KIPS Tr. Comp. and Comm. Sys.
Vol.5, No.12 pp.463~470 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2016.5.12.463

A Study to Improve Recovery Ratio of Deleted File Using the Parsing Algorithm of the HFS + Journal File

Seung Gyu Bang[†] · Sang Jun Jeon[††] · Do Hyun Kim[†††] · Sang Jin Lee[††††]

ABSTRACT

With the growing demand for MAC-based system, the need for digital forensic techniques of these system has been increasing. In the digital forensic analysis process, sometimes analysts have recovered the deleted files when they prove the allegations if system user try to remove the evidence deliberately. Research and analysis that recover the deleted files from a file system constantly been made and HFS+ that is a file system of MAC-based system also has been researched. Carving techniques primarily has been used to recover the deleted file from HFS+ a file system because metadata of folder or file overwrite metadata of a deleted file when file is deleted from a file system on HFS+ characteristic. But if the file content is saved by separated state in a file system, Carving techniques also can't recover the whole or a part of the deleted file. In this paper we describe technique the deleted file recovery technique using HFS+ file system a journal. This technique that is suggested by existing research and analysis result is the technique that recover the deleted file by metadata that is maintained in a journal on HFS+ file system. but this technique excludes specific files and this problem needs to be reformed. In this paper we suggest algorithm that analysis a journal of HFS+ file system in detail. And we demonstrate that the deleted file cat be recovered from the extracted metadata by this algorithm without the excluded file

Keywords : HFS+, HFS+ Journal, File Recovery, File System, Digital Forensic

HFS+ 저널 파일 파싱 알고리즘을 이용한 삭제된 파일 복구 기법 향상 방안

방 승 규[†] · 전 상 준[††] · 김 도 현[†††] · 이 상 진[††††]

요 약

최근 MAC 시스템의 점유율 증가로 MAC 기반 디지털 포렌식 기술의 필요성이 증대되고 있다. 디지털 포렌식 분석 과정에서 시스템 사용자가 의도적으로 증거를 삭제한 경우, 시스템에서 삭제된 파일을 복구하여 혐의를 입증하기도 한다. 이를 위해 파일시스템으로부터 삭제된 파일을 복구하기 위한 연구가 꾸준히 이루어져 왔으며, MAC 기반 파일시스템인 HFS+ 또한 이에 대한 연구가 수행되어왔다. HFS+의 운영 및 구조적 특성상 파일이 삭제되면 해당 파일의 메타데이터가 다른 파일 또는 폴더의 메타데이터에 의해 삭제되기 때문에 주로 시그니처를 활용한 카빙 기법이 사용되어왔다. 하지만 File Content가 파일시스템 상에 분할되어 저장되는 경우, 카빙 기법 또한 파일의 일부분만을 복구하거나 파일 전체를 복구할 수 없었다. 본 논문에서는 HFS+ 저널을 이용한 삭제된 파일의 복구 기법에 대해 소개한다. 이는 기존 연구를 통해 제안된 기법으로 HFS+ 저널에 남아있는 메타데이터를 이용하여 삭제된 파일을 복구하는 기법이다. 하지만 해당 기법은 특정 파일이 복구 대상에서 배제되기 때문에 이에 대한 개선의 여지가 남아있다. 본 연구에서는 HFS+ 저널을 상세히 분석할 수 있는 알고리즘을 제시한다. 또한 해당 알고리즘을 기반으로 추출한 메타데이터를 통해 복구 대상에서 배제되는 파일 없이 삭제된 파일을 복구할 수 있음을 실험을 통해 입증한다.

키워드 : HFS+, HFS+ 저널, 파일복구, 파일 시스템, 디지털 포렌식

※ 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단-공공복지안전사업의 지원을 받아 수행된 연구임(2012M3A2A1051106).
† 준 회 원 : 고려대학교 정보보호대학원 정보보호학과 석사과정
†† 비 회 원 : 고려대학교 정보보호대학원 정보보호학과 석·박사통합과정수료
††† 비 회 원 : 고려대학교 정보보호대학원 정보보호학과 박사과정수료
†††† 종신회원 : 고려대학교 정보보호대학원 교수
Manuscript Received : June 29, 2016
First Revision : September 5, 2016
Accepted : September 8, 2016
* Corresponding Author : Sang Jin Lee(sangjin@korea.ac.kr)

1. 서 론

특정 OS의 점유율 증가는 이후 해당 OS 기반 시스템이 디지털 포렌식 분석 대상으로 수집되는 상황 또한 증가될 수 있음을 의미한다. 현재 OS 시장 점유율 1위인 Windows는 최근까지도 해당 OS 기반 디지털 포렌식 기술이 연구되

고 있으며, Windows 기반 파일시스템인 NTFS에 대한 삭제된 파일의 복구 기술 또한 활발히 연구되어 왔다[1, 2]. Windows와의 격차는 크지만 OS 시장에서 MAC OS X의 점유율은 점차 증가하고 있으며, 이에 따라 MAC OS X 기반 디지털 포렌식 기술의 필요성이 증대되고 있다. 최근까지 MAC OS X의 아티팩트 연구[3] 및 특정 어플리케이션이 남기는 아티팩트[4] 등에 관한 연구는 진행되어왔지만, 삭제된 파일의 복구 연구는 비교적 더디게 진행되었다.

MAC OS X는 HFS+ 파일시스템을 사용한다. 해당 파일시스템은 계층형 구조로 데이터를 관리하며, 1998년에 도입되어 현재까지 Apple의 Macbook, iPhone, iMac 등에 사용되고 있다. HFS+는 파일이 삭제되면 파일의 메타데이터가 파일시스템 상에서 삭제(다른 파일 및 폴더의 메타데이터로 덮어써짐)되기 때문에 삭제된 파일의 복구를 위해 시그니처 기반 카빙 기법이 사용되어왔다. 하지만 파일이 단편화되어 저장된 경우엔 카빙 기법을 이용하더라도 완전한 복구가 불가능하며, 만약 파일이 압축되거나 암호화된 경우엔 파일 전체를 복구할 수 없다.

HFS+ 상에서 카빙 기법의 한계점은 분명히 존재했으며, 기존 연구[5]를 통해 이를 보완할 수 있는 기법이 제안되었다. 이는 HFS+의 저널에 남은 삭제 파일의 메타데이터를 통해 파일을 복구하는 기법이다. 해당 기법은 HFS+에 정의된 두 유형의 파일(Non-Extent Overflow 파일 이하 Non-Overflow 파일, Extent Overflow 발생 파일 이하 Overflow 파일) 중 Overflow 파일은 복구 대상에서 배제하였다. 결국, 기존 기법을 이용한다면 저널 상에 삭제된 Overflow 파일의 메타데이터가 잔존하더라도 복구되지 않는다.

본 논문에서는 HFS+ 저널을 이용하여 배제되는 파일 없이 삭제된 파일을 복구하는 알고리즘을 제시하며, 해당 알고리즘을 통해 실제 MAC 시스템 상에서 두 유형의 파일 모두 복구할 수 있음을 실험을 통해 입증한다.

2. 관련 연구

기존 HFS+ 저널을 이용한 삭제된 파일 복구 기법은 2008년에 소개되었다[5]. 해당 논문은 HFS+ 저널 내에서 Catalog File 레코드를 추출할 수 있는 절차와 스코어링 리스트(복구 가능성을 수치화할 수 있는 체크리스트)를 제시하였다. 또한 절차 및 리스트를 바탕으로 수행한 실험 결과를 제시하여 저널을 이용한 파일 복구가 가능하다는 사실을 입증하였다. 하지만 해당 논문은 Catalog File 레코드를 이용한 복구에 집중하였고, 결과적으로 Overflow 파일의 복구는 배제되었다.

논문이 발표된 후 Kzamiya가 2013년 7월 해당 논문을 바탕으로 HFS+ 기반 파일 복구 스크립트인 EnPack을 개발하여 공개했다[6]. 공개된 EnPack은 기존 논문에서 제시한 기법과 마찬가지로 저널로부터 Catalog File 레코드만을 추출하여 파일을 복구하였다. 또한 Kzamiya는 자신이 개발한 EnPack이 저널에 저장된 가장 최근 트랜잭션 정보만을 추출할 수 있다고 명시하였다. 즉, 저널 header의 start/end 속성이 가리키는 Offset에 포함된 block_list들만을 추출하여 분석하는 것이다. 결국 Kzamiya가 개발한 EnPack은 저널에 잔존하는 데이터 중 일부만을 분석하였으며, 기존 논문과 동일하게 두 파일 유형 중 Non-Overflow 파일만을 복구할 수 있다.

이와 유사한 연구로 2008년 Aaron Burghardt와 Adam J.Feldman이 아이폰에 적용되는 Data Protection을 분석하여 기기 내에 암호화된 파일들을 복호화하는 오픈소스를 공개하였다[9]. 이 오픈소스는 저널로부터 삭제된 파일의 레코드를 추출하여 파일을 복구하는 기능도 제공하였지만 해당 기능은 저널을 카빙하여 Catalog File 레코드만을 추출하는 방식을 사용하였으며, 기존 연구와 동일하게 Extent Overflow File 레코드는 배제되어 결과적으로 삭제된 Overflow 파일을 복구할 수 없었다.

HFS+ 파일시스템의 경우 저널 구조의 특성상 메타데이터만을 이용하여 저널 내에 모든 데이터를 추출하고 분류하는 것이 불가능하다. 이런 이유로 현재까지 카빙 방식을 이용한 삭제된 파일의 복구 기법이 주로 사용되어왔지만, 이 또한 결과적으로 Catalog File 레코드만을 추출하기 때문에 Overflow 파일은 복구대상에서 배제된다. 본 논문은 저널의 구조와 운영상 특징을 이용한 알고리즘을 제시하며, 이를 통해 기존에 저널로부터 추출할 수 없었던 Extent Overflow File 레코드를 추출하여 Non-Overflow 파일 뿐만 아니라 Overflow 파일 또한 복구할 수 있는 방안을 제시한다.

3. HFS+ 파일시스템

3.1 HFS+ 파일시스템의 전체 구조[7, 8]

HFS+ 파일시스템 구조는 Fig. 1과 같다. 파일시스템 전체 영역은 Reserved, Volume Header 및 5개 Special File (Allocation, Extents Overflow, Catalog, Attribute, Startup)들로 구성되며, 각 구성요소를 제외한 나머지 영역에 파일 데이터들이 저장된다. 각 구성요소들은 운영체제 및 파일시스템을 운영하기 위해 필요한 정보들을 저장한다.

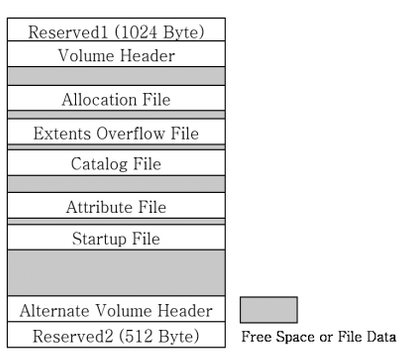

Fig. 1. Structure of the HFS+ File System

Volume Header는 파일시스템의 메타데이터(파일 수, 볼륨 크기, 생성 시간 등)와 Special File들의 메타데이터(볼륨 내 위치, 크기 등)를 저장한다. HFS+는 볼륨 전체를 블록

단위로 나눠 할당 및 해제하며, Allocation File에서 bitmap을 이용하여 할당/미할당 여부를 나타낸다. Catalog File은 볼륨 내에 저장된 파일이나 폴더의 메타데이터를 저장하는 파일이다. 또한 내부적으로 데이터를 B+Tree를[10] 이용하여 관리하며, 파일 전체가 Header Node, Index Node, Leaf Node로 구성된다. 파일 및 폴더의 메타데이터는 가장 하위 계층인 Leaf Node에 레코드 단위로 저장된다.

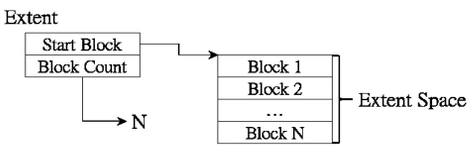

Fig. 2. Structure of an Extent Attribute

파일시스템에 파일이 생성되는 경우 File Content를 저장하기 위한 영역이 할당된다. 이때 할당된 영역의 메타데이터(위치, 크기)가 Catalog File 레코드에 포함(Fig. 2, Extent 속성)되며, 한 개의 레코드는 총 8개의 Extent 속성을 가지고 있다. 파일은 9개 이상의 영역으로 분할될 수 있고, 9개 이상으로 분할된 파일을 Overflow 파일이라고 한다. Overflow 파일의 경우 Catalog File 레코드에 할당받은 영역에 대한 정보를 모두 포함시킬 수 없기 때문에 8개 영역에 대한 정보는 Catalog File 레코드에 저장하고, 나머지 영역의 정보는 Extent Overflow File 레코드에 저장한다. 결과적으로 Non-Overflow 파일이 생성되면 Catalog File 레코드만 생성되지만, Overflow 파일이 생성되면 Catalog File 레코드 1개와 1개 이상의 Extent Overflow File 레코드가 생성된다. 또한 Extent Overflow File 또한 B+Tree를 이용하여 데이터를 관리한다.

3.2 HFS+ 파일시스템 저널링[7]

HFS+의 저널링 기능은 MAC OS X v10.3부터 기본 설정으로 포함되었다. 이는 시스템의 갑작스러운 정전이나 알 수 없는 오류에도 메타데이터의 원자성을 보장할 수 있도록 한다. 즉, Volume Header 및 5개 Special File들을 대상으로 저널링을 적용하며, File Content는 제외된다.

HFS+ 저널링은 일종의 Buffering 방식과 흡사하다. 만약 Volume Header 또는 5개 Special File 데이터에 대한 변경이 요구되면, 해당 데이터를 특정 단위만큼 저널에 복사한 뒤 저널 내에서 데이터를 변경한다. 이때 변경 도중 오류가 발생하면 파일시스템은 기존 상태를 유지하며, 오류 없이 성공적으로 완료된 경우 변경된 데이터를 파일시스템에 적용한다.

3.3 HFS+ 저널파일 구조[7, 8]

저널은 파일로 관리되기 때문에 Catalog File에 저장된 저널 파일 레코드를 참조하거나 Volume Header의 속성을 참조하여 추출할 수 있다. 또한 파일시스템 크기에 비례하여 10GB 당 1MB 크기(최대 100MB, 최소 8MB)로 파일이 생성된다.

저널은 Fig. 3과 같은 구조를 갖는다. header와 data buffer로 구분되며, data buffer에 block_list가 연속적으로 저장된다. 또한 data buffer는 원형 버퍼로 사용되기 때문에 block_list가 data buffer를 모두 채우면 기존에 저장된 block_list는 덮어써진다.

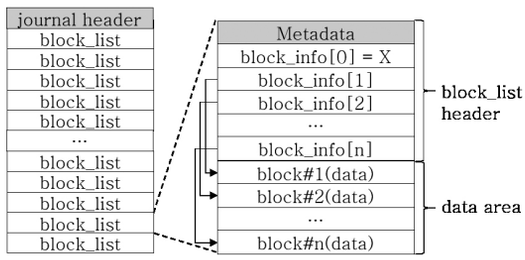

Fig. 3. Structure of the HFS+ Journal File

journal header는 Table 1과 같은 속성을 갖는다. size는 저널 전체 크기를 나타내며, jhdr_size와 blhdr_size는 journal header와 block_list header의 크기를 나타낸다. start/end 속성은 하나의 트랜잭션을 이루는 처음과 마지막 block_list Offset을 나타낸다. 만약 재실행할 트랜잭션이 존재하지 않는 경우(시스템의 정상 종료, 파일시스템의 정상적인 해제 등) start/end 속성은 동일한 Offset을 저장한다.

Table 1. Some of the Attributes in Journal Header

Attribute	Offset(Byte)	Size(byte)
start	0x08	0x08
end	0x10	0x08
size	0x18	0x08
blhdr_size	0x20	0x04
jhdr_size	0x28	0x04

block_list는 Fig. 3과 같이 header와 data 영역으로 구분되며, header 영역엔 block_info들이 저장된다. header에 저장된 n 번째 block_info는 data 영역에 저장된 n 번째 block에 매핑되며, block_info에 매핑된 block의 메타데이터가 저장된다. 각각의 block에는 Volume Header나 5개 Special File의 데이터가 Table 2에 명시된 단위로 저장된다. B+Tree를 이용하여 데이터를 관리하는 3개 파일은 내부 데이터가 변경될 때 Node Size 단위로 복사되며, Volume Header는 Volume Header Size 단위, Allocation File은 Block Size 단위로 복사된다.

Table 2. Copy unit of the HFS+ Journal

Name	Copy Size
Volume Header	Volume Header Size
Allocation File	Block Size
Extent Overflow File	Node Size
Catalog File	Node Size
Attribute File	Node Size

block_list header의 첫 16 Bytes는 Table 3과 같이 block_list의 크기(byte_used)와 block_info의 개수(bnum_block)가 저장된다. 16Byte 이후 bnum_block에 명시된 수만큼 block_info가 연속적으로 위치한다. block_list header는 journal header에 명시된 blhdr_size 값만큼 고정된 크기를 가지며, 사용되지 않은 영역을 0x5A 값으로 패딩한다.

Table 3. Some of the Attributes in Block__List Header

Attribute	Offset(Byte)	Size(Byte)
num_block	0x02	0x02
byte_used	0x04	0x04
bnum	0x10	0x08
bsize	0x18	0x04

4. HFS+ 저널을 이용한 파일 복구

4.1 HFS+ 저널파일 파싱 알고리즘의 필요성

HFS+ 파일시스템은 저널의 구조 및 운영상 특징에 의해 일반적인 방식으로는 저널에 저장된 모든 데이터를 추출할 수 없다. 실제 시스템의 저널 구조는 Fig. 4와 같이 buffer의 시작이 분할된 block_list로 시작되며, 새로 저장되는 block_list에 의해 가장 오래된 block_list의 일부가 덮어써진다. 이는 저널 버퍼가 원형 버퍼로 운영되고, 그 내부에 저장되는 block_ list가 가변적인 크기를 갖기 때문이다.

이러한 상황에서 저널의 메타데이터는 단순히 가장 최근 발생한 트랜잭션의 처음과 마지막 block_list의 Offset 정보만을 가지고 있기 때문에 이외 다른 block_list에 접근하는 것이 불가능하다. 이는 가변적인 크기를 갖는 block_list와 저널이 원형 버퍼로 운영되기 때문이다. 먼저 가변크기를 갖는 block_list는 byte_used 속성을 이용하여 이어지는 block_list에 접근할 수 있지만, 이전에 저장된 block_list의 크기를 알 수 없어 이전 block_list들에 접근할 수 없다. 또한 Fig. 4와 같이 새롭게 저장되는 block_list에 의해 가장 오래된 block_list 일부가 덮어써지기 때문에 end 속성이 가리키는 Offset 이후에 위치한 block_list에 접근할 수 없는 한계가 존재한다.

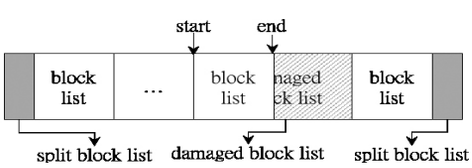

Fig. 4. Structure of HFS+ Journal in the Actual System

메타데이터만을 이용해선 저널에 저장된 모든 block_list들을 추출할 수 없으며, 결과적으로 추출된 block_list들의 수가 적을수록 삭제된 파일의 복구율은 낮아지기 때문에 모든 block_list들을 추출할 수 있는 알고리즘이 요구된다. 본 논문에서 제안하는 알고리즘은 저널의 메타데이터와 운영상의 특징을 이용하며, 해당 알고리즘을 통해 저널에 저장된 모든 block_list들을 추출할 수 있다.

4.2 HFS+ 저널파일 파싱 알고리즘

본 논문에서 제안하는 알고리즘의 핵심은 저널 상에 저장된 block_list들 중 가장 오래된 block_list(이하 old_block _list)를 찾는 것이다. old_block_list를 저널로부터 추출하기 위해 고안한 알고리즘은 Fig. 5와 같으며, 이에 사용된 속성 값은 Table 4와 같다. 해당 알고리즘을 이용하면 old_block_list를 추출할 수 있으며, 이후 byte_used 속성을 이용하여 연속되어 저장된 모든 block_list들을 추출할 수 있다.

Table 4. The Attributes that are Used in the HFS+ Journal Parsing Algorithm

Attribute Name	Description
End	Offset of the most recent block_list
blhdr_size	Attribute of journal header, Describe size of block_list header
n.bsize in block_info	Size of block N
byte_used	Total Size of block_list

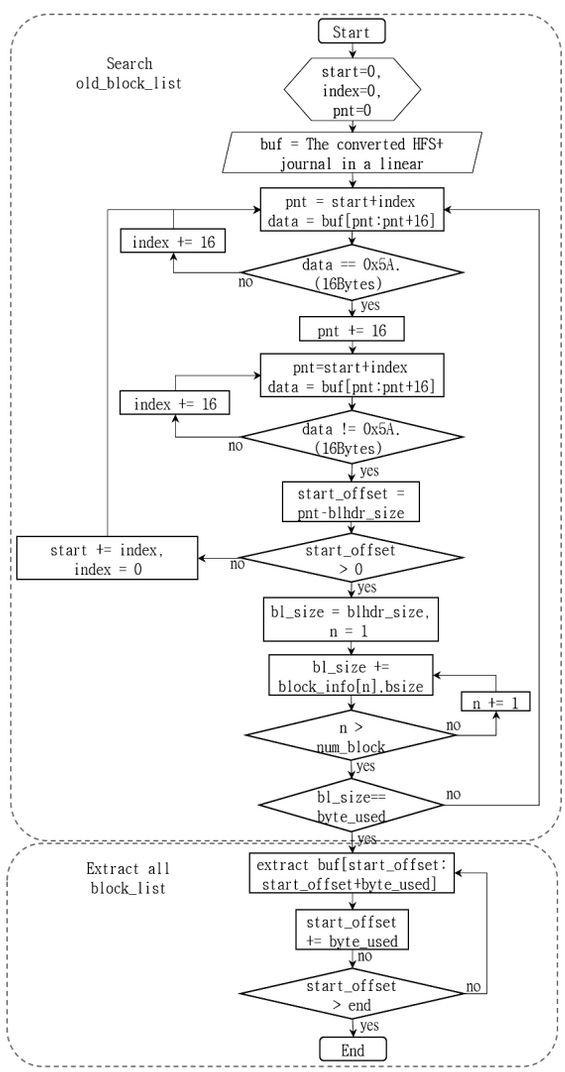

Fig. 5. Parsing Algorithm of HFS+ Journal File

실제 사용 중인 시스템의 저널 파일을 확인해보면 대부분 Fig. 4와 같이 block_list들 중 하나가 손상되어있다. 이는 새로운 block_list가 저장될 때 기존의 block_list를 덮어쓰기 때문이다. Fig. 4와 같은 상태에서 old_block_list는 손상된 block_list 다음에 위치한 block_list이다.

Fig. 4와 같은 저널 상태에서 분석의 편의를 위해 원형 버퍼로 사용되던 저널 파일을 선형 형태의 저널로 변환한다. 변환 방법은 다음과 같다. Fig. 4와 같이 구성된 저널 버퍼를 end 속성을 기준으로 이후의 데이터를 모두 저널의 앞쪽에 연결하면 Fig. 6과 같은 상태가 된다. 변환된 저널 버퍼는 손상된 block_list로 시작되며, 손상된 block_list 이후 모든 block_list가 연속되어 저장된다.

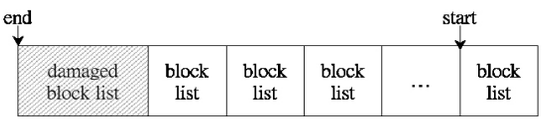

Fig. 6. Structure of the Linearly Changed HFS+ Journal

저널 버퍼로부터 old_block_list를 찾기 위해 Fig. 7과 같이 header 영역에 기록된 0x5A 패딩 값을 시그니처(16Byte 크기의 0x5A 이하 시그니처)로 활용한다. 먼저 선형 저널 버퍼(Fig. 6)의 시작으로부터 16 Bytes씩 추출하여 시그니처와 비교한다. 시그니처와 동일한 값이 추출됐을 때 block_list header의 패딩 영역이라 판단하고, 이후 더 이상 시그니처가 나오지 않을 때까지 검색을 진행한다. 이 과정에서 시그니처가 나오지 않는 영역을 찾은 뒤엔 해당 위치를 기준으로 block_list header 크기만큼 Offset을 앞으로 이동시킨다. 이는 고정된 크기를 갖는 block_list header의 특징을 활용한 것이다. 이동된 위치를 old_block_list의 시작 위치로 가정하고 두 가지 검증작업을 수행한다.

000B8800	FF 07 06 00 00 C2 00 00	46 2E 51 39 03 00 00 00
000B8810	00 00 00 00 00 00 00 00	00 00 00 00 26 AD 15 00
000B8820	10 00 00 00 00 00 00 00	00 10 00 00 FF FF FF FF
000B8830	E0 A9 01 00 00 00 00 00	00 10 00 00 98 69 96 78
000B8840	02 00 00 00 00 00 00 00	00 02 00 00 5E 31 E6 49
000B8850	18 00 00 00 00 00 00 00	00 10 00 00 7F 7F 7F 7F
000B8860	E8 A9 01 00 00 00 00 00	00 10 00 00 E6 88 8B EC
000B8870	5A 5A 5A 5A 5A 5A 5A 5A	5A 5A 5A 5A 5A 5A 5A 5A
000B8880	5A 5A 5A 5A 5A 5A 5A 5A	5A 5A 5A 5A 5A 5A 5A 5A
000B8890	5A 5A 5A 5A 5A 5A 5A 5A	5A 5A 5A 5A 5A 5A 5A 5A
000B88A0	5A 5A 5A 5A 5A 5A 5A 5A	5A 5A 5A 5A 5A 5A 5A 5A
000B88B0	5A 5A 5A 5A 5A 5A 5A 5A	5A 5A 5A 5A 5A 5A 5A 5A

Fig. 7. Padding Area in the Block_List Structure

이때 검증작업을 거치는 이유는 old_block_list라고 가정한 block_list가 손상된 block_list일 가능성이 존재하기 때문이다. 이는 Fig. 8에서와 같이 end 속성이 손상된 block_list의 어떤 영역을 가리키는지에 따라 결정된다. 만약 block_list의 header 영역을 가리킨다면, 가정된 old_block_list가 손상된 block_list임을 알 수 있다. 반면에 end 속성이 data buffer 영역을 가리키고 있었다면 가정된 old_block_list가 실제 old_block_list임을 판단할 수 있다.

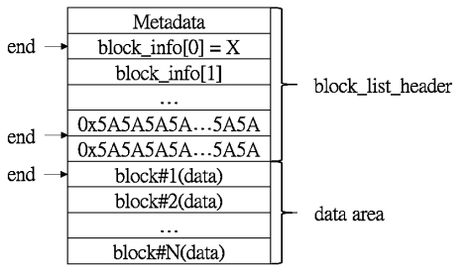

Fig. 8. The Location Pointed End Attribute in the HFS+ Journal

이러한 이유에 의해 검증 과정은 반드시 거쳐야 하며, 두 단계의 검증 과정이 필요하다. 첫째로 가정된 old_block_list offset이 0보다 같거나 커야 한다. 만약 offset이 0보다 작다면, 가정된 old_block_list는 손상된 block_list로 판단할 수 있다.

첫 번째 조건을 만족하는 경우 몇몇 속성의 연산을 통해 가정된 old_block_list가 실제 old_block_list인지 검증한다. 검증 방식은 Equation (1)과 같다. 먼저 journal header의 blhdr_size 값과 block_list에 저장된 모든 data 크기(block_info의 Bsize 속성)를 합하여 block_list의 크기를 계산한다. 이후 block_list의 byte_used 속성과 앞서 계산한 값을 비교하여 두 값이 동일하다면 가정된 old_block_list를 실제 old_block_list로 판단한다. 만약 검증 과정에서 두 조건을 하나라도 만족하지 않는다면 손상된 block_list를 기준으로 다시 시그니처 검색 과정을 수행한다. 지금까지 과정을 통해 old_block_list를 찾았다면 이후 연속된 block_list들을 byte_ used 속성을 통해 모두 추출할 수 있다.

$$byte\_used = blhdrsize + \sum_{k=1}^{n} block\_info[k].bSize \quad (1)$$

본 논문에서 제안한 알고리즘을 이용하면, 저널 버퍼에 저장된 모든 block_list들을 추출할 수 있다. 하지만 삭제된 파일을 복구하기 위해선 추가적으로 block_list들에 저장된 데이터를 Volume Header 및 5개 Special File 별로 분류해야 한다. 데이터 분류는 block_list 추출에 비해 비교적 간단하다. block_list 내부에 저장된 n 번째 block은 n 번째 block_info와 매핑된다. block_info는 num_block 속성을 통해 매칭되는 block이 파일시스템 어느 영역에 저장되었는지 나타내기 때문에 이를 통해 각 데이터를 분류할 수 있다.

지금까지 과정을 통해 추출 및 분류한 데이터 중 Extent Overflow File의 데이터와 Catalog File의 데이터를 이용하면, Non-Overflow 파일과 Overflow 파일 모두 복구할 수 있다.

4.3 파일 복구

HFS+ 저널부터 추출한 Extent Overflow File의 노드(leaf node)와 Catalog File의 노드(leaf node)를 이용하여 삭제된 파일의 복구를 수행한다. 먼저 각 노드로부터 파일 레코드를 추

출하여 파일의 삭제 여부를 판단한다. 이는 추출한 레코드의 File ID가 현재 파일시스템에 저장된 파일 중 동일한 File ID를 갖는지 비교하여 파악할 수 있다.

이후 삭제 파일의 레코드가 가리키는 extent 영역의 할당/미할당 여부를 확인하여 복구 가능성을 확인한다. 만약 레코드가 가리키는 영역이 현재 사용 중이라면 삭제된 파일의 데이터 일부가 덮어써졌기 때문에 파일의 완전한 복구는 불가능하며, 모든 extent 영역이 미사용 중이라면 삭제된 파일의 복구는 가능하다. 하지만 이 경우에도 특정 파일에 의해 extent 영역의 일부가 사용된 뒤 해제됐다면, 완전한 복구는 불가능하다.

일반 파일의 경우 Catalog File의 레코드만 추출해도 복구될 수 있지만, Overflow 파일의 경우 Extent Overflow File의 레코드와 Catalog File의 레코드를 모두 저널로부터 추출했을 때 파일이 복구될 수 있다. 레코드를 모두 추출했을 때 파일을 복구하는 방법은 파일시스템에 저장되어있는 파일을 추출하는 방법과 동일하다.

5. 실험 결과

실험에 사용된 시스템 사양은 Table 5와 같다.

Table 5. Specifications of the System Used in the Experiment

Product Name		Macbook Air A1466
OS Version		v10.10.5
OS Installation Time		2014-02-18 06:39:36
Volume Size	Total	230 GB
	Allocation	182.57 GB
	Unallocation	47.43 GB
Journal Size		24 MB

5.1 Extent Overflow 발생 비율

Table 5에 명시한 MAC 시스템의 Extent 분할 비율을 확인하였으며, 실험 결과는 Table 6과 같다. 4KB 이상 파일 중 단일 블록 파일(하나의 Extent 영역에 File Content 저장)은 전체 341,507개의 파일 중 337,169개로 가장 큰 비율을 차지하였으며, 두 개 블록으로 분할된 파일은 2,221개가 존재하였다. 또한 Overflow 파일은 663개로 앞서 언급한 두 개 파일 다음으로 높은 비율을 차지하였다.

실험을 통해 각 카테고리의 평균 크기를 확인한 결과 Overflow 파일이 가장 큰 크기 값을 가지고 있었다. 또한 각 Overflow 파일을 확인해봤을 때 비교적 크기가 큰 압축 파일, 문서 파일, 동영상 파일 등에서 Extent Overflow가 발생했음을 확인할 수 있었다. 반면에 시스템 상에 가장 많이 존재하는 단일 블록 파일의 경우 대부분 시스템 파일이 포함되었음을 확인할 수 있었다.

결과적으로 실제 사용자가 생성, 수정, 복사한 파일을 기준으로 비율을 측정한다면, Table 6의 결과는 크게 변동될 수 있다. 특히 해당 비율은 사용자가 시스템을 사용하는 방식에 크게 의존되므로 절대적인 수치를 계산하기 어렵다.

본 논문의 목적은 기존 기법에서 배제된 Overflow 파일 또한 복구 대상으로 포함시키는 것이다. 이러한 관점에서 봤을 때 Overflow 파일의 비율보다 해당 파일의 레코드 또한 저널에 저장될 수 있다는 점에 주목해야 한다. 즉, 본 논문에서 제안한 알고리즘을 이용하여 Overflow 파일 또한 복구될 수 있는 가능성이 존재한다. 다음 항목에서 실제 시스템 상에서 저널을 통해 Overflow 파일을 복구한 실험 결과에 대해 설명한다.

Table 6. File Extent Ratio of the Experimental System

Extent	File Count	Average of File Size
1	337169	0.1 MB (130.6 KB)
2	2221	3.4 MB (3431.2 KB)
3	553	9.2 MB (9468.1 KB)
4	345	8.4 MB (8676.3 KB)
5	206	8.5 MB (8806.2 KB)
6	145	15.8 MB (16183.4 KB)
7	114	14.5 MB (14816.7 KB)
8	94	13.9 MB (14281.3 KB)
Overflow	663	91.9 MB (94175.0 KB)
Total	341,507	-

5.2 Journal을 이용한 삭제된 파일 복구 결과

실험을 위해 아래와 같은 절차를 진행 한 뒤 MAC 시스템의 이미지를 추출하였으며, iv. 단계에서 Table 7에 명시한 4개의 파일을 순차적으로 삭제하였다. 이후 아래 단계를 거쳐 추출한 이미지 파일을 대상으로 기존 연구의 결과물들과 본 논문에서 제시한 알고리즘을 적용하여 삭제된 Overflow 파일의 복구결과를 비교분석하였다.

i. MAC 시스템의 이미지 추출
ii. 시스템 상에 저장된 Overflow 파일 확인
iii. 동일한 Node(폴더) 내에 포함된 Overflow 파일의 레코드들을 확인
iv. iii.에서 확인한 Overflow 파일들을 순차적으로 삭제(저널 상에 삭제된 Overflow 파일의 레코드가 남는 상황을 가정)
v. 이미지 추출

Table 7. The Files Deleted from the Experimental System

Name	File ID	Size(Byte)	File Type
해쉬알고리즘을 이용한 전자서명 구현.pdf	468639	3,728,895	Overflow
20100430154612577 9.pdf	468638	5,433,865	Overflow
2014572416_.hwp	468637	5,000,192	Overflow
시스템_아티팩트_분석.pdf	468744	6,939,016	Overflow

본 논문에서 제안한 알고리즘을 적용한 결과는 Table 8과 같다. 삭제된 4개의 Overflow 파일 중 3개의 파일을 복구하였으며, 이때 Overflow 파일의 Catalog File 레코드뿐만 아니라

Extent Overflow File의 레코드 또한 저널로부터 추출하여 파일의 완전한 복구가 가능하였다. 추출된 레코드를 이용해 파일을 재조합하는 과정은 단순히 Catalog File 레코드가 가리키는 영역 뒤에 Extent Overflow 레코드가 가리키는 영역들을 순서에 맞춰 병합하면 되며, 이때 Extent Overflow 레코드들 간의 순서는 Start Block 속성에 의해 결정된다. Start Block 속성은 해당 레코드가 가리키는 Extent 영역의 데이터들이 파일의 어떤 Offset에 위치해야 하는지를 나타낸다.

Table 8. Overflow Files Recovered Using the HFS+ Parsing Algorithm

File ID	Start Block(Offset)	Record가 속한 파일
468637	0	Catalog
468637	778	Extent Overflow
468637	1073	Extent Overflow
468638	0	Catalog
468638	707	Extent Overflow
468638	944	Extent Overflow
468638	977	Extent Overflow
468638	1018	Extent Overflow
468638	1170	Extent Overflow
468638	1265	Extent Overflow
468639	0	Catalog
468639	641	Extent Overflow
468639	870	Extent Overflow

두 번째 복구실험으로 동일한 이미지에 대해서 Kzamiya가 개발한 Enpack인 HFS+ Journal Parser(Enpack ver. 1.0, Encase v7.11.01 사용)[6]를 적용하였으며, 복구 결과는 Table 9와 같다. 삭제된 4개의 Overflow 파일 중 2개 파일에 대한 레코드만 추출하였으며, 이때 Extent Overflow File 레코드를 제외한 Catalog File 레코드만을 복구하여 삭제된 파일의 완전한 복구가 불가능하였다. 특히 PDF 파일의 경우 파일의 끝에 내부 메타데이터의 위치를 저장하는 XRef 구조체가 존재[11]하기 때문에 Enpack을 통해 복구된 파일의 경우 파일의 열람조차 불가능하였다.

Table 9. Overflow Files Recovered Using the Enpack Created by Kzamiya

File ID	Start Block(Offset)	Record가 속한 파일
468638	0	Catalog
468639	0	Catalog

마지막으로 동일한 이미지에 대해 Data Protection에 포함된 Journal Carving[9] 도구를 적용하였으며 실행 결과는 Table 10과 같다. 해당 도구는 3개 파일에 대한 레코드를 복구하였지만, 이 또한 Catalog File 레코드만을 추출하여 Enpack과 동일한 결과를 보였다.

Table 10. Overflow Files Recovered Using the Data Protection

File ID	Start Block(Offset)	Record가 속한 파일
468637	0	Catalog
468638	0	Catalog
468639	0	Catalog

기존에 연구된 기법의 경우 HFS+ 저널을 카빙하여 Catalog File 레코드만을 추출하거나 추출과정에서 Extent Overflow File 레코드 자체를 배제시키기 때문에 삭제된 Overflow 파일의 완전한 복구는 불가능하였다. 하지만 본 논문에서 제안한 알고리즘의 경우 카빙 방식이 아닌 저널의 상세한 파싱을 통해 데이터를 분류하기 때문에 Catalog File 레코드뿐만 아니라 Extent Overflow File 레코드 또한 모두 추출하였으며, 이를 통해 Non-Overflow 파일에 더불어 Overflow 파일 또한 저널로부터 복구될 수 있음을 실험을 통해 입증하였다.

6. 결 론

디지털 포렌식 조사 과정에서 삭제된 파일을 복구하는 것은 아주 중요하다. 이때 복구되는 파일은 시스템 아티팩트가 될 수도 있고, 사건을 결정지을 수 있는 특정 파일이 될 수 있다. 이러한 이유에 의해 NTFS, HFS+, EXT 등의 파일시스템에 대한 구조 및 삭제된 파일의 복구 기법과 관련된 연구가 꾸준히 이루어져 왔다.

이때 HFS+ 파일시스템의 경우 카빙 방식을 이용한 파일 복구가 가능하지만, 파일이 분할되어 저장된 경우 파일의 일부분만이 복구되거나 파일 전체를 복구할 수 없는 상황이 발생된다. 분할된 파일은 파일시스템에 남아있는 메타데이터를 이용한 복구가 필요하지만, HFS+는 운영의 특성상 파일이 삭제되면 해당 파일의 메타데이터 또한 파일시스템 상에서 삭제된다.

기존에 연구된 기법 중 HFS+의 저널을 이용하는 기법이 존재하지만 해당 기법은 HFS+가 정의한 두 파일 유형 중 Overflow 파일은 복구 대상에서 배제되었다.

본 논문은 HFS+ 파일시스템으로부터 HFS+가 정의하는 두유형의 파일을 모두 복구할 수 있는 기법으로써 HFS+ 저널 파일을 상세히 파싱할 수 있는 알고리즘을 제시하였다. 해당 알고리즘은 HFS+ 저널을 상세히 분석하여 저널의 저장된 데이터를 스페셜 파일별로 분류시킨다. 이를 통해 Catalog File 레코드 및 Extent Overflow File 레코드를 각각 수집하고, 해당 레코드들을 이용하여 Non-Overflow 파일과 Overflow 파일 모두 복구할 수 있다.

결과적으로 본 논문은 HFS+ 저널을 이용한 복구 기법의 새로운 기준을 제시하며, 이는 기존 기법에서 배제하던 파일까지 모두 포함시킨다. 현재까지 HFS+의 삭제 파일을 복구하는 도구 중 Overflow 파일을 포함하여 복구시키는 도구는 없다. 본 논문에서 제시한 알고리즘 및 복구 기법을 이용한 복구 도구를 개발하였으며, 이는 포렌식 분석 과정에서 삭제된 파일을 복구하는데 도움이 될 것으로 예상된다.

References

[1] Z. Kai, C. En, and G. Qinquan, "Analysis and implementation of NTFS file system based on computer forensics," *Education Technology and Computer Science (ETCS),* Vol.1, pp.325-328, 2010.

[2] Byeongyeong Yoo, et al, "A Study on a Carving Method for Deleted NTFS Compressed Files," *Human-Centric Computing (HumanCom), 2010 3rd International Conference on. IEEE*, pp.1-6, 2010.

[3] R. A. Joyce, J. Powers, and F. Adelstein, "MEGA: A tool for Mac OS X operating system and application forensics," *Digital Investigation*, Vol.5, pp.S83-S90, 2008.

[4] A. Case and G. G. Richard, "Advancing Mac OS X rootkit detection," *Digital Investigation*, Vol.14, pp.S25-S33, 2015.

[5] A. Burghardt and A. J. Feldman, "Using the HFS+ journal for deleted file recovery," *Digital Investigation*, Vol.5, pp.S76-S82, 2008.

[6] HFS+ Deleted File Recovery EnScript [Internet], https://www.kazamiya.net/en/HFSJournalParser.

[7] Apple [Internet], https://developer.apple.com/legacy/library/technotes/tn/tn1150.html.

[8] Apple [Internet], hfs_format.h, http://opensource.apple.com//source/xnu/xnu-1456.1.26/bsd/hfs/hfs_format.h.

[9] iOS forensic tools [Internet], https://code.google.com/p/iphone-dataprotection/wiki/HFSJournalCarving.

[10] D. Comer, "Ubiquitous B-tree," *ACM Computing Surveys (CSUR)*, Vol.11, No.2, pp.121-137, 1979.

[11] Adobe Systems Incorporated, Document management - Portable document format - Part 1: PDF 1.7, Adobe Systems Incorporated, 2008.

방 승 규

e-mail : bangkert89@naver.com
2015년 목포대학교 정보보호학과(학사)
2015년~현 재 고려대학교 정보보호대학원 정보보호학과 석사과정
관심분야 : Digital Forensic, Reverse Engineering

전 상 준

e-mail : heros86@korea.ac.kr
2010년 고려대학교 정보경영공학과(학사)
2010년~현 재 고려대학교 정보보호대학원 정보보호학과 석·박사통합과정수료
관심분야 : Reverse Engineering, Data Carving, Mobile Forensic

김 도 현

e-mail : exdus84@korea.ac.kr
2011년 서울과학기술대학교 컴퓨터공학과 (학사)
2013년 고려대학교 정보보호대학원 정보보호학과(석사)
2013년~현 재 고려대학교 정보보호대학원 정보보호학과 박사과정수료
관심분야 : Digital Forensics, Mobile Forensics

이 상 진

e-mail : sangjin@korea.ac.kr
1987년 고려대학교 수학과(학사)
1989년 고려대학교 수학과(석사)
1994년 고려대학교 수학과(박사)
1989년~1999년 ETRI 선임연구원
1999년~현 재 고려대학교 정보보호대학원 교수
2008년~현 재 고려대학교 디지털포렌식연구센터 센터장
관심분야 : Digital Forensic, Steganography, Hash Function