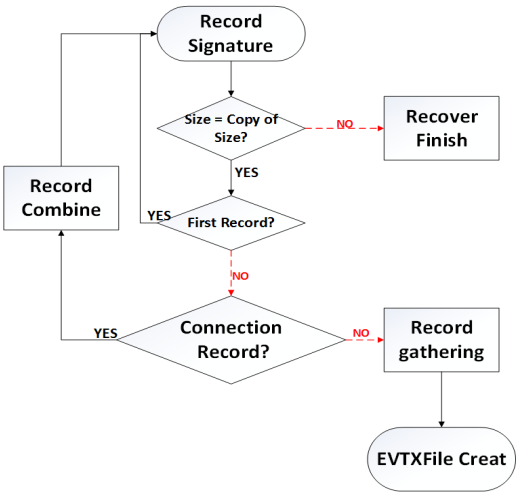


Fig. 9. Record Unit Recovery Algorithm

단위를 넘거나 새로운 첫 번째 Record가 나온 경우 그 동안 모은 Record로 Chunk를 생성한다. 이 경우 Chunk Header는 Fig. 2의 조건에 따른다. 이때 Common String Array나 TemplatePtr은 모두 00으로 해도 상관없다. Chunk를 생성하고 EVTX Header를 Fig. 1의 정보에 맞게 수정한다. Chunk 하나에 관한 내용이므로 First/Last Chunk Number를 00으로, Number of Chunk를 01로 설정하고 EVTX Header의 CRC32를 구하여 EVTX Header를 구성하면 해당 Chunk를 복구 가능하다.

4.3 제안하는 복구 방식에 대한 비교 분석

Table. 4에서 제안한 두 복구 방식에 대한 특징을 비교하였다. Chunk 단위의 복구는 파일 시스템에서 이벤트 로그가 Chunk 단위로 저장되는 특징을 이용하여 특정한 이벤트 로그가 손상이 되어도 정상적인 Chunk에 대하여 복구가 가능하며 클러스터 단위로 저장되는 파일시스템의 특성에 가장 어울리는 효율적인 복구 방식이라고 할 수 있다. Record 단위 복구 방식은 Header나 Chunk가 손상된 경우에도 Record만 남아 있다면 복구가 가능하다. 따라서 Record 단위 복구 방식은 이벤트 로그를 손상시키는 안티 포렌식 기법에 효과적인 대응 기술이라 할 수 있다.

V. 실험 결과

EVTX 파일 복구와 분석을 위해, Intel Core i7 3.60Ghz CPU, Windows7 pro, 16GB 메모리의 PC 환경에서 실험을 하였다. 제안한 방법은 파일 시스템 상의 이벤트 로그 저장 방식에 따라 할당되는데 그 단위는 Chunk로 저장한다. 따라서 Chunk 단위로 복구할 경우 고의로 손상하지 않는 경우에는 대부분이 복구 가능하다.

한국 디지털 포렌식 학회에서 2014년 실시한 디지털 포렌식 챌린지[6]에 공개된 테스트용 공개 덤프 이미지로 테스트한 결과 각 파일의 손상 정도에 따라 복구율이 다르긴 하지만 높은 비율로 복구가 되었다. Table. 5는 앞서 제안한 Chunk를 이용한 복구 방식으로 공개용 데이터를 복구한 결과이다. Table. 5의 파일의 개수는 공개용 데이터 덤프 이미지에 포함된 Chunk Signature(45 6C 43 68 6E 6B $00_{(16)}$)의 개수이고 복구 파일 개수는 본 논문의 방법으로 복구한 EVTX 파일의 형태로 복구에 성공한 Chunk의 개수이다. 본 방법은 EVTX Header에 영향을 받지 않고 Chunk에 따라 EVTX Header를 생성하는 구조이므로 EVTX Header를 이용한 복구 방법에 비해 더 좋은 결과를 얻을 수 있었다.

마찬가지로 Record를 이용한 복구 방식으로도 이와 유사한 결과를 얻었으나 Chunk 복구 방법보다

Table 4. Comparison of Chunk Unit Recovery and Record Unit Recovery Algorithms

	Chunk Unit Recovery Algorithm	Record Unit Recovery Algorithm
Verification	Chunk Signature, Chunk Validation, Record Signature Record Validation	Record Signature Record Validation
Recovery Method	The Above Verification is used to create an EVTX file with the Verified Chunk	The Above verification is used to create an EVTX file with a Chunk having all the verified record

Table 5 2014 Digital Forensic Challenge Test

	Our Recovery Method		
	Number of Files	Number of recovered files	Recover Rate(%)
Data #1	1515	1356	89.50
Data #2	859	859	100
Data #3	832	665	79.93

EVTX 파일이 복구되는 개수가 적은 결과를 얻었다. 이 차이는 이벤트 로그파일이 존재하지만 해당 이벤트가 발생하지 않아 Record가 없는 EVTX 파일이 존재하여 해당 결과를 보였다. 원 이미지의 구조가 완벽한 메타 데이터가 '삭제'인 경우 Chunk 옵션의 복구가 효율적이며 사용자가 이미지를 고의적으로 손상 시킨 경우에는 Record를 이용한 복구 방식이 효율적이다.

제안한 복구 방식을 이용하여 비할당 영역을 복구할 경우에도 효율적인 결과를 얻었다. 비할당 영역이란 디스크에서 삭제된 영역을 뜻하는데 삭제된 데이터가 단편화된 상태로 존재한다[7]. 데이터 파편의 경우 Random 특성을 가지기 때문에 Signature 분석과 데이터 파편 분석 방법으로는 의미 있는 정보를 얻지 못한다. 하지만 Fig. 6과 같이 EVTX는 저장 시 각각 연속된 Chunk 단위로 저장되는 EVTX 특징에 따라 클러스터 단위로 비할당 영역을 모으고 데이터 파편을 분석하여 복구한다.

본 논문에서 제안한 두 복구 알고리즘의 성능을 비교하기 위해 Fig. 10과 같이 (손상된) 다양한 이벤트 로그를 구성하여 테스트해 보았다. Fig. 10에서 각 이벤트 로그 파일을 각각 모양 (▨, ■, ▩)으로 표시하였다. ▨으로 표시된 이벤트 로그 파일(Normal)은 삭제되었지만 손상되지 않는 상태를 나타내며, ■으로 표시된 이벤트 로그 파일(Damaged)은 Chunk 24 이후의 모든 Chunk가 손상된 경우를 나타낸다. 또한 ▩으로 표시된 이벤트 로그 파일(Empty)은 이벤트 로그를 활성화 시키지 않아서 빈 상태인 경우이다. 각각의 경우를 제안한 알고리즘으로 복구한 결과는 Table. 6과 같다. Table. 6의 첫 N, D, E는 Normal, Damaged, Empty 상태의 EVTX 파일을 의미하며, 뒤 C, R은 Chunk Unit Recovery, Record Unit Recovery를 각각 나타낸다. Normal 상태의 EVTX 파일은 모든 이벤트 로그의 Record에 대하여 복구 가능하다. Damaged 상태인 파일은 Record가 손상되지 않은 부분에 대하여 모두 복구 되며 Record가 손상된 경우 해당 Chunk의 첫 번째 Record의 손상 여부에 따라 EVTX 파일의 복구가 결정된다. Empty 파일에 대해선 Chunk Unit Recovery로는 복구 가능하지만 Record가 존재하지 않으므로 Record Unit Recovery로는 이벤트 로그가 복구되지 않았다.

Table 6. Various EVTX Recovery Results

Option	Recovery	#File headers	#Chunks
N - C	Possible	31	31
N - R	Possible	31	31
D - C	Possible	25	25
D - R	Possible	25 + @	25 + @
E - C	Possible	1	1
E - R	Impossible	-	-

VI. 결 론

본 논문에서는 손상된 EVTX에 대한 Chunk 단위의 복구 방법과 Record 단위의 복구 방법을 제안하였다. 본 연구에서 제안한 두 복구 알고리즘은 EVTX 파일 단편화에 따라 구조가 온전하지 않은 경우, 상황에 따라 사용 가능하다.

본 논문에서 제안한 Chunk 단위의 복구 방법은 파일 시스템의 저장 방식에 따라 관리되는 EVTX를 이미지에서 복구하거나, 이벤트 로그를 삭제한 경우,

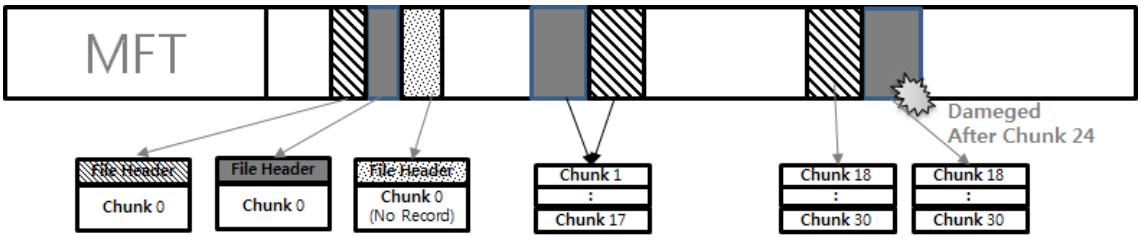


Fig. 10. Experimentally Used Various EVTX Files

복구에 효과적이며 Record 단위의 복구 방식은 이벤트 로그를 손상시키거나 Chunk가 손상된 경우에 효과적이다. 본 논문의 연구 결과는 디지털 포렌식 기술을 활용한 범죄 수사 시 삭제되거나 손상된 이벤트 로그 파일 복구에 유용하게 사용될 것으로 기대한다.

References

[1] S.J.J, Kloet , "Measuring and Improving the Quality of File Carving Methods," Master's thesis, Eindhoven University of Technology, Oct. 2007.

[2] P. Deutsch, "GZIP file format specification version 4.3," RFC 1952, May. 1996.

[3] Andreas Schuster, "Introducing the Microsoft Vista Event Log file format," DFRWS2007, pp. 65-72, May. 2007.

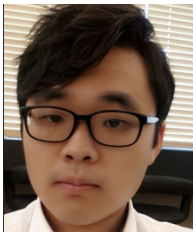
[4] Joachim Metz, "Windows XML Event Log (EVTX)," GitHub. "https://github.com/li by-al/libevtx/blob/master/documentation/Windows%20XML%20Event%20Log%20(EVTX).asciidoc", Feb. 2014.

[5] Minsu Park, " Record File Carving Technique for Efficient File Recovery in Digital Forensic Investigation," KIPS Transactions on Computer and Communication Systems. 2(2), pp. 93-10, Feb. 2013.

[6] 2014 Digital Forensic Challenge, http://kdfs.or.kr/journal_notice/3426, Nov. 2014.

[7] Binglong Li, Qianxian Wang, and Junuong Luo, "Forensic Analysis of Document fragment based on SVM," Proceedings of the 2006 IEEE Xplore Digital Library, pp. 236-239, DEC. 2006.

〈저자소개〉



신 용 학 (Yonghak Shin) 학생회원
2015년 2월: 국민대학교 수학과 졸업
2015년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식

전 준 영 (Junyoung Cheon) 정회원
2010년 2월: 세종대학교 컴퓨터공학과 졸업
2015년 2월: 고려대학교 정보보호대학원 석사
2012년 3월~현재: 행복마루 컨설팅(주)
〈관심분야〉 디지털 포렌식, 정보보호

김 종 성 (Jongsung Kim) 종신회원
2000년 8월/2002년 8월: 고려대학교 수학 학사/이학석사
2006년 11월: K.U.Leuven, ESAT/SCD-COSIC 정보보호 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2007년 3월~2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월~현재: 국민대학교 수학과 부교수
2014년 3월~현재: 국민대학교 일반대학원 금융정보보안학과 부교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식