디지털산업정보학회 논문지 제6권 제2호-2010년 6월

NTFS 파일 시스템의 저널 파일을 이용한 파일 생성에 대한 디지털 포렌식 방법

김 태 한* · 조 규 상**

A Digital Forensic Method for File Creation using Journal File of NTFS File System

Kim, Tae Han · Cho, Gyu Sang

〈Abstract〉

This paper proposes a digital forensic method to a file creation transaction using a journal file($LogFile) on NTFS File System. The journal file contains lots of information which can help recovering the file system when system failure happens, so knowledge of the structure is very helpful for a forensic analysis. The structure of the journal file, however, is not officially opened. We find out the journal file structure with analyzing the structure of log records by using reverse engineering. We show the digital forensic procedure extracting information from the log records of a sample file created on a NTFS volume. The related log records are as follows: bitmap and segment allocation information of MFT entry, index entry allocation information, resident value update information($FILE_NAME, $STANDARD_INFORMATION, and INDEX_ALLOCATION attribute etc.).

Key Words : Digital Forensics, NTFS Journaling, $LogFile, NTFS File System

Ⅰ. 서론

NTFS는 1990년대 초 Microsoft가 FAT 파일 시스템을 대체하기 위해 개발한 파일 시스템으로 Windows 2000, XP, Vista, 7 등에 탑재되어 있다. 이것은 파일 시스템에 대한 트랜잭션을 기록하고 있기 때문에 디스크 상의 에러에 대한 복구가 가능한 파일 시스템이다[1]. 데이터의 신뢰성을 높이기 위해 볼륨에서 수행되는 모든 작업에 대해 트랜잭션 단위로 기록함으로써 볼륨의 상태가 불안정하게 되지 않도록 해준다. 다른 시스템에서 사용되는 저널링 파일 시스템은 JFS, Ext3, ReiserFS 등[2]이 있다.

NTFS에서 사용하는 저널 파일은 $LogFile이다. 이 저널 파일의 구조에 대한 몇 가지 연구가 있었다. Singireddy의 문서[3]에서는 NTFS 로그 레코드 연산의 유

* 동양대학교 대학원 컴퓨터공학과 박사과정(제1저자)
** 동양대학교 컴퓨터정보전학과 교수(교신저자, 책임저자)

형들을 싣고 있다. 조[4]는 이 $LogFile의 로그 레코드에서 디지털 포렌식에 활용하기 위해 로그 레코드에서 연관된 일련의 정보를 추출하고, 파일의 복사와 삭제 연산에 대한 사례를 분석하여 $LogFile의 활용 가능성을 제시하였다. 또한 조[5] 등의 연구에서는 Set Bits In Nonresident BitMap의 Opcode 0x15와 Clear Bits In Nonresident BitMap의 Opcode 0x16 유형의 로그 레코드의 데이터 구조를 분석하여 그것의 자세한 구조를 밝혔다.

김[6] 등의 연구에서는 상주 속성 파일이 삭제될 때의 로그 레코드의 데이터를 분석하고, 파일의 삭제 트랜잭션이 발생했을 때의 사례 연구를 통해, $LogFile에 대한 디지털 포렌식을 진행하는 절차와 방법을 제안하였다.

본 연구는 $LogFile에 기록된 일련의 로그 레코드들이 파일 생성에 연관된 것인지와 어떤 내용이 기록되어 있는지 알기 위해 로그 레코드들의 구조를 역공학적 방법으로 밝히고, 파일 생성에 관련된 로그 레코드들을 추출하고 그 안의 데이터를 이용하여 어떤 작업이 수행되었는지 판단할 수 있는 디지털 포렌식 방법을 제안한다.

Ⅱ. NTFS 파일 시스템의 저널링

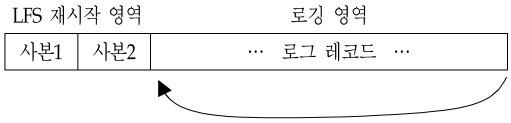
2.1 로그 파일 서비스(LFS)

NTFS 파일 시스템은 데이터를 저장하기 위해 MFT(Master File Table)를 사용하며, 볼륨 상의 모든 파일과 디렉토리들의 정보를 포함하고 있다. MFT 엔트리는 0번부터 26번까지는 시스템 파일에 예약되어 있으며, 볼륨 상에서 일반 파일이나 디렉토리들은 MFT 엔트리 27번부터 사용 가능하다. 이중에 MFT 엔트리 2번인 $LogFile은 볼륨에서 일어나는 트랜잭션 정보를 담는다. 이런 기술을 저널링 또는 로깅이라고 부른다.

NTFS 파일 시스템에서 MFT의 대부분의 구조와 역할은 알려져 있으나, $LogFile의 데이터 구조는 공개되어 있지 않다.

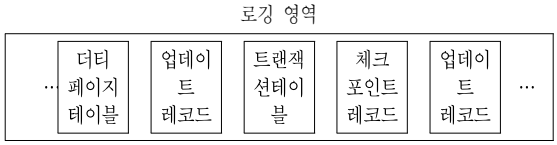
로그 파일 서비스(LFS: Log File Services)는 NTFS 드라이버 내 커널 모드 함수들의 집합체이며 NTFS가 로그 파일에 접근하기 위해 사용한다. 로그 파일의 구조는 4KB씩 2개의 재시작 영역과 로그 레코드가 기록되는 로깅 영역으로 구성되어 있다.

그림 1은 $LogFile의 영역구분을 나타낸 것이다. 이것은 비상주 속성 형식 형태의 $DATA 속성을 가지며, 재시작 영역과 로깅 영역으로 구성되어 있다. 로깅 영역에서는 볼륨 상에서 일어난 트랜잭션 정보를 저장하고 있으며, 재시작 영역은 마지막으로 수행되었던 로깅 영역을 가리키고 있다[8, 11].



<그림 1> $LogFile의 영역 구분

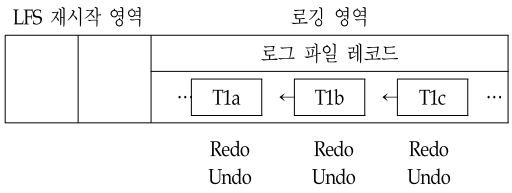
<그림 2>는 로깅 영역으로 LFS가 로그 파일에 저장하는 업데이트 레코드와 체크 포인트 레코드, 더티페이지 테이블, 트랜잭션 테이블[8]이 기록된 구조를 나타낸 것이다.



<그림 2> $LogFile의 로깅 영역[8]

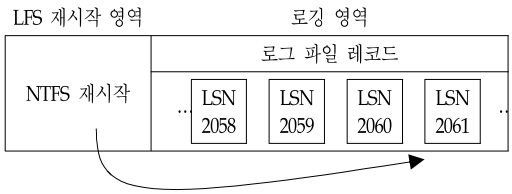
<그림 3>은 로그 파일의 로깅 영역에서의 업데이트 레코드를 나타낸 것이다. 로그 레코드는 볼륨에 대한 트랜잭션의 로그 기록은 완료되었으나 캐시로부터 변경 사항이 적용되기 전에 시스템 실패가 발생된 경우, 재작업에 필요한 Redo(재실행) 정보와 시스템 실패 시작 시점에 작업을 수행하지 못한 트랜잭션을 취소하고 이전 상태로 되돌리는데 필요한 정보인 Undo(취소) 정보를 포함하고 있다.

NTFS는 매 5초마다 로그 파일에 체크 포인트 레코드를 주기적으로 쓴다. 체크 포인트 레코드에 저장된 정보를 이용하여 NTFS는 복구를 위해 로그 파일 상의 어느 지점으로 돌아가야 하는지를 알 수 있다[8].



<그림 3> 로그 파일에서의 업데이트 레코드[8]

캐시에 있는 페이지들을 기록하고 있는 더티 페이지 테이블은 디스크에 미처 기록되지 않은 파일 시스템 구조에 대한 수정 사항들에 대한 정보를 가지고 있다. 트랜잭션 테이블은 트랜잭션 테이블 엔트리에 아직 커밋되지 않은 트랜잭션들을 저장한다. 트랜잭션 테이블 데이터 영역은 이것들의 번호를 포함하고, 나머지 부분은 테이블헤더에서 정의된 일련의 번호로 채워진다[8-9].



<그림 4> 로그 파일에서의 체크 포인트 레코드[8]

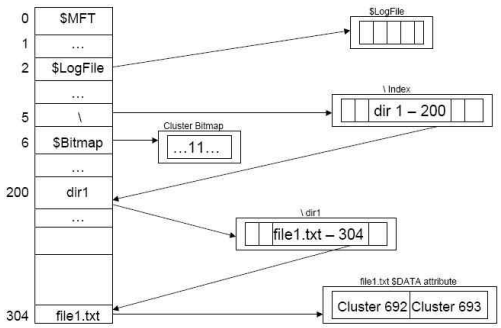
Singireddy의 문서[3]에서는 NTFS 로그 레코드 연산의 유형을 제시하고 있다. <표 1>의 로그 레코드 연산의 유형에 따라 NTFS가 $LogFile에 기록하는 정보는 서로 다르다. NFTS의 로깅 영역의 트랜잭션 테이블, 업데이트 레코드, 더티 페이지 테이블, 체크 포인트 레코드의 데이터가 $LogFile에 기록되어 있는 구조와 정보는 <그림 2>에 나타내었다.

2.2 파일 생성 과정

NTFS 볼륨 상에서 파일의 생성 후의 상태는 <그림 5>와 같으며, 디스크가 포맷된 후에 루트 디렉토리에 file1. txt가 볼륨 상에 추가된 상태를 나타낸 것이다.

파일의 생성 과정은 아래와 같은 과정을 거친다. Volume에서 MFT의 위치를 찾기 위해 boot sector를 읽는다. 이때 MFT의 시작 클러스터 주소를 파악한다. MFT 전체 구조를 알기 위해 MFT에서 첫 번째 엔트리를 읽는다. 생성된 파일을 위해 쓰고 있지 않은 새로운 MFT 엔트리를 할당한다. 다음으로 MFT 엔트리 내의 $STANDARD_INFORMATION 등의 속성을 생성하고 best-fit 알고리즘을 사용해서, 비어있는 클러스터를 찾기 위해 MFT 엔트리 6번 $Bitmap을 검사한다.

$Bitmap 파일 안에 비어있던 클러스터를 결정한 후, 플래그 값을 1로 바꾼다. 해당 클러스터에 파일 내용을 쓰고, 해당 파일의 MFT의 $DATA 속성에 데이터 런과 런의 길이를 기록한다. MFT 엔트리 5번 root directory (.)를 읽고, 인덱스 구조를 순회 (traverse)하고, dir1을 발견하고 MFT 엔트리 번호 200을 할당받았음을 나타낸다. dir1을 위해 $INDEX_ROOT 속성을 읽고, file1. txt를 위해 MFT 번호 304를 할당한다. 새로운 인덱스 엔트리를 생성하고 인덱스 트리에 정렬한다. 마지막으로 위의 각 단계를 파일 시스템에 문제가 발생하는 때, 오류 복구를 위해 저널 기록을 $LogFile에 저장한다[10].



<그림 5> 파일 생성 후의 상태[11]

<표 1> 로그 레코드 연산의 유형[3]

Opcode	NTFS LOG OPERATIONS
0x00	No operation
0x01	Compensation Log Record
0x02	Initialize File Record Segment
0x03	Deallocate File Record Segment
0x04	Write End Of File Record Segment
0x05	Create Attribute
0x06	Delete Attribute
0x07	Update Resident Value
0x08	Update Nonresident Value
0x09	Update Mapping Pairs
0x0A	Delete Dirty Clusters
0x0B	Set New Attribute Sizes
0x0C	Add Index Entry Root
0x0D	Delete Index Entry Root
0x0E	Add Index Entry Allocation
0x0F	Delete Index Entry Allocation
0x12	Set Index Entry Vcn Allocation
0x13	Update File Name Root
0x14	Update File Name Allocation
0x15	Set Bits In Nonresident BitMap
0x16	Clear Bits In Nonresident BitMap
0x19	Prepare Transaction
0x1A	Commit Transaction
0x1B	Forget Transaction
0x1C	Open Nonresident Attribute
0x1D	Transaction Table
0x1F	Dirty Page Table Dump
0x20	Transaction Table Dump
0x21	Update Record Data Root

Ⅲ. $LogFile의 포렌식 분석

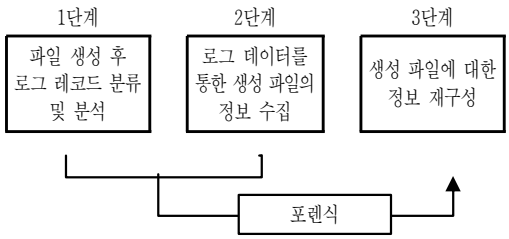
본 논문에서는 로그 레코드의 분석을 위해 <그림 5>의 파일의 생성 시 볼륨 상의 MFT 정보와 파일의 생성이 완료된 후에 $LogFile에 저장된 파일 생성에 관련된 로그 레코드들의 데이터를 역공학적인 방법으로 비교·대조하여 분석[4-6]한다.

로그 레코드의 데이터의 분석을 위한 도구는 X-Ways Software Technology AG사의 WinHex 15.4를 사용한다.

본 연구에서 제안된 포렌식 절차를 통해, $LogFile에 기록된 로그 레코드의 데이터에 대한 분석을 한다. 이를 위해 1단계와 2단계의 절차를 거친 후, 3단계에서는 분석한 데이터에 대한 정보를 재구성한다.

3.1 $LogFile의 로그 레코드 분석의 절차

$LogFile에 기록된 로그 레코드의 데이터 분석을 위한 생성된 파일에 대한 분석 절차는 <그림 6>과 같다.



<그림 6> 파일 생성 시, 로그 레코드 분석의 절차

USB 메모리를 NTFS 방식으로 포맷한 후, 루트 디렉토리에 윈도우 탐색기 창에서 마우스 오른쪽 버튼을 이용하여 파일을 생성한다. myfile.txt 파일이 볼륨 상에 생성되어, 볼륨에 일어난 변화가 $LogFile에 기록된다. 이 사례의 대상 파일은 파일명의 길이가 짧은 형식이므로 8.3 DOS 형식의 파일명을 갖는다.

<표 2> 대상 파일의 생성

분석 대상	NTFS로 포맷된 USB 메모리
파일 정보	위치 : 루트 디렉토리, 파일명 : myfile. txt
파일 형식	상주 속성 형식의 파일
생성 방법	윈도우 탐색기 창에서 마우스 오른쪽 버튼을 이용하여 데이터가 없는 파일명만을 생성

<그림 2>에서 알아본 바와 같이 $LogFile에 기록되는 로그 레코드들에는 여러 가지 종류가 있다. 이 중에서 포

렌식에 사용할 목적으로, 파일 생성에 관련된 로그 레코드들을 선별한다. 첫 번째 과정에서는 수많은 로그 레코드 데이터에서 로그 레코드의 유형에 따라 로그 레코드를 찾아 분류, 분석하는 절차를 거쳐야 한다.

2단계에서는 분류, 분석을 거친 로그 레코드에서 로그 Opcode에 대한 분석을 한다. 로그 레코드 데이터를 분석하여 Redo 데이터와 Undo 데이터 영역에 담긴 로그 레코드 데이터들에 대한 의미를 파악하는 과정이다. 이 과정에서는 로그 Opcode에 따라, 로그 레코드 연산만 수행하는 경우와 로그 레코드 데이터들을 담고 있는 경우에 대한 분석을 해야 하며, 인덱스 구조, MFT 엔트리 정보, $BITMAP 속성의 클러스터 런(Cluster Run)의 데이터에 대한 분석을 통해 오류 복구 시의 위치에 대한 정보를 추출해야 한다. 분석 과정을 통해 파일명의 형식, 시간 정보, 인덱스 구조에 관련된 $INDEX_ROOT 속성과 $INDEX_ALLOCATION 속성, $Bitmap 할당 정보를 통해 파일 시스템에 대한 전체적인 정보를 수집한다.

마지막 3단계에서는 이전 단계에서 얻은 로그 레코드들의 데이터들을 종합하여 디지털 포렌식 정보를 얻는 과정을 수행한다. 이 과정을 통해서 $LogFile의 로그 레코드 정보만으로 파일이 언제, 어디서, 어떤 내용으로 생성되었는지 판단한다.

3.2 $LogFile의 로그 레코드 분석

3.2.1. 로그 레코드의 기록 과정

<그림 7>과 <그림 8>은 트랜잭션 기록들이 $LogFile의 $DATA 속성에 로그 레코드를 기록하는 과정을 나타낸 것이다. <그림 7>은 myfile. txt 파일 생성 전의 볼륨 상태이며, <그림 8>은 파일이 생성된 후 $LogFile에 로그 레코드가 기록된 후의 클러스터 상태를 나타낸다. <그림 7>의 ①, ②, ③은 이전 트랜잭션을 기록한 로그 레코드를 나타내며, <그림 8>의 ④~⑦는 파일 생성 후의 추가된 로그 레코드를 나타낸다.

파일 생성 전, 볼륨 상에 있던 클러스터는 마지막 로그 레코드 기록이 ①, ②, ③의 순서로 이루어졌으며, 클러스터 1991번, 1992번과 1985번의 앞부분에 일부 로그 레코드 데이터를 기록하고 있다. 그리고 9986번 클러스터는 9998번 클러스터의 있는 로그 데이터의 사본을 기록한다. 마지막으로 재시작 영역에 체크 포인트의 LSN을 기록한다. 파일 생성 후 추가되는 레코드는 9993번 클러스터부터 순차적으로 저장이 된다.

LFS	재시작영역		로깅 영역						
클러스터 주소	9983	9984	9985	9986	9987	…	9991	9992	9993
클러스터	ⓐ	사본	③	③			①	②	

<그림 7> 로그 레코드 기록 전의 $LogFile의 상태

파일이 생성 된 후, LFS는 ③의 일부 로그 레코드 데이터를 9993 클러스터에 옮긴 후, 바로 뒤의 로깅 영역에 파일 생성과 관련되는 로그 데이터를 기록하고, 남은 일부 데이터는 로깅 영역의 앞 부분에 ⑦을 기록한다.

마지막으로 NTFS는 복구 시 검사점 설정을 위해 체크 포인트 레코드를 작성한 후, 재시작 영역 ⓐ에 체크 포인트의 LSN을 저장한다.

LFS	재시작 영역		로깅 영역								
클러스터 주소	9983	9984	9985	9986	9987	…	9991	9992	9993		
클러스터	ⓐ	사본	⑦	⑦					③	⑤	⑥
			④								

<그림 8> 로그 레코드 기록 후의 $LogFile의 상태

3.2.2 파일 생성에 관련된 로그 레코드 추출

파일의 생성 후, $LogFile의 로깅 영역에 추가된 로그 레코드 데이터를 대상으로 로그 레코드 연산(<표 1>) 유

형에 따라 분류하여, 로그 레코드를 추출한다. $LogFile에서 파일 생성과 관련된 로그 레코드 데이터를 추출하는 절차를 아래와 같이 3단계로 나누어 실시한다.

1단계 : 로그 레코드 데이터에서 로그 레코드의 분류
2단계 : Opcode의 로그 레코드 연산 순서의 추출
3단계 : 기능이 연결된 로그 레코드의 분류

$LogFile에서 파일 생성과 관련된 데이터를 추출한 후, <표 1>의 Opcode 유형별로 로그 레코드 데이터를 분류한다. 이때 사용된 로그 레코드의 분류 방법과 절차는 조규상과 김태한 등의 연구[4, 6]를 활용하였다.

파일 생성과 관련된 데이터에서 추출된 Opcode들은 본 연구에서는 Singireddy의 문서[3]의 16진수 표기법을 참고하여, "1B000100" 코드는 0x1B/0x01로 나타낸다. 또한 "07000700" 코드는 Update Resident Value를 의미하고, 0x07/0x07로 표기한다. 그리고 "14001400" 코드는 Update File Name Allocation을 의미하므로 0x14/0x14로 표기할 수 있다.

```
FF FF FF FF 82 79 47 11  C1 15 10 00 00 00 00 00
82 15 10 00 00 00 00 00  00 00 00 00 00 00 00 00
28 00 00 00 00 00 00 00  01 00 00 00 18 00 00 00
00 00 00 00 00 00 00 00  1B 00 01 00 28 00 00 00
28 00 08 00 18 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  B9 EB 4E 80 00 00 00 00
-----------------------------------------------------
CC 15 10 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  78 00 00 00 00 00 00 00
01 00 00 00 18 00 00 00  00 00 00 00 00 00 00 00
07 00 07 00 28 00 28 00  50 00 28 00 18 00 01 00
38 00 20 00 02 00 00 00  01 00 00 00 00 00 00 00
02 29 00 00 00 00 00 00  B2 F2 FB 82 14 EC CA 01
B2 F2 FB 82 14 EC CA 01  0C 55 FE 82 14 EC CA 01
06 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
30 EA B2 79 14 EC CA 01  30 EA B2 79 14 EC CA 01
30 EA B2 79 14 EC CA 01  06 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  E1 15 10 00 00 00 00 00
-----------------------------------------------------
CC 15 10 00 00 00 00 00  CC 15 10 00 00 00 00 00
98 00 00 00 00 00 00 00  01 00 00 00 18 00 00 00
00 00 00 00 00 00 00 00  14 00 14 00 28 00 38 00
60 00 38 00 44 00 01 00  00 00 98 04 00 00 00 00
00 00 00 00 00 00 00 00  8C 3D 00 00 00 00 00 00
30 EA B2 79 14 EC CA 01  B2 F2 FB 82 14 EC CA 01
B2 F2 FB 82 14 EC CA 01  0C 55 FE 82 14 EC CA 01
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

<그림 9> 파일 생성 후 추가된 데이터에서 로그 레코드의 분류

```
0x15/0x16 → 0x00/0x03 → 0x0E/0x0F → 0x0E/0x0F →
0x02/0x00 → 0x1B/0x01 → 0x07/0x07 → 0x14/0x14 →
0x1B/0x01 → 0x1D/0x00 → 0x1E/0x00 → 0x1F/0x00 →
0x00/0x00 → 0x0F/0x0E → 0x06/0x05 → 0x06/0x05 →
0x0F/0x0E → 0x05/0x06 → 0x0E/0x0F → 0x1B/0x01 →
0x07/0x07 → 0x14/0x14 → 0x1B/0x01 → 0x07/0x07 →
0x14/0x14 → 0x1B/0x01 → 0x1D/0x00 → 0x1E/0x00 →
0x1F/0x00 → 0x00/0x00 → 0x1D/0x00 → 0x1E/0x00 →
0x1F/0x00 → 0x00/0x00 → 0x1D/0x00 → 0x1E/0x00 →
0x00/0x00
```

<그림 10> Opcode 유형별 로그 레코드 연산 순서의 추출

<그림 9>의 데이터에서 분류한 Opcode의 연산 순서는 "0x1B/0x01 → 0x07/0x07 → 0x14/014"와 같이 나타낼 수 있다. <그림 9>는 파일 생성과 관련된 로그 레코드들의 일부분을 나타내고 있다. 실제 레코드는 많은 클러스터에 걸쳐 저장되어 있지만 지면 관계상 일부분을 나타낸 것이다.

이와 같은 방법으로 분류된 Opcode에는 체크 포인트 레코드, 트랜잭션 테이블, 더티 페이지 테이블들이 포함되어 있다. <그림 10>은 2단계 과정을 나타낸 것으로, 포렌식 과정에서 필요한 로그 레코드만을 선별하기 위한 과정이다. 데이터 분석에 직접적으로 필요치 않은 정보들(0x1D/0x00, 0x1E/0x00, 0x1F/0x00) 3가지 레코드들과 0x00/0x00은 레코더 헤더에 플래그 값 0x02를 갖는 체크 포인트 레코드[12]를 제외시킨다.

조규상[3]은 0x1B 로그 Opcode는 로그 레코드 뒤에 나오는 로그 레코드들은 앞에 나온 로그 레코드의 Previous LSN 값을 지정하지 않는다고 설명하며, 연관된 일련의 로그 레코드들의 끝을 알리는 것이라고 설명하고 있다. 이렇게 분류한 결과는 <그림 11>과 같이 5가지의 일련의 연관된 로그 레코드들로 분류가 된다.

3.2.3 역공학적인 방법으로 로그 레코드의 구조 분석

본 논문에서 다루는 로그 레코드는 데이터 구조가 공개되지 않아 <그림 5>의 NTFS 볼륨에서의 파일 생성 알고리즘 과정을 로그 레코드 분석 과정에 활용하고, WinHex 등의 툴을 사용하여 역공학적인 방법으로 데이

작업	연관된 로그 레코드		
로그 레코드들 1	0x15/0x16	0x00/0x03	0x0E/0x0F
	0x0E/0x0F	0x02/0x00	0x1B/0x01
로그 레코드들 2	0x07/0x07	0x14/0x14	0x1B/0x01
로그 레코드들 3	0x0F/0x0E	0x06/0x05	0x06/0x05
	0x0F/0x0E	0x05/0x06	0x0E/0x0F
		0x1B/0x01	
로그 레코드들 4	0x07/0x07	0x14/0x14	0x1B/0x01
로그 레코드들 5	0x07/0x07	0x14/0x14	0x1B/0x01

<그림 11> 기능이 연관된 로그 레코드

터를 분석한다. 다양한 상황에서 $LogFile에 기록된 로그 레코드의 데이터들 중에서 파일 생성에 관련된 로그 레코드들을 분석하여 포렌식에 사용하기 위한 정보를 얻기 위한 방법을 제시한다.

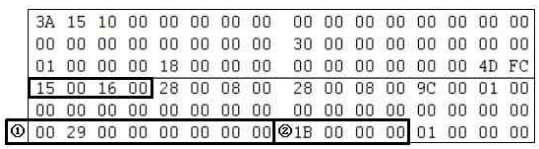
본 논문에서 로그 레코드의 구조체를 <표 3>와 같이 정의하여 기술한다.

<표 3> 로그 레코드의 구조

로그 레코드 헤더		
로그 레코드 데이터	로그 레코드 데이터 헤더	
	로그 레코드 데이터	Redo 데이터
		Undo 데이터

(1) 0x15/0x16 로그 레코드

0x15/0x16은 로그 레코드는 파일 생성과 관련된 데이터의 가장 먼저 생성되었으며, 0x15 로그 Opcode는 Set Bits In Nonresident BitMap을, 0x16 로그 Opcode는 Clear Bits In Nonresident BitMap[2]을 수행한다.



<그림 12> 0x15/0x16 로그 레코드의 구조

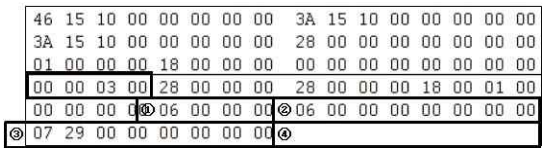
①은 클러스터 주소 값을 나타내며, 비상주 속성 파일인 $MFT의 $DATA 속성이 볼륨 상에 있는 위치 10496 클러스터 정보를 담고 있다. ②는 0x1B는 비어있던 MFT 할당 정보를 통해, 새로운 MFT 엔트리 번호로 27번을 할당하고, Redo와 Undo 데이터에 기록해 놓은 것이다.

<표 4> 0x15/0x16 로그 레코드의 분석

항목	로그 데이터가 담고 있는 정보	값
①	비상주 속성 파일인 $MFT의 $DATA 속성의 볼륨상 위치	10496 클러스터
②	Redo와 Undo 데이터	MFT 엔트리 27번

(2) 0x00/0x03 로그 레코드

0x00/0x03 로그 레코드에서 0x00 로그 Opcode는 No Operation을 의미하며, 0x03 로그 Opcode는 Deallocate File Record Segment를 나타낸다. 이 로그 레코드 연산은 VCN 0x06, 0x2907 클러스터로부터 오프셋 0x06의 위치에 MFT 엔트리를 할당된 후의 정보가 기록되어 있다.



<그림 13> 0x00/0x03 로그 레코드의 구조

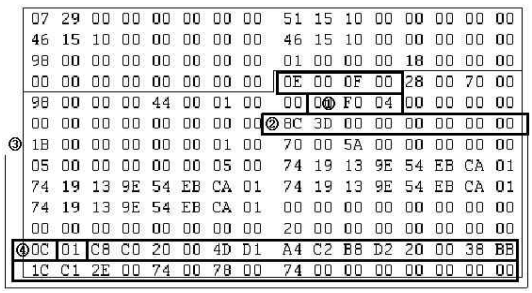
③은 VCN(Virtual Cluster Number) 정보를 나타낸다. 따라서 0x2907은 10503 클러스터 주소를 나타낸다. 0x2907 클러스터는 VCN 6번에 매핑되고, 오프셋 0x06 위치에 MFT 엔트리 27번이 위치하고 있다.

<표 5> 0x00/0x03 로그 레코드의 분석

항목	로그 데이터가 담고 있는 정보	값
①	10503 클러스터에서 오프셋	6
②	VCN 정보	VCN 6
③	클러스터 정보	10503 클러스터
④	Redo와 Undo 데이터 없음	

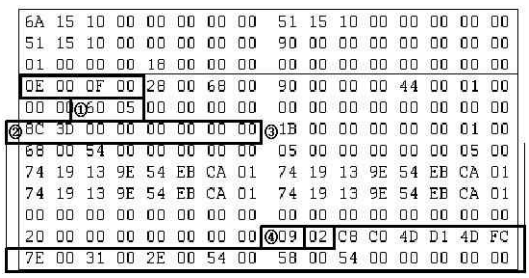
0x00/0x03 로그 레코드를 통해, 볼륨 상에서 파일이 생성된 위치를 파악할 수 있다. 부트 섹터는 0x30 오프셋의 데이터를 통해 MFT의 시작 위치를 파악한다. 따라서 <표 5>의 로그 레코드에 있는 VCN과 해당 MFT 엔트리까지의 오프셋 값을 통해 MFT 엔트리가 생성된 볼륨상의 위치를 정확히 담고 있다. 또한 ④는 오류 복구 시에 수행되는 복구 작업에 해당하는 로그 레코드의 내용은 정의되지 않으며 MFT 엔트리 할당을 해제하는 로그 레코드 연산만 수행한다.

(3) 0x0E/0x0F 로그 레코드



<그림 14> 0x0E/0x0F 로그 레코드 분석

0x0E/0x0F 로그 레코드는 인덱스 엔트리를 추가하고 삭제하는 로그 레코드 연산을 수행한다. 0x0E 로그 Opcode는 Add Index Entry Allocation을, 0x0F 로그 Opcode는 Delete Index Entry Allocation 로그 레코드 연산을 수행한다.



<그림 15> 0x0E/0x0F 로그 레코드 분석

0x0E/0x0F 로그 레코드 연산은 <그림 14>과 <그림 15>에서 보는 바와 같이 같은 0x0E/0x0F 로그 레코드가 두 번 연속 기록되어 있다. <표 6>과 같이 파일 시스템에서 DOS의 8.3 파일명 형식, Win32 파일명 형식 등 다양한 파일 형식을 다루기 위한 것이다. ①과 ②는 인덱스 구조에서 엔트리의 위치 정보를 나타내며, ①은 오프셋을 ②는 15756 클러스터 주소를 나타내며 $INDEX_ALLOCATION 속성의 인덱스 구조로 $RootDirectory를 가리킨다. ③은 생성된 파일에 대한 인덱스 엔트리의 정보를 Redo 데이터에 담고 있다. 파일 참조번호(File Reference Address)와 $FILE_NAME 속성을 담고 있다.

<표 6> 파일의 형식

파일 이름의 길이	파일 형식 값	파일명(hex)
0x0C	0x01(Win32)	C8 C0 20 00 4D …
0x09	0x02(DOS)	C8 C0 4D D1 4D …

<그림 15>의 0x09는 파일 이름의 길이가 9bytes임을 의미한다. 그리고 0x02는 DOS 형식의 파일 이름을 담고 있음을 의미하며, $FILE_NAME를 담고 있는 인덱스 엔트리가 Root Directory에 2개가 생성된다.

<표 7> 0x0E/0x0F 로그 레코드의 분석

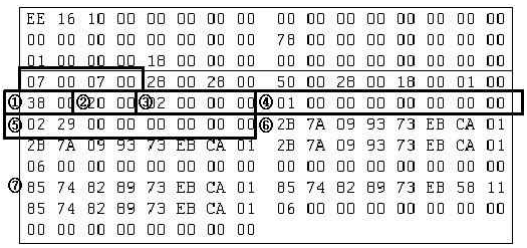
항목	<그림 14>의 로그 레코드	값
①	인덱스 엔트리까지의 오프셋	1264 bytes
②	클러스터 정보	15756 클러스터
③	인덱스 엔트리 정보	MFT 27번
④	파일 이름 정보	Win32 형식
항목	<그림 15>의 로그 레코드	값
①	인덱스 엔트리까지의 오프셋	1376 bytes
②	클러스터 정보	15756 클러스터
③	인덱스 엔트리 정보	MFT 27번
④	파일 이름 정보	DOS 형식

(4) 0x07/0x07 로그 레코드

0x07 로그 Opcode는 Update Resident Value를 나타

내며, 상주 속성 값을 업데이트하는 연산을 수행한다.

그림 16의 ①은 56bytes ②는 32bytes ③은 오프셋 위치 2를 나타낸다. ④는 $MFT에서의 VCN 1을 나타낸다. ⑤은 0x2902는 해당 파일이 있는 볼륨 상의 클러스터 주소를 나타낸다. 따라서, 0x2902, VCN 1, 오프셋 2의 위치는 MFT 엔트리 번호 5번 $MFT이며, 오프셋 56bytes, 오프셋 32bytes의 위치를 가리킨다. 즉, MFT 엔트리 5번의 위치를 가리킨다. ⑥은 Redo 데이터는 MFT 5번의 $STANDARD_INFORMATION의 시간정보 3가지와 (Modification, Record change, Last access time) Flag 값 6을 담고 있다. ⑦은 Undo 데이터를 나타내며, 담고 있는 데이터의 구성은 ⑥과 같다.



<그림 16> 0x07/0x07 로그 레코드의 구조

<표 8> 0x07/0x07 로그 레코드의 분석

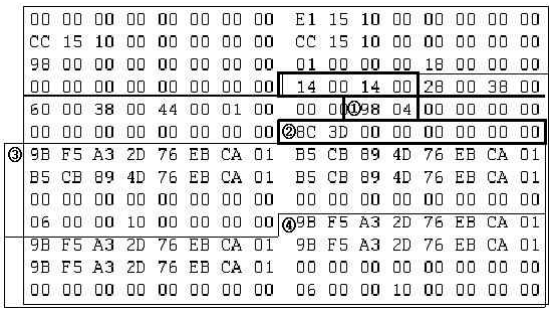
항목	로그 데이터가 담고 있는 정보	값
①	오프셋	56bytes
②	오프셋	32bytes
③	MFT 엔트리까지의 오프셋	2
④	VCN	1
⑤	클러스터	10498
⑥	Redo 데이터	$STANDARD_INFORMATION
⑦	Undo 데이터	

<표 8>에서 얻은 정보를 통해, MFT 엔트리 5번인 $MFT의 $STANDARD_INFORMATION의 속성에서 시간 정보가 갱신되었음을 알 수 있다.

(5) 0x14/0x14 로그 레코드

0x14/0x14에서 0x14 로그 Opcode는 Update Filename Allocation을 나타낸다. 이는 파일 이름 정보를 업데이트하는 로그 레코드 연산을 수행한다.

①은 루트 인덱스 노드에서 해당 파일이 있는 인덱스 엔트리까지의 오프셋을 나타낸다. ② 해당 인덱스 엔트리가 있는 루트 인덱스 노드의 클러스터 주소를 나타낸다. ③과 ④에는 MFT 5번.(Root directory)의 $DATA 속성에서 MFT 엔트리 5번에 대한 파일 생성 시간, 파일 수정 시간, MFT 수정 시간, 마지막 접근 시간에 시간 정보를 Redo 데이터와 Undo 데이터로 가지고 있다.



<그림 17> 0x14/0x14 로그 레코드의 구조

<표 9> 0x14/0x14 로그 레코드의 분석

항목	로그 데이터가 담고 있는 정보	값
①	인덱스 엔트리까지의 오프셋	1176 bytes
②	$INDEX_ALLOCATION 속성의 인덱스 레코드의 클러스터	15756 클러스터
③	Redo 데이터	$FILE_NAME 속성
④	Undo 데이터	

③과 ④의 데이터에는 MFT 엔트리 5번인 $MFT의 $FILE_NAME 속성을 담고 있다.

(6) 0x02/0x00 연산

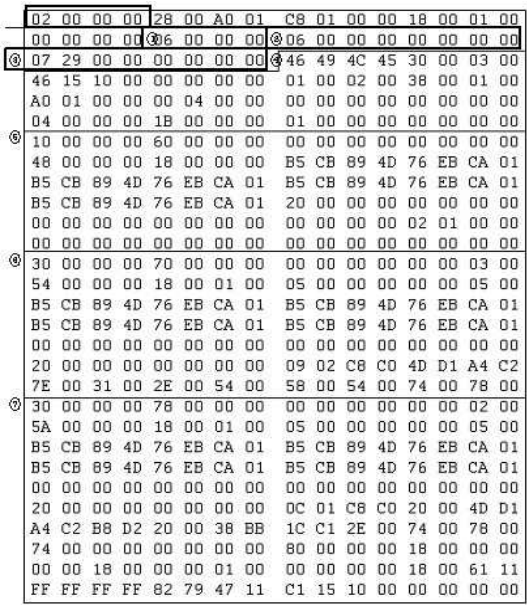
0x02/0x00 로그 레코드에서 0x02 로그 Opcode는 Initial File Record Segment를 나타낸다. MFT 세그먼트

를 초기화하는 로그 레코드 연산을 수행한다.

0x02 로그 Opcode의 Redo 데이터는 해당 클러스터의 VCN에 생성된 파일의 MFT 엔트리의 표준 구조를 포함하고 있다. 따라서 MFT 헤더와 0x10 $STANDARD_INFORMATION 속성 그리고 0x30 FILE_NAME 속성 구조를 가지게 된다.

<그림 18>의 ①은 오프셋 ②는 VCN 정보를 나타낸다. ③는 0x2907은 클러스터 주소를 나타내며, 0x2907 클러스터의 VCN 6에 매핑되는 클러스터에서, 오프셋 6의 위치는 MFT 27번을 가리킨다.

0x02/0x00 로그 레코드를 분석함으로써, 포렌식이 대상이 되는 생성 파일의 MFT 엔트리에 대한 전체 정보를 얻을 수 있다. ①, ②, ③ 에서는 생성된 파일의 MFT 엔트리의 위치 정보를 얻을 수 있다. ④는 생성된 파일의 MFT 헤더를 나타낸다. ⑤는 $STANDARD_INFORMATION 속성을, ⑥과 ⑦은 $FILE_NAME 속성을 담고 있다. 파일명 형식에 따라 ⑥은 DOS 형식의 파일명을 ⑦은 Win32 형식의 파일명을 담고 있다.



<그림 18> 0x02/0x00 로그 레코드의 구조

<표 10> 0x02/0x00 로그 레코드의 분석

항목	로그 데이터가 담고 있는 정보	값
①	MFT 엔트리까지의 오프셋	VCN 6
②	VCN	6
③	클러스터	10503 클러스터
④	생성 파일의 MFT 헤더	
⑤	$STANDARD_INFORMATION 속성	
⑥	$FILE_NAME 속성	DOS파일형식
⑦	$FILE_NAME 속성	Win32파일형식

볼륨의 MFT 엔트리 정보를 활용하지 않고, $LogFile의 기록된 로그 레코드의 정보만을 활용하여 볼륨 상에서 생성된 파일에 대한 정보를 수집할 수 있었다.

$LogFile의 로그 레코드의 데이터 분석대상은 <그림 12>, <그림 13>, <그림 14>, <그림 15>, <그림 16>, <그림 17>, <그림 18>에 나타내었다. 로그 레코드의 데이터 분석 결과는 <표 4>, <표 5>, <표 7>, <표 8>, <표 9>, <표 10>에 제시하였다. 각각의 로그 레코드에 기록된 로그 Opcode의 순서는 볼륨에서 일어난 변화들이 차례로 기록된 것이며, 또한 데이터 분석을 통해 포렌식에 사용할 수 있는 생성된 파일에 대한 정보도 $LogFile에서 얻을 수 있다.

<표 11>은 주요 로그 레코드에서 얻은 정보를 종합한 것이다. $LogFile에 여러 로그 레코드를 통해 생성된 파일은 myfile. txt라는 것을 알 수 있다. 또한 볼륨 상에 위치, 파일 생성 시간 정보, 인덱스 엔트리 정보, 파일의 MFT 정보를 종합해보면, 파일 시스템 상에서 myfile. txt가 생성되면서 일어난 파일 시스템상의 변화를 재구성해 낼 수 있다. 또한 로그 레코드 데이터에 들어 있는 시간 정보들을 활용하여 타임 라인에 따라 재구성하게 되면 파일시스템의 변화도 파악해 낼 수도 있다.

<표 11>에서 제시하는 바와 같이 로그 Opcode와 로그 레코드 데이터의 분석을 통해 얻어진 정보는 포렌식에 활용될 수 있는 매우 유용한 정보들이다.

본 연구를 통해 볼륨에서 생성된 파일 myfile. txt 파일에 대한 정보가 기록된 $LogFile의 로그 레코드에서

<표 11> 주요 로그 레코드 분석을 통해 얻은 정보

로그 레코드 (Redo/Undo)	로그 레코드 연산에 따른 분석 (로그 레코드의 Redo/Undo 데이터)
0x15/0x16	파일의 MFT 엔트리 비트맵 할당 정보
0x00/0x03	파일의 MFT 엔트리 할당 정보
0x0E/0x0F	파일의 인덱스 엔트리 정보
0x07/0x07	$MFT의 업데이트 정보
0x14/0x14	인덱스 엔트리의 $FILE_NAME 정보
0x02/0x00	생성된 파일의 볼륨 상의 위치 정보 생성된 파일의 MFT 헤더, 0x10 속성, 0x30 속성

얻은 정보들은 포렌식을 위한 정보로 활용할 수 있으며, 볼륨의 MFT 엔트리 정보가 아닌, $LogFile에 기록된 정보만으로도 포렌식을 위한 정보를 추출할 수 있음을 보였다. 로그 파일에 기록된 로그 레코드 연산은 파일 생성의 과정의 각 단계는 매우 깊은 연관이 있음을 알 수 있다. 파일 생성 시에 $LogFile에 기록되는 로그 레코드는 어떤 경우나 동일하다. 그러므로 누가 분석을 하여도 객관적인 결과를 얻을 수 있으므로 객관성을 가질 수 있다.

Ⅳ. 결론

본 연구에서는 NTFS 파일 시스템의 $LogFile의 구조를 역공학적인 방법으로 구조를 분석하였고, 분석된 로그 레코드의 주요 정보를 통해서 파일 생성에 연관된 로그 레코드들 역할을 분석하였다. 로그 레코드들 중에서 0x02, 0x03, 0x07, 0x14, 0x15, 0x16, 0x17, 0x0E, 0x0F 등의 Opcode를 갖는 로그 레코드가 분석의 대상이며 이것들은 일련의 연관된 동작 관계로 구성됨을 밝혔다.

본 연구의 결과는 $LogFile의 로그 레코드의 역공학적인 분석 방법에 관한 절차와 방법 제시뿐만 아니라 포렌식 절차와 방법을 통해 볼륨 상에서 파일이 생성될 때에, NTFS가 $LogFile에 로그 레코드를 기록하는 순서와 방법을 역공학적으로 분석하였다.

본 연구에서는 기존의 연구[4-6] 보다 더 자세한 로그 레코드 분석을 수행하였다. NTFS의 위치 정보를 분석하는 필요한 VCN과 MFT 엔트리의 오프셋에 대한 데이터 분석 방법을 추가함으로써, 볼륨 상에서 발생한 하나의 트랜잭션의 주요 부분을 분석할 수 있는 방법과 절차를 제안하였다. 이것은 파일 생성에 관련된 로그 레코드의 분석에 적용되었다.

본 연구에서 $LogFile에 대한 로그 레코드의 구조를 알기 위해 많은 시간을 역공학적 방법에 투자해야 했음을 안타깝게 생각하고 있다. 구조 분석의 시간을 절약할 수 있었다면 좀 더 빨리 연구 결과를 발표할 수 있었을 것이다. 이 연구는 $LogFile을 이용한 디지털 포렌식을 위한 본격적인 연구의 출발점이 되기를 바라고 있다.

참고문헌

[1] FAT, HPFS 및 NTFS 파일 시스템의 개요, http://support.microsoft.com/kb/100108/ko/

[2] 김태석(역), “임베디드 리눅스 시스템 구축하기,” 한빛미디어, 2004, p. 313.

[3] Pramada Singireddy, “Recoverability Support in NT File System(NTFS),” http://eas.asu.edu/~cse532/

[4] 조규상, “컴퓨터 포렌식을 위한 NTFS 저널 파일의 분석,” 디지털 포렌식 연구, Vol. 3, No. 1, 2009, pp. 51-60.

[5] 조규상・김태한, “컴퓨터 포렌식에 사용하기 위한 NTFS $LogFile의 로그 레코드 데이터 구조 분석,” ICS'2000 정보 및 제어심포지엄 논문집, 2010, pp. 230-231.

[6] 김태한・조규상, “NTFS $LoFile에서 상주 속성 파일의 컴퓨터 포렌식,” ICS'2000정보및제어심포지엄 논문집, 2010, pp. 69-70.

[7] 정준석・정원용, “임베디드 개발자를 위한 파일시스템의 원리와 실습,” 한빛미디어, 2006, pp. 274-275.

[8] Mark E. Russinovich, David A. Solomon,

"WINDOWS INTERNALS," 2006, pp. 995-1000.
[9] NTFS Recovery Support, http://codeidol.com/other/inside-windows-2000/File-Systems/NTFS-Recovery-Support/
[10] Priscilla Oppenheimer, "New Technologies File System(NTFS)," 2008.
[11] B. Carrier, File System Forensic Analysis, Addison-Wesley, 2005, pp. 340-341.
[12] K. Dreher, "NTFS", Master Thesis of Department of Information Technology Institute of Technology, Lund, Nov., Sweden, 1998.

■ 저자소개 ■

김 태 한
Kim, Tae Han

2006년 3월~현재 동양대학교대학원 박사과정
2004년 8월 안동대학교대학원 컴퓨터교육(석사)

관심분야 : 파일시스템 분석, Digital forensic
E-mail : school4u@korea.com

조 규 상
Cho, Gyu Sang

1996년 3월~현재 동양대학교 컴퓨터정보전학과 교수
1997년 2월 한양대학교 전자공학과(공학박사)

관심분야 : 정보보호, Automated forensic
E-mail : cho@dyu.ac.kr

논문접수일 : 2010년 5월 6일
수 정 일 : 2010년 5월 25일
게재확정일 : 2010년 6월 1일