논문 2012-49CI-4-1

NTFS 파일시스템의 $LogFile의 로그레코드에 연관된 컴퓨터 포렌식 대상 파일을 찾기 위한 방법

(Method for Finding Related Object File for a Computer Forensics in a Log Record of $LogFile of NTFS File System)

조 규 상*

(Gyu-Sang Cho)

요 약

이 연구는 NTFS 파일시스템에서 디스크의 에러를 복구를 위해 사용하는 저널링 파일($LogFile)에 기록되어 있는 로그 레코드들을 컴퓨터포렌식에 활용하는 방법에 관한 것이다. $LogFile에는 파일 연산의 과정을 그대로 담고 있어서 포렌식의 관점에서는 증거의 보고이다. 그러나 이 구조는 완전하게 공개되어 있지 않아서 구조를 정확히 알고 있지 않다는 어려움이 있다. 이 연구에서는 로그레코드들 기록된 주소 영역에 대해서 역공학적 방법으로 주소 데이터의 구조를 명확히 밝히고 로그레코드마다 그에 관련된 대상 파일이 어떤 것인지 분석하여 포렌식 대상 파일을 식별하고 포렌식 정보를 취득에 활용한다.

Abstract

The NTFS journaling file($LogFile) is used to keep the file system clean in the event of a system crash or power failure. The operation on files leaves large amounts of information in the $LogFile. Despite the importance of a journal file as a forensic evidence repository, its structure is not well documented. The researchers used reverse engineering in order to gain a better understanding of the log record structures of address parts, and utilized the address for identifying object files to gain forensic information.

Keywords : Computer Forensics, $LogFile, Log Record, Windows NTFS

Ⅰ. 서 론

NTFS는 Windows 2000, XP, Vista, 7 등에 탑재되어 있다. 이 파일 시스템은 작업한 트랜잭션을 기록하고 있기 때문에 에러 복구가 가능하다[1]. 복구 기능을 저널링이라고 부른다. 저널링 기능이 있는 다른 파일 시스템들은 JFS, Ext3, ReiserFS 등이 있다[2]. NTFS에서의 저널링 기능은 $LogFile 파일에 로그레코드를 기록하여 수행하고 있다. NTFS의 많은 부분들[1~2, 12]은 구조가 알려져 있지만 이 저널 파일의 구조에 대해서는 구조가 잘 알려져 있지 않다.

몇 연구에서 $LogFile의 분석을 시도하였지만 완전한 구조를 내놓지는 못했다[3~4]. 최근에 역공학적 방법으로 이것에 대한 분석과 활용에 대한 연구가 이어지고 있다. 조[5]는 이 $LogFile의 로그 레코드에서 디지털 포렌식에 활용하기 위해 파일의 복사와 삭제 연산을 수행할 때 어떤 연산의 로그 레코드들은 여러 개의 로그 레코드의 집합으로 연산이 구성된다는 것을 밝혔다. 그 후의 연구[6]에서는 역공학적 분석으로 Opcode 0x15/0x16인 로그레코드의 데이터 구조를 분석하고 상주 속

* 정회원, 동양대학교 컴퓨터정보전학과
(Dongyang Univ., Dept. of Computer Information Warfare)
접수일자: 2012년6월19일, 수정완료일: 2012년7월4일

성 파일이 삭제될 경우에 대한 로그레코드의 분석을 수행하여 데이터의 의미를 이해함으로써 포렌식에 활용할 수 있는 정보를 얻었다[7]. 후속연구[8]에서는 $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 제안하였다.

다른 연구[9]에서는 로그레코드들에 대한 역공학적 분석을 통해서 폴더를 생성할 때 발생하는 로그레코드들에서 포렌식 정보와 그 과정을 보이면서 로그레코드를 활용한 포렌식의 유용성을 입증하였다. 그리고 툴을 사용하여 파일의 타임스탬프를 변조하는 것을 탐지하기 위한 포렌식 방법을 제안하였다. 파일의 연산들이(생성, 복사, 이동 등)과 파일시간변조 툴을 사용할 때의 타임스탬프의 패턴이 다르다는 점에 착안한 포렌식 방법을 소개하였다[10].

NTFS의 파일 시스템에서 디스크의 복구를 위한 저널링 파일($LogFile)은 파일 연산에 대한 많은 정보를 담고 있어서 포렌식의 관점에서는 매우 중요한 정보의 보고이다. 그러나 이 파일의 구조가 완전하게 공개되어 있지 않아서 활용의 어려움이 있었다. 그 동안 여러 연구를 통해서 역공학적인 방법으로 이 구조를 파악해왔다[5~11]. 그 결과 많은 부분의 구조를 알게 되었고 그것을 포렌식 정보를 얻는데 활용할 수 있게 되었다. 이 연구에서는 로그레코드의 정보가 좀 더 체계적이고 논리적으로 완성된 방법으로 활용되기 위해서 로그레코드에 기록된 대상 파일의 주소에 대한 명확한 분석으로 대상 파일의 주소를 산술적으로 계산하는 방법을 제시하고 그 대상 파일에서 얻을 수 있는 포렌식 활용방법에 대해서 다루기로 한다.

Ⅱ. 로그레코드의 구조

1. LFS(Logging File Service)

NTFS에서는 파일과 디렉토리 정보를 MFT(Master File Table)에 저장한다[1]. 파일 또는 디렉토리의 일련번호를 나타내는 MFT 엔트리 중에서 0~15까지는 시스템 파일(Metadata file)로 사용하고 일반 파일/디렉토리는 그 이후 번호부터 사용한다. 그 중에서 저널링 또는 로깅이라고 부르는 디스크 상에서 발생하는 트랙잭션 정보는 $LogFile(MFT Entry 2번)에 기록된다. 이 작업은 윈도우즈의 LFS(Log File Service)[2]를 통해서 수행된다. 로그 파일은 4KB 단위로 페이지를 구성하고 그 안에 여러 레코드들을 저장한다. 첫 번째 두 개의 페이지는 재시작(Restart) 영역과 그것의 사본이 저장된다. 그 뒤에 보통의 로그 레코드들이 저장되는 4KB단위의 여러 개의 페이지로 구성된 로깅 영역으로 구성되어 있다. 전체의 영역(즉, $LogFile의 크기)은 크기가 고정되어 있기 때문에 그 끝에 도달하면 로깅 영역의 첫 위치로 다시 순환하면서 덮어쓰기를 한다[2]. 로그레코드들은 사용되는 목적에 따라 업데이트 레코드, 체크포인트 레코드로 구분된다. 업데이트 레코드는 가장 일반적인 형태의 로그레코드이다. 이 업데이트 로그레코드에는 Redo와 Undo의 두 종류의 정보[2, 4]가 들어 있다. Redo정보는 커밋이 완료되었지만 아직 캐시에서 디스크에 써지기 전에 시스템 오류가 발생한 경우에 트랜잭션을 다시 적용하기 위해 사용되는 정보이다. Undo 정보는 커밋이 완료가 되지 않은 상태에서 시스템 오류가 발생하였을 때 이미 수행한 트랜잭션 부분들을 역으로 되돌리는(취소하는) 정보가 들어있다. $LogFile의 전체 구성도를 그림 1에 나타냈었다. 각 블록은 4KB단위의 페이지 단위를 의미한다. 그림에서는 편의상 페이지당 한 개의 로그레코드만 표시하였지만 실제로는 각 페이지에는 여러 개의 로그레코드들이 들어있다.

NTFS는 디스크 복구를 위해 5초마다 주기적으로 체크포인트 레코드를 저장한다. 이것을 이용하여 디스크에 오류가 발생하는 즉시 어떤 지점으로 되돌아가 오류 복구를 시작해야하는지 알리기 위해 사용된다. 체크포인트 레코드를 페이지에 기록하고 나서 시작 영역에 체크포인트의 LSN 번호를 기록을 하여 가장 최근에 기록된 체크포인트가 어디인지 알 수 있도록 하며 이 정보를 이용하여 오류가 발생했을 때 빠르게 복구의 시작지

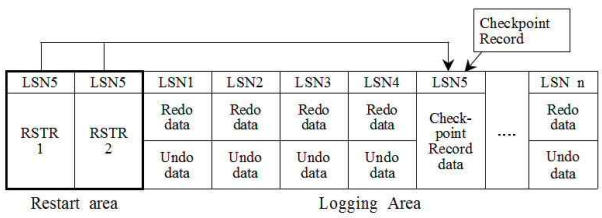

그림 1. $LogFile의 전체 구성도
Fig. 1. Configuration of $LogFile.

을 찾기 위함이다[2, 4].

로그레코드는 디스크 작업 실패를 복구하기 위한 목적으로 트랜잭션 정보들을 저장하고 있다. 컴퓨터 포렌식의 관점에서는 체크포인트 레코드의 중요성 보다는 업데이트 로그 레코드가 관심의 대상이 된다. 그 안에 파일 연산에 대한 구체적인 작업내역(데이터)들을 기록하고 있기 때문이다. 그래서 이 연구에서는 업데이트 로그 레코드의 구조만을 다루고 있고, 간단히 로그레코드라고 부를 때는 업데이트 로그레코드를 의미한다.

2. 로그 레코드의 구조

그림 2는 로그 레코드 데이터 구조를 나타낸 것이다. 0x58(88바이트)바이트까지는 고정된 속성들로 구성된다. Undo Op./Redo Op.의 코드 값에 따른 데이터는 0x58바이트 이후에 첨부된다. 그 내용은 코드의 내용에 따라 각기 다른 형태의 가변적인 데이터로 구성된다. 각각의 속성들에 대한 설명은 다음과 같다.

This LSN은 이 레코드의 LSN(Logical Sequence Number)의 번호를 나타내는 것이고 Previous LSN은 이 레코드 바로 앞의 레코드, Undo Next LSN은 작업을 취소할 때 되돌아갈 LSN의 번호를 나타낸다. 보통의 경우 이 두 개는 동일한 값을 갖지만 반드시 같지는 않다.

Data Length는 이 레코드에 할당된 전체 데이터 크기를 의미한다. Sequence Number는 $LogFile이 재시작할 때 이 값이 0으로 되고 종료될 때 1로 증가된다. 결국, 종료되기 전에 체크하게 되므로 이 값은 늘 0인 상태로 체크된다. Client Index 값도 항상 0이다.

Record Type에서 값 1은 업데이트 레코드를 의미하고 값 2는 체크포인트 레코드를 의미한다. Transaction ID는 레코드의 작업 속성을 의미한다. Flags는 로그 레코드 페이지가 넘쳐서 다음 페이지에 이어서 로그 레코드를 저장해야 하는 경우 1이 기록된다. 주로 페이지의 맨 마지막 로그 레코드의 경우에 이 값에 1이 된다.

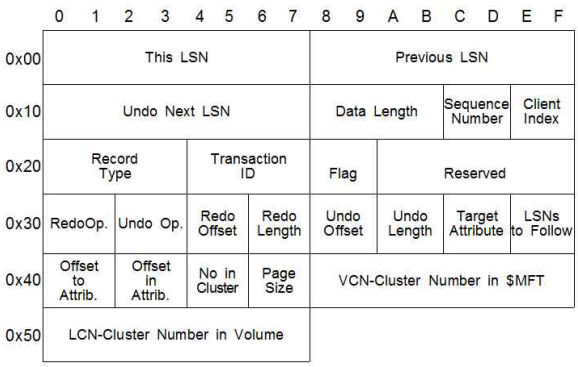

그림 2. 로그레코드의 구조
Fig. 2. Structure of Log Record.

Redo Op.와 Undo Op.는 각각 Redo(커밋이 완료된 트랜잭션)의 경우와 Undo(커밋이 완료되지 않은 트랜잭션)의 종류를 나타내는 Op. 코드의 값을 나타낸다. (Op. 코드표는 참고문헌 [5]와 [7]을 참조 바람). Redo Offset과 Undo Offset은 각각 Redo와 Undo를 수행할 때의 트랜잭션의 내용이 담겨있는 곳까지의 오프셋 값을 나타낸다. Redo Length와 Undo Length는 각각 Redo 또는 Undo의 오프셋 위치부터 시작된 트랜잭션의 내용이 길이를 나타낸다. Target Attribute는 작업 대상의 속성 값을 나타낸다. LSNs to Follow는 이 로그 레코드와 연관된 다음의 LSN이 있는지를 알리는데 사용한다. 이 값이 1일 때는 다음 로그 레코드가 이 로그레코드와 연관성을 갖고 있다는 의미이다. 이 때 다음 에 오는 로그레코드의 Previous LSN 값이 이 로그레코드의 LSN 값이 된다.

Ⅲ. 로그레코드와 연관된 파일 찾기

1. 로그레코드의 주소속성

그림 2의 로그레코드에서 0x40바이트 위치부터 0x57바이트(28바이트 크기)의 속성들은 로그레코드가 담고 있는 Redo와 Undo의 데이터가 적용되는 파일의 위치(주소)를 나타내는 정보이다. 포렌식의 관점에서 이들의 관계를 잘 파악하는 것이 포렌식 대상을 정확하게 분석하기 위한 중요한 분석의 기초 지점이다.

문헌상에 등장하는 로그레코드의 주소에 대한 정보는 Russon등[12]부터 참고할 가치가 있다. 이 문서는 리눅스 NTFS 파일 시스템을 개발하기 위한 프로젝트의 문서이다. 여기서 제공한 로그레코드의 주소 정보가 실제와 다른 부분이 있다. 그 후에 조[8]등의 연구와 김[11]의 연구에서 진전이 있었지만 기존의 연구에서는 로그레코드 마다 주소의 기록 방식이 다르다는 점에 대해서 적절한 명확히 설명을 하지 않았다. 주소의 표현방법은 작업 대상에 따라 방법이 다르다. 그것의 기준은 $MFT에 기록되는 MFT Entry의 내용이 변경이 될 경우에 주소를 기록하는 방법과 $Bitmap이나, $UsnJrnl

등과 같이 구조가 다른 파일인 경우에 주소를 기록하는 방법이 서로 차이가 있다.

가. 대상에 따른 주소 변환

로그레코드의 용도에 따라서 적용되는 파일이 다르게 된다. 로그레코드에 들어 있는 주소들이 서로 다른 방식으로 해석되는 이유이다. 다음의 네 가지 주소의 방식은 로그레코드의 사용목적에 따라 분류된 방식이다.

가. MFT엔트리 형식의 주소

그림 2에서 0x40부터 2바이트형식의 네 개의 값들은 순서별로 각각 “Offset to Attribute”, “Offset in Attribute”, “No in Cluster”, “Page Size”를 나타낸다.

“Offset to Attribute”은 $MFT파일에 들어있는 각 MFT엔트리에 들어있는 속성들의 놓인 곳의 오프셋 값을 나타낸다. “Offset in Attribute”는 “Offset to Attribute”로 지정된 속성의 시작지점부터의 지정한 위치까지의 오프셋 값이다. 4KB클러스터 안에는 1KB크기의 MFT엔트리가 4개가 들어간다. “No in Cluster”는 클러스터 내의 MFT엔트리의 위치를 나타내는 숫자이다. 숫자의 단위는 섹터값을 나타내고 0, 2, 4, 6중의 한 값을 갖는다. “Page Size“는 MFT엔트리에 해당하는 경우는 2로 정해진다. 한 개의 MFT엔트리의 크기는 두 섹터(1,024바이트 크기)로 구성된다는 의미이다.

“VCN”은 $MFT파일 내의 클러스터에 번호를 부여한 것으로 0번부터 시작한다. 그러므로 이것은 파일 내부의 클러스터 번호를 나타낸다. “LCN”은 전체 디스크 볼륨에서의 절대 클러스터 번호를 나타낸다. 대상 파일이 놓여있는 클러스터 번호이다. 여러 개의 클러스터로 구성된 파일인 경우에 이 값은 대상 클러스터의 값을 나타낸다. 반드시 파일의 첫 번째 클러스터 위치를 나타내는 것은 아니다. 첫 클러스터 번호로부터 시작되는 경우라면 “LCN”과 “VCN”값은 같은 값이 될 수도 있겠지만 일반적으로는 같지 않다.

이 방식은 다음과 같이 주소를 공식화 할 수 있다.

$MFT 대상의 주소=LCN
+No in Cluster
+Offset to Attribute
+Offset in Attribute (1)

나. Bitmap 할당 형식의 주소

비트맵은 파일이 클러스터에 차지하고 있는 상태를 나타내는 정보이다. NTFS에서는 MFT엔트리의 할당상태를 나타내는 비트맵 정보는 $MFT:$Bitmap에 저장되고 데이터만 저장되어 있는 클러스터의 할당상태를 나타내는 비트맵은 $Bitmap 파일에 저장된다. 이 두 가지는 다음과 같은 방식으로 주소가 결정된다.

비트맵 할당 방식의 주소를 알아보기 위해서는 파일의 주소를 나타내는 정보와 함께 파일내의 할당된 비트정보를 나타내는 8바이트(0x58~0x5F)의 데이터 정보가 더 필요하다. 그림 3에서 0x40에서 시작하는 첫 번째 8바이트는 항상 0이다. 0x48위치의 “VCN” 값이 각 비트맵 파일의 첫 클러스터부터 상대주소 값으로 나타낸 것이다. 0x50 위치의 “LCN”값은 전체 디스크 볼륨에서의 절대 클러스터 번호를 나타낸다. 앞에서 설명한 MFT엔트리 방식과 마찬가지로 이 클러스터 값은 작업대상 클러스터의 번호를 의미하므로 여러 클러스터로 구성된 큰 파일인 경우 이 값은 대개 첫번째 클러스터 값이 아닌 경우가 많다. 비트의 위치와 길이는 “Offset from head”와 “Bit Length”값을 계산하여 작업 대상의 위치가 결정된다.

$Bitmap의 주소=LCN
+Offset from head
+Bit Length (2)

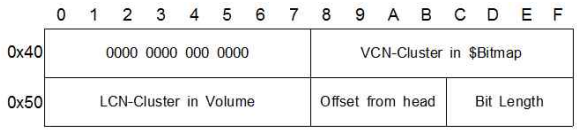

그림 3. $Bitmap에 관련된 주소결정 방식
Fig. 3. Address Calculation of $Bitmap.

다. INDEX형식의 주소

대상이 Index(디렉토리, 폴더)인 경우는 기록할 대상이 있는 클러스터의 번호를 0x58위치의 “LCN”에서 우

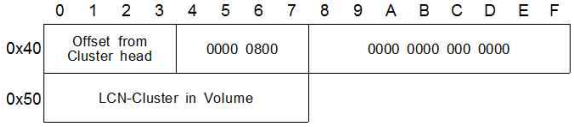

그림 4. INDEX에 관련된 주소결정 방식
Fig. 4. Address Calculation of INDEX.

선적으로 찾고 첫 번째 4바이트(0x40~0x43)에 들어 있는 오프셋 값을 계산한다. 두 번째 4바이트에는 항상 0x00000800이 들어 있다. 0x48~0x4F에는 항상 0이 들어 있다. (그림 4)

INDEX의 주소=LCN
+Offset from head (3)

라. $UsnJrnl:$J형식의 주소

$UsnJrnl파일은 파일이나 디렉토리에 대한 변경된 사항을 기록한다. 파일명, 시간변화, 어떤 종류의 변경이 발생했는지 기록하지만 이 저널 파일에는 데이터가 변경된 내용은 저장하지는 않는다.

기록할 대상이 있는 클러스터의 번호를 0x58위치의 "LCN"에서 먼저 찾는다. 첫 번째 4바이트(0x40~0x43)에 들어 있는 오프셋 값을 계산한다. 두 번째 4바이트에는 항상 0x00000000이 들어 있다. 0x48~0x4F에는 "VCN"값이 들어 있다. 이 방식은 INDEX방식과 유사하게 보이지만 이 방식과 INDEX방식의 차이점은 두 번째 4바이트에 0x00000000이 들어있다는 점과 VCN의 내용이 기록된다는 점이다.(그림 5)

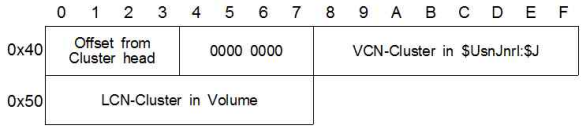

그림 5. $UsnJrnl:$J의 주소 결정
Fig. 5. Address Calculation of $UsnJrnl:$J.

$UsnJrnl:$J의 주소=LCN
+Offset from head (4)

2. CopySample.txt의 복사 연산의 사례

포렌식 대상 파일은 d:\test\CopySample.txt이다. 다른 드라이브에서 d:\test 폴더로 복사한다. 파일의 크기는 1,698바이트이다. 표 1은 이 때에 발생한 전체의 로그레코드들이고 각 로그레코드에 관련된 모든 파일들을 나타낸 것이다. 포렌식의 관점에서 이 표는 파일의 복사 연산 시에서 발생하는 로그레코드의 데이터가 어떤 파일들에 관련된 것인지 파악하고 활용하기 위함이다. 다음의 각 예들은 로그레코드의 관련 주소를 구하는 사례를 나타낸 것이다. 이 로그레코드들은 모두 표 1에 포함된 것들이다.

그림 6은 $MFT에 들어있는 MFT엔트리와 관련된 위치를 결정하는 방식이다. MFT 엔트리를 저장하는 각 클러스터 크기는 4KB이고 각 파일의 내부 기본 속성의 크기는 1KB이다. 각 클러스터마다 4개까지 파일 정보를 저장할 수 있다. 이 로그레코드의 Opcode는 0x00/0x03이다. 0x00은 No operation를 의미하고 0x03은 Deallocate file record segment의 동작을 수행하는 코드이다.

"c"의 LCN에 들어있는 0x0C5C9E(810142)는 $MFT 파일의 절대 클러스터 주소이다. 이 주소에는 MFT엔트리 #94840번 파일이 들어있다. "a"의 Offset to Attribute=0x0000이고 Offset in Attribute=0x0000이다. No in Cluster=0x006이므로 이것은 6번째 섹터를 의미한다. Page Size=0x0002이다. 그러므로 포렌식 대상 파일 CopySample.txt은 0x0C5C9E(810142)에 6*0.5K=3을 더한 위치는 절대위치이고 파일 내의 상대위치를 나타내는 VCN=0x00005C9E부터 계산하면 이 파일의 MFT 번호는 #94843이 된다.

그림 7은 0x0c/0d 로그레코드(Add/Delete index entry root)의 경우이다. 그림의 "c" 부분에서 LCN절대 클러스터 번호 0x0c5aa8(809640)은 파일의 MFT번호가 #92832인 \test폴더의 MFT정보가 들어있는 클러스터이다. 0x5aa8은 VCN, 즉 $MFT파일 선두부터 0x5aa8(23208)클러스터 위치에 해당 파일이 들어있다는 의미이다. 0x0128+0x40위치는 \test 폴더정보의 $Data 속성(0x90번 속성)의 위치를 의미한다. 그 자리부터 0x70크기의 데이터(그림 7의 "d")를 기록한다. 그런데 실제

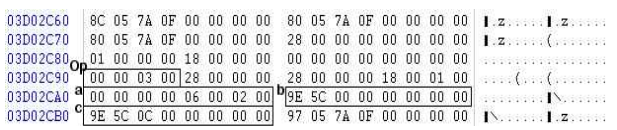

그림 6. 0x00/0x03 로그레코드의 주소
Fig. 6 Address of 0x00/0x03 Log Record.

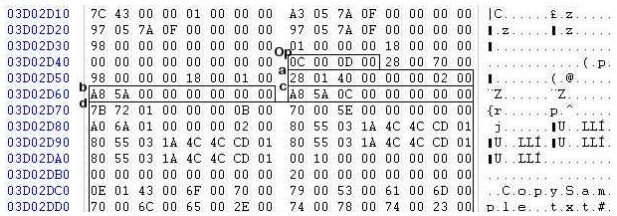

그림 7. 0x0c/0x0d 로그레코드의 주소
Fig. 7 Address of 0x0c/0x0d Log Record.

표 1. “CopySample.txt“의 복사연산의 로그레코드와 그에 관련된 파일의 위치(16진수, little endian)
Table 1. Log records and related file location of “CopySample.txt“ copy operation.(hexadecimal, little endian).

LSN	Redo/Undo	주소 필드			관련파일	비고
		0x40~0x47	0x48~0x4F(VCN)	0x50~0x57(LCN)		
0F7A0580	15/16	0000 0000 0000 0000	02000000 00000000	74E93800 00000000	$MFT:$Bitmap	Bitmap형식
0F7A058C	00/03	0000 0000 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A0597	15/16	0000 0000 0000 0000	CA000000 00000000	D685B102 00000000	$Bitmap	Bitmap형식
0F7A05A3	0C/0D	2801 4000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A05BC	06/05	2801 0000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0628	15/16	0000 0000 0000 0000	CA000000 00000000	D685B102 00000000	$Bitmap	Bitmap형식
0F7A0634	05/06	2801 0000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0649	05/06	7801 0000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0659	07/07	7801 2000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0666	0B/0B	2801 0000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0677	1C/00	0000 0000 0000 0000	00000000 00000000	08DD6EB0 C0FE4F84		주소형식아님
0F7A0689	08/00	0000 0000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A06ED	05/06	2801 0000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A0703	0E/0F	0000 B000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A071C	02/00	0000 0000 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A0762	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A0775	08/00	B001 0000 0000 0000	DA020000 00000000	71507501 00000000	$UsnJrnl:$J	$J데이터형식
0F7A078B	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A079E	1B/01	0000 0000 0000 0200	00000000 00000000	50396AC1 16C34988		LSN의 연결 끝
0F7A07A9	07/07	3800 2000 0000 0200	A85A0000 00000000	A85A0C00 00000000	\test	MFT형식
0F7A07C4	14/14	0000 9805 0000 0800	01000000 00000000	035A9B00 00000000	\root	INDEX
0F7A07DD	1B/01	0000 0000 0000 0200	00000000 00000000	8C5A30C0 005A30C0		LSN의 연결 끝
0F7A07E8	0B/0B	8801 0000 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A07F9	07/07	3800 2000 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A081C	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A082F	08/00	0802 0000 0000 0000	DA020000 00000000	71507501 00000000	$UsnJrnl:$J	$J데이터형식
0F7A0845	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A0858	1B/01	0000 0000 0000 0200	00000000 00000000	F80D0000 C830D3B0		LSN의 연결 끝
0F7A0863	14/14	0000 4000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A087C	14/14	0000 B000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A0895	1B/01	0000 0000 0000 0200	00000000 00000000	1C000000 03000000		LSN의 연결 끝
0F7A08A0	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A08B3	08/00	6002 0000 0000 0000	DA020000 00000000	71507501 00000000	$UsnJrnl:$J	$J데이터형식
0F7A08C9	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A08DC	1B/01	0000 0000 0000 0200	00000000 00000000	5038B1B1 16C34988		LSN의 연결 끝
0F7A08E7	07/07	3800 2000 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A0902	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A0915	08/00	B802 0000 0000 0000	DA020000 00000000	71507501 00000000	$UsnJrnl:$J	$J데이터형식
0F7A092B	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A093E	1B/01	0000 0000 0000 0200	00000000 00000000	480D0000 C830D3B0		LSN의 연결 끝
0F7A0949	14/14	0000 4000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A0962	14/14	0000 B000 0000 0800	00000000 00000000	A5436500 00000000	\test의 INDEX	INDEX
0F7A097B	1B/01	0000 0000 0000 0200	00000000 00000000	00000000 08DD6EB0		LSN의 연결 끝
0F7A0986	07/07	3800 5800 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A0993	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A09A6	08/00	1003 0000 0000 0000	DA020000 00000000	71507501 00000000	$UsnJrnl:$J	$J데이터형식
0F7A09BC	0B/0B	0801 0000 0600 0200	9D5A0000 00000000	9D5A0C00 00000000	$UsnJrnl	MFT형식
0F7A09CF	07/07	3800 5800 0600 0200	9E5C0000 00000000	9E5C0C00 00000000	\test\CopySample.txt	MFT형식
0F7A09DC	1B/01	0000 0000 0000 0200	00000000 00000000	02A5F584 B4396AC1		LSN의 연결 끝

이 값은 외부속성을 나타내므로 데이터는 index의 외부 속성 파일에 저장된다. 이 로그레코드는 루트 인덱스를 추가하고 삭제하는 기능을 수행하지만 인덱스의 주소방식이 아닌 MFT의 주소 방식을 사용한다. 이것은 $MFT 내에 들어있는 인덱스 파일인 \test에 새로 복사에 대한 내용을 기록하는 과정이다.

그림 8은 0x0E/0x0F 로그레코드이다. 각 코드는 Add/Delete index entry allocation을 의미한다. “c” 부분의 주소 0x006543A5는 클러스터의 절대주소인 “LCN” 주소를 의미한다. “a”의 0xB0는 INDEX에서의 오프셋을 의미한다. “a”부분의 끝의 값은 항상 0x0008이다. 기록할 내용은 “d”부분이고 이것의 크기는 0x70 바이트이다. 인덱스 파일에 DOS 포맷의 파일 정보를 기록한다. 클러스터의 물리적인 주소는 0x06543A5000에 오프셋 값 0xB0을 더한 0x06543A50B0가 대상 위치에 “d”부분의 데이터를 기록하게 된다.

그림 9는 $Bitmap 파일에 적용되는 주소방식이다. 0x15/0x16코드는 각각 Set/Clear bits in nonresident bitmap 동작을 하는 코드이다. “c”부분의 0x02b185d6 (45188566)은 $Bitmap의 여러 개의 클러스터들 중에서

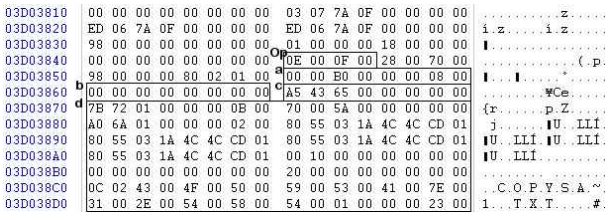

그림 8. 0x0E/0x0F 로그레코드의 주소
Fig. 8 Address of 0x0E/0x0F Log Record.

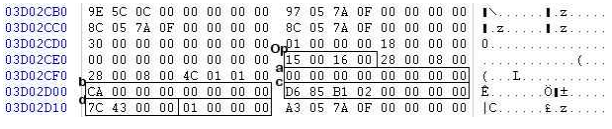

그림 9 0x15/0x16 로그레코드의 주소
Fig. 9 Address of 0x15/0x16 Log Record.

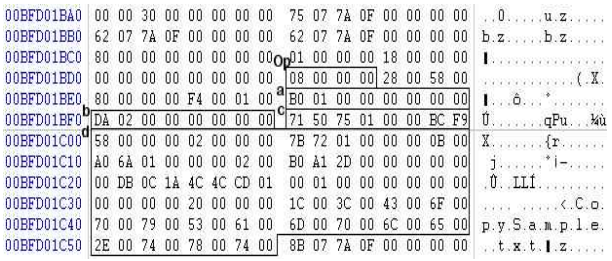

그림 10. 0x08/0x00 로그레코드의 주소
Fig. 10 Address of 0x08/0x00 Log Record.

하나이다. 이 클러스터의 값은 $Bitmap중에서 선두 클러스터를 의미하지 않는다. “b”에 기록된 값 0xCA(202) 번째 떨어진 클러스터 위치에 작업 대상이 되는 클러스터가 들어 있다는 의미이다. 비트맵의 비트의 설정은 “d”부분에 있는 데이터 영역에서 비트의 위치를 결정한다. 비트의 오프셋은 0x437C번째 비트를 의미하고 기록될 비트의 수는 1개이다.

그림 10은 0x08/0x00 로그레코드이다. 각 코드는 Update nonresident value과 No operation을 의미한다. “c” 부분의 주소 0x01755071은 클러스터의 절대주소를 나타낸다. 이 부분은 $UsnJrnl:$J가 들어 있는 곳이다. “b”의 0x02DA(730)은 VCN을 나타낸다. 현재 작업 대상 클러스터는 이 파일의 크기가 커서 선두부터 730번째 클러스터 위치라는 의미이다. “a”의 0x01B0는 클러스터의 선두부터의 오프셋을 의미한다. 물리주소로 표현하면 0x01755071000의 클러스터 위치에 0x01B0를 더한 위치인 0x017559711B0위치에 “d”부분을 기록한다. “c”의 끝 두 바이트는 fixup record를 나타낸다. 실제 값은 0x0000이다.

Ⅳ. 결 론

전원이 꺼지는 등의 갑작스런 파일시스템의 오류가 발생할 때 복구를 하기 위한 방법으로 사용하는 저널링 파일인 $LogFile은 컴퓨터 포렌식의 관점에서는 정보의 보물창고인 것이다. 그러나 그 구조는 완전하게 공개되어 있지 않아서 이것을 활용하는데 어려움이 따른다. 구조에 대해서 명확한 이해를 위하여 이 연구가 수행되었다. 그 중에서도 각 로그레코드가 갖고 있는 주소 정보에 대한 해석을 할 수 있도록 로그레코드에 들어있는 주소의 정보에 관련 있는 파일들을 찾는 방법을 네 가지 방식의 주소 표현방법으로 로그레코드와 관련된 주소를 구하고 파일복사 연산의 사례에 그 방법을 수행한 결과를 보였다. 본 연구의 방법은 $LogFile을 활용한 포렌식 툴을 개발하는데 필수적인 요소로 사용될 것이다.

참 고 문 헌

[1] B. Carrier, *File System Forensic Analysis*, Addison-Wesley, pp. 340-341, 2005.

[2] Mark E. Russinovich and David A. Solomon, *Windows Internls*, Microsoft Press, Chapter 10, pp. 995-1000, 2006
[3] Pramada Singireddy, "Recoverability Support in NT File System(NTFS)", http://eas.asu.edu/ ~cse532/
[4] K. Dreher, "NTFS", *Master Thesis of Department of Information Technology Institute of Technology*, Lund, Nov., Sweden, 1998.
[5] 조규상, "컴퓨터 포렌식을 위한 NTFS 저널 파일의 분석", *디지털 포렌식 연구*, Vol. 3, No. 1, 2009, pp. 51-60.
[6] 조규상・김태한, "컴퓨터 포렌식에 사용하기 위한 NTFS $LogFile의 로그 레코드 데이터 구조 분석", *ICS'2000 정보 및 제어심포지엄 논문집*, 2010, pp. 230-231.
[7] 김태한・조규상, "NTFS $LoFile에서 상주 속성 파일의 컴퓨터 포렌식", *ICS'2000정보및제어심포지엄논문집*, 2010, pp. 69-70.
[8] 김태한・조규상, "NTFS 파일 시스템의 저널 파일을 이용한 파일 생성에 대한 디지털 포렌식 방법", *디지털산업정보학회 논문지*, 6권2호, pp,107-118, 2010.
[9] Gyu-Sang Cho and M. Rogers, "Finding Forensic Information on Creating a Folder", *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST (Proceedings of ICDF2C 2011)*, Springer, 2012
[10] Gyu-Sang Cho and M. Rogers, "A Computer Forensic Method for Detecting Timestamp Forgery in NTFS", *Computer & Security*. 2012. (예정)
[11] 김태한, "NTFS 파일 시스템에서 저널 파일의 역공학 분석 통한 컴퓨터 포렌식", *동양대학교 박사학위논문*, pp.26-29, 2010.
[12] Richard Russon and Yuval Fledel, *NTFS Documentation, Chapter 3. NTFS files:$LogFile*, pp. 38-42, http://linux-ntfs.sourceforge.net

저 자 소 개

조 규 상(정회원)
1986년 한양대학교 전자공학과 학사졸업
1989년 한양대학교 전자공학과 석사졸업
1997년 한양대학교 전자공학과 박사졸업.
2010년～2011년 미국 Purdue대학교. Visiting scholar
1996년～현재 동양대학교 컴퓨터정보전학과 교수
<주관심분야 : 시스템보안, 컴퓨터포렌식>
<주관심분야 : 시스템보안, 컴퓨터포렌식>