Journal of The Korea Institute of Information Security & Cryptology(JKIISC) ISSN 1598-3986(Print)
VOL.23, NO.3, June 2013 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2013.23.3.417

안드로이드 운영체제의 Ext4 파일 시스템에서 삭제 파일 카빙 기법*

김 도 현,† 박 정 흠, 이 상 진‡
고려대학교 정보보호연구원

File Carving for Ext4 File System on Android OS*

Dohyun Kim,† Jungheum Park, Sangjin Lee‡
Center for Information Security Technologies, Korea University

요 약

많은 리눅스 운영체제와 안드로이드 운영체제에서 Ext4 파일 시스템을 사용하고 있어 디지털 포렌식 관점에서 Ext4 파일 시스템 상의 삭제 파일 복구가 현안이 되고 있다. 본 논문은 Ext4 파일 시스템에서 파일의 할당, 삭제 시 특징을 분석함으로써 삭제된 파일에 대한 복구 방법을 제시하였다. 특히 안드로이드 운영체제의 파일 할당 정책을 바탕으로 비 할당 영역에 조각나 있는 삭제된 파일에 대한 복구 방법을 제시한다.

ABSTRACT

A lot of OS(Operating Systems) such as Linux and Android selected Ext4 as the official file system. Therefore, a recovery of deleted file from Ext4 is becoming a pending issue. In this paper, we suggest how to recover the deleted file by analyzing the entire structure of Ext4 file system, the study of metadata area, the distinct feature when file is assigned and deleted. Particularly, we focus on studying the features of file which is assigned in Ext4 file system in Android OS and also suggest the method to recover the deleted file that is fragmented from the un-allocated area.

Keywords: Digital Forensics, Android Forensics, Ext4 File System, File Carving

I. 서 론

대부분의 리눅스 시스템과 안드로이드 운영체제의 파일 시스템이 Ext4로 대체되고 있다. 그동안 서버 등의 특정 시스템에서만 사용되던 Ext4 파일 시스템이 구글의 안드로이드 운영체제의 공식 파일 시스템으로 지정되면서 스마트폰, 태블릿 PC, mp3 플레이어 등 일상에서 사용하는 여러 임베디드 기기에서 사용률이 지속적으로 증가할 것으로 예상된다.

스마트폰과 같이 일상생활과 밀접한 임베디드 기기 내부에는 여러 가지 포맷으로 전화기록, 문자메시지, 이메일, 인터넷 검색 기록 등의 사용자 데이터가 많이 존재하며 범죄자는 사건과 관련된 민감한 데이터를 의도적으로 삭제할 가능성이 높다. 따라서 디지털 포렌식 관점에서 기기 내부의 활성 데이터뿐만 아니라 삭제된 데이터에 대한 분석이 필요하며 이러한 관점에서 임베디드 기기에서 사용되는 Ext4 파일 시스템에서 삭제된 데이터의 복구에 대한 연구가 매우 중요하다.

Ext4 파일 시스템은 일반적으로 윈도우에서 사용되는 NTFS와 FAT 파일 시스템과는 달리 파일이 삭제될 때 메타데이터의 데이터 블록 포인터가 초기화되므로 메타데이터를 활용한 파일 복구가 불가능하다.

접수일(2013년 2월 18일), 게재확정일(2013년 3월 19일)
* 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업의 일환으로 수행되었습니다.
〔10035157, 실시간 분석을 위한 디지털 포렌식 기술 개발〕
† 주저자, exdus84@korea.ac.kr
‡ 교신저자, sangjin@korea.ac.kr(Corresponding author)

또한, I/O의 효율성을 높이기 위하여 블록 포인터를 extents 구조체로 저장함으로써 Ext3 파일 시스템에 비하여 파일의 단편화는 줄어들었지만 Ext3에서 사용하던 파일 복구 방식을 사용하지 못하게 되었다[1]. Ext4 파일 시스템에서 삭제된 파일 복구에 대한 연구들이 진행되었지만 extents 구조체로부터 삭제된 파일의 데이터 블록 정보가 초기화되기 때문에 복구가 불가능 하다는 사실을 확인하고[1], journal log 영역에 남아있는 가장 최근의 삭제된 파일만 확인이 가능하다는 등의 많은 한계점이 존재한다[2]. 또한, Ext4 파일 시스템이 안드로이드 운영체제에서 사용될 경우 일반적인 리눅스 시스템과는 사용 방식이 다르고 그에 따라 내부 데이터의 저장 방식 및 종류들도 다르기 때문에 이에 대한 추가적인 연구가 필요하다.

안드로이드 운영체제는 주로 애플리케이션 위주로 동작되기 때문에 새로운 파일이 생성되는 것 보다 애플리케이션이 설치되면서 생성된 이미 존재하는 파일에 대한 수정이 빈번하게 발생된다는 특징이 있다. 따라서 파일에 새로운 데이터가 추가될 때 이전 데이터 블록과 연속적으로 저장되지 못하면서 많은 단편화가 발생하며 실제로 안드로이드폰에서 사용자 데이터가 존재하는 대부분의 SQLite 데이터베이스 파일은 여러 조각으로 단편화되어 있다.

본 논문은 Ext4 파일 시스템의 전체적인 구조와 파일시스템의 메타데이터에 해당하는 영역에 대한 연구 및 파일의 할당, 삭제 시 특징을 분석하였다. 또한, 추가적으로 안드로이드에서 사용되는 Ext4 파일 시스템에서 파일이 할당되는 특징을 통해 파일의 단편화 특성을 연구함으로써 비 할당 영역으로부터 조각나 있는 삭제된 파일에 대한 카빙 방법을 제시하였다.

본 논문은 다음과 같이 구성된다. 2절에서는 본 논문과 관련된 기존의 연구들에 대해서 설명하고, 3절에서는 Ext4 파일 시스템의 구조를 설명한다. 4절에서는 Ext4 파일 시스템의 특성을 설명하고, 5절에서는 안드로이드 운영체제에 탑재된 Ext4 파일 시스템의 특성에 대해서 설명한다. 6절에서는 삭제된 파일에 대한 카빙 방법을 소개하며, 7절에서 결론을 내리고 향후 연구계획을 밝힌다.

II. 관련 연구

Ext2 파일 시스템은 NTFS, FAT 파일 시스템처럼 파일 삭제 시 메타데이터의 정보가 지워지지 않기 때문에 삭제 파일 복구에 대한 연구들이 많이 진행되었고[3][4], 그 결과를 토대로 비교적 쉽게 파일 복구를 할 수 있었다. 하지만 Ext3 파일 시스템부터는 이러한 메타데이터의 정보를 초기화하기 때문에 다른 방식으로 복구를 수행해야 한다. Ext3 파일 시스템에서 삭제된 파일 복구에 관한 연구는 많이 진행되어 왔는데 주로 Ext3 파일 시스템이 사용한 파일 저장 방식인 inode table에 존재하는 indirect block pointer의 정보를 통해 비할당 영역에서 조각난 파일들의 offset을 추정하여 복구하는 연구가 진행되었고[5][6][7], ext3grep, extcarve 와 같은 대부분의 Ext3 파일 시스템 복구 도구는 이러한 방식을 사용한다[8][9]. 하지만 Ext4 파일 시스템에서는 Ext3와는 달리 파일의 데이터 블록 포인터 정보를 extents 구조체를 사용하여 저장하기 때문에 Ext3 파일 시스템에서 사용한 파일 복구 방법을 적용할 수 없다.

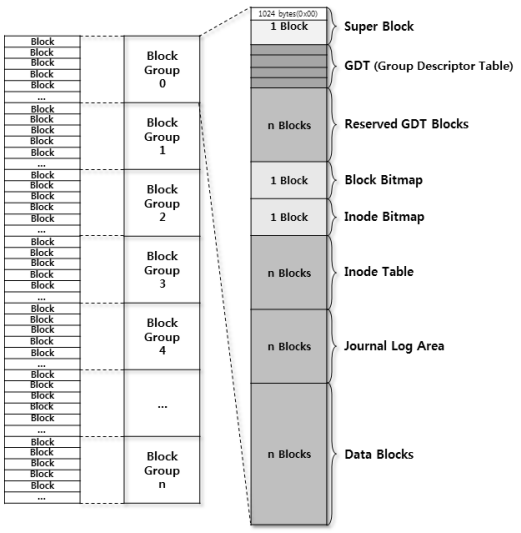
Ext4 파일 시스템에서 삭제된 파일의 복구에 대한 연구로, 삭제된 파일의 inode table에 남아있는 extents index node의 offset을 획득하여 삭제된 파일을 복구하는 연구가 진행되었지만, 파일의 extents tree의 깊이가 0일 경우에는 extents index node가 존재하지 않고 extents tree의 깊이가 1 이상일 경우 extents index node의 offset을 획득하더라도 실제 블록들의 offset 정보는 초기화되기 때문에 삭제된 파일의 데이터 블록을 확인하기 어렵다[1].

Ext3 파일 시스템부터 존재하는 journal log area의 로그 정보 중 데이터 삭제 트랜잭션과 관련된 로그 내부에 존재하는 block bitmap 정보를 해석하여 삭제된 블록을 복구하는 방법도 진행되었다[2]. 하지만, journal log area에는 사용자로 인해 생성된 로그뿐만 아니라 운영체제가 동작하면서 생성되는 로그들도 많이 존재하며 journal log area의 크기도 제한적이기 때문에 사용자가 삭제한 데이터들에 대한 많은 정보를 획득하기 어렵다. 따라서 현재까지는 비할당 영역을 수집하여 파일 시그니처를 기반으로 삭제된 파일 복구를 수행하는 방법이 가장 널리 사용되고 있으며 기존의 extcarve, giis-ext4, Stellar Phonix Linux Data recovery, EaseUS Data Recovery Wizard등의 Ext4 파일 시스템 복구 도구들도 이러한 방식을 사용한다[9][10][11][12]. 하지만 복구를 지원하는 파일 포맷이 제한적이며 복구 성공률이 낮고 리눅스 시스템에서 마운트된 파티션만 지원하는 등의 기능상 여러 가지 제약이 존재한다. 또한, 안드로이드 운영체제에서 사용된 Ext4 파일 시스템일 경우 많은 파일들이 단편화되는 특성 때문에 파

일 복구 성공률이 더욱 낮아진다. 따라서 안드로이드 운영체제에서 사용된 Ext4 파일 시스템의 파일 할당 및 조각나는 특성을 이용하여 단편화된 파일 분석 및 복구에 대한 연구가 필요하다.

III. Ext4 파일 시스템의 구조

본 절에서는 Ext4 파일 시스템의 전체적인 구조와 메타데이터 영역에 대해서 알아본다. Ext4 파일 시스템은 데이터 저장의 최소 단위인 블록들로 이루어져 있으며 블록의 크기는 1KB, 2KB, 4KB 중 하나를 선택하여 사용할 수 있지만 기본적으로는 4KB를 사용한다. Ext4 파일 시스템은 데이터를 효과적으로 관리하기 위해 [그림 1]과 같이 Ext4 파일 시스템 내부의 모든 블록들을 여러 개의 블록 그룹으로 묶어서 관리한다.



[그림 1] Ext4 파일 시스템 구조

하나의 블록 그룹은 최대 32,768개의 블록을 관리하기 때문에 블록의 크기가 4KB일 경우 하나의 블록 그룹의 크기는 최대 128MB이며, 만약 Ext4 파일 시스템의 파티션의 크기가 1GB(1,048,576MB)일 경우 8개의 블록 그룹(128MB × 8 = 1,048,576MB)이 존재한다.

블록 그룹 내부에는 super block, GDT(Group Descript Table), block bitmap, inode bitmap, inode table, journal log area 등의 메타데이터 영역들이 존재한다. super block은 블록의 크기, 전체 블록의 개수, inode table의 크기, 각 블록 그룹 당 inode table의 개수 등과 같은 파일 시스템의 전체적인 정보를 담고 있으며 이 영역은 블록 그룹 0번, 2n-1번(n ≧ 1)에 존재한다. GDT에는 각 블록 그룹에 존재하는 block bitmap, inode bitmap, inode table 영역의 위치 정보가 존재한다. block bitmap과 inode bitmap 영역에는 해당 블록 그룹 내부의 블록들과 inode들의 할당 정보를 bit단위로 mapping하여 저장된다. inode table 영역은 해당 블록 그룹 안의 모든 데이터에 대한 inode table들이 inode 번호에 따라 순서대로 존재하며 각각의 inode table에는 데이터의 생성, 접근, 변경, 삭제 시간 등의 시간정보와 크기, 실제 데이터가 존재하는 블록 포인터 등이 존재한다. 마지막으로 journal log area에는 파일시스템에서 발생한 트랜잭션들에 대한 로그들이 존재하며 다른 영역들과는 달리 특정 블록 그룹에만 존재하고 이 영역의 위치와 크기는 super block에 저장되어 있다. 이러한 메타데이터 영역들 다음부터 data blocks 영역이 존재하는데, 이 영역에는 데이터의 inode 번호, 이름 등의 메타데이터가 저장되어 있는 directory entry에 해당하는 블록과 실제 데이터에 해당하는 블록들이 존재한다[13].

IV. Ext4 파일 시스템의 특성

본 절에서는 Ext4 파일 시스템이 파일을 할당하는 방식과 파일의 단편화 현상 등의 특징들을 알아보고 파일을 삭제할 때 파일 시스템의 내부 메타데이터 영역에 나타나는 변화에 대해서 알아본다.

4.1 파일 할당

3절에서 살펴본 것과 같이 Ext4 파일 시스템은 여러 개의 블록그룹들로 구성되며 블록 그룹 내의 메타데이터 영역(super block, GDT, block bitmap, inode bitmap, inode table, journal log area)들을 제외한 약 127MB는 directory entry와 실제 데이터를 저장하는데 사용된다. Ext4 파일 시스템은 파일의 단편화를 최소화하기 위하여 파일을 최대한 연속적인 블록에 저장하기 때문에 이론적으로 하나의 파일은 최대 127MB까지 단편화가 발생하지 않는다.

Ext4 파일 시스템은 파일을 블록 그룹 내의 앞쪽 블록부터 차례로 저장하는데 I/O의 효율성을 위해 같

은 디렉터리내의 파일들은 같은 블록 그룹에 저장하며 추가적으로 저장될 파일들을 위해 블록 그룹 내의 비할당 영역을 모두 사용하지 않는다. 따라서 데이터의 저장을 위해 블록 그룹의 앞쪽 공간을 먼저 사용하고 해당 블록 그룹의 비할당 영역이 어느 정도 줄어들게 되면 다음에 저장되는 파일은 다른 블록 그룹에 저장한다. 만약 파일이 조각나는 경우 I/O의 효율성을 위해 최대한 조각들을 같은 블록 그룹 내에 존재하도록 한다. 이때 비할당 영역의 앞쪽부터 조각난 데이터가 저장되기 보다는 해당 파일과 가까운 곳에 위치한 비할당 영역에 데이터를 저장한다[14].

Ext4 파일 시스템은 inode table 내부에 extents 구조체를 사용하여 실제 파일 할당 정보를 저장하는데 하나의 extent는 12bytes로 구성되며 이 구조체를 통해 실제 데이터가 몇 번 offset으로부터 몇 개의 블록을 할당하여 저장되어 있는지 알 수 있다. inode table에는 최대 4개의 extent가 존재할 수 있고 최초의 extent 앞에는 12bytes 크기의 extents header가 존재하며 이 영역의 정보를 통해 inode table내에 총 몇 개의 extent가 존재하는지 알 수 있다. [표 1]은 extents header의 구조다.

[표 1] extents header 구조[13]

바이트	항 목	설 명	값
0-1	magic number	시그니처	0x0A F3
2-3	number of extents	extents 개수	가변적
4-5	max number of extents	extensts 최대 개수	가변적
6-7	depth of tree	트리의 깊이	가변적
8-11	generation ID	extents ID	가변적

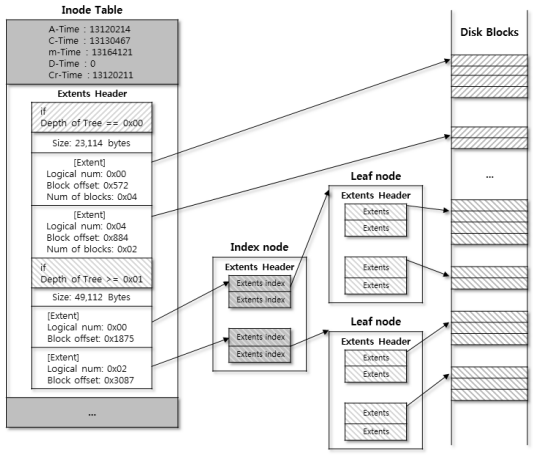
[표 2]는 extents header의 'depth of tree' 항목의 값에 따라 달라지는 extents의 구조에 대한 설명이며, [그림 2]는 extents 정보에 따른 데이터의 포인터 정보를 그림으로 도식화한 것이다[15][16].

저장하려는 파일이 파일 시스템에 의해 '대용량 파일' 로 분류되거나 '조각난 파일'일 경우 4개 이상의 extents가 필요할 수 있다. 이러한 경우 extents tree를 사용하여 데이터를 저장하는데 tree의 깊이에 따라 extents에 존재하는 정보가 달라진다.

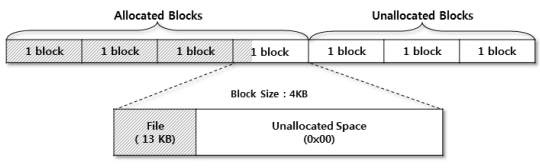
tree의 깊이 정보는 extents header 구조체의 'depth of tree' 항목에 저장된다. 만약 tree의 깊이가 0일 경우에는 [그림 2]와 같이 inode table에 존재하는 extents 구조체에 데이터가 존재하는 실제 블록의 offset이 저장되며 깊이가 1이상일 경우에는 index node에 실제 데이터 블록의 offset을 참조하는 값이 저장된다. 즉, 깊이가 1일 경우에는 index node에 실제 데이터 블록의 offset이 저장되지만 깊이가 2일 경우에는 index node에 leaf node의 offset이 저장되고 leaf node에 실제 데이터 블록의 offset이 저장된다.

[표 2] extents 구조[13]

오프셋	크기	항 목 (설명)	값
0	4	Logical Block Number	가변적
Depth of Tree의 항목이 0일 경우			
4	2	Number of Blocks in Extents	가변적
6	2	Upper 16 bits of Pysical Block Address	가변적
8	4	Lower 32 bits of Physical Block Address	가변적
Depth of Tree의 항목이 1이상일 경우			
4	4	Lower 32 bits of Physical Block Address	가변적
8	2	Upper 16 bits of Physical Block Address	가변적
10	2	Not Used (사용 안함)	0x00



[그림 2] 할당된 파일의 inode table의 extents 구조



[그림 3] 데이터의 크기가 블록의 배수가 아닐 경우

Ext4 파일 시스템은 〔그림 3〕과 같이 데이터의 크기가 블록 크기의 배수가 아닐 경우, 데이터가 저장되는 마지막 블록의 비할당 영역을 '0x00'으로 초기화하는데 이러한 영역은 데이터의 마지막을 판단할 수 있는 기준으로 사용될 수 있다. 블록의 크기는 4KB, 데이터의 크기는 13KB로 가정하였다.

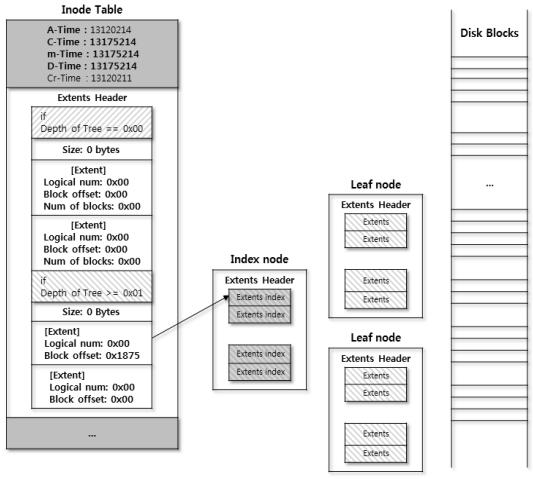
4.2 파일 단편화

Ext4 파일 시스템은 I/O의 효율성을 위해 최대한 파일 단편화를 회피하지만 단편화가 발생되는 이유는 크게 두 가지가 있다. 첫째, 생성되는 파일의 크기가 127MB 이상이거나 해당 블록그룹에서 관리하는 데이터 블록에 더 이상 저장할 공간이 없는 경우에는 파일이 조각나서 저장되며 이러한 경우 두 개 이상의 extents를 사용하게 된다. 둘째, 이미 존재하는 파일의 내용이 수정되면서 내부 데이터가 증가할 경우 추가된 데이터는 기존 파일의 데이터 블록에 연속적으로 저장되지 못하고 다른 영역으로 조각나서 저장될 수 있다. 첫 번째의 경우는 Ext4 파일 시스템을 사용하는 일반적인 리눅스 시스템과 안드로이드 운영체제에서 비슷하게 발생하지만 두 번째의 경우는 주로 안드로이드 운영체제에서 발생하며 그 빈도가 매우 높다. 이러한 경우는 5.2에서 실제 안드로이드폰의 내부 데이터를 통해 확인한다.

4.3 파일 삭제

Ext4 파일 시스템에서는 〔그림 4〕와 같이 extents tree의 깊이가 0인 파일과 1 이상인 파일이 삭제될 경우 메타데이터의 변화가 다르며 〔그림 2〕와 비교하면 파일이 할당되어 있을 때와의 변화된 사항을 쉽게 확인할 수 있다.

extents tree의 깊이가 0인 파일이 삭제된 경우 해당 파일에 대응되는 inode table의 extents 정보인 실제 데이터가 저장되어 있는 블록의 offset과 파일의 크기 정보가 초기화된다. 또한, 타임스탬프 중 inode 변경 시간, 파일 내용 수정 시간이 모두 파일 삭제 시간으로 변경되므로 삭제된 파일의 복구가 힘들고 파일이 생성된 시간과 삭제된 시간 외의 타임스탬프 정보 또한 알기 어렵다. 해당 파일의 extents tree의 깊이가 1이상인 경우에는 〔그림 4〕에서의 포인터와 같이 inode table의 exetnts정보에서 index node가 존재하는 가장 처음의 위치 offset 정보는 남아있으나 index node가 참조하는 실제 데이터 블록의 offset 및 타임스탬프 정보들은 모두 초기화 되므로 실제 데이터 조각들을 알아낼 수 없기 때문에 파일 복구에 큰 의미를 갖지 못한다〔1〕.



〔그림 4〕 삭제된 파일의 inode table의 extents 구조

V. 안드로이드 Ext4 파일 시스템의 특성

본 절에서는 안드로이드 운영체제의 특징 및 그로 인해 발생되는 안드로이드 운영체제에서 사용된 Ext4 파일 시스템의 내부 데이터의 종류 및 할당 특성에 대해서 알아본다. 특히, 사용자 데이터가 존재하는 XML, SQLite 데이터베이스 파일에 대해 자세히 알아본다.

5.1 안드로이드 운영체제와 Ext4 파일 시스템

안드로이드 운영체제는 내부에 여러 개의 파티션들이 존재하며 현재 가장 많이 사용되는 안드로이드 버전 4.0(Ice Cream Sandwich)이후 /sdcard 파티션을 제외한 시스템과 관련된 데이터들이 있는 /system, /efs, /tombstones, /cache 파티션, 사용자 데이터들과 관련된 데이터들이 있는 /data 파티션 등이 Ext4 파일 시스템을 사용한다. 안드로이드는 일반 PC처럼 애플리케이션 위주로 동작되지만 애플리케이션과 관련 없는 데이터들은(다운받은 문서, 사진 파일 등) vFAT 또는 exFAT 파일 시스템으로 이루어진 외장 메모리(/sdcard 파티션)로 저장되고 애플리케이션을 사용하며 생성되는 데이터들은(SQLite

〔표 3〕 안드로이드 Ext4 파일시스템 내부 파일 분류

분 류	파일 포멧	파일 개수	점유율(%)
사용자 데이터 관련 파일	SQLite database, XML	4,125	30.80
애플리케이션 관련 파일	APK, CSS, DAT, DEX, ELF, HTML, JS	3,472	25.92
파일시스템 메타데이터	directory entry	3,213	23.99
그래픽 관련 파일	BMP, GIF, JPG, PNG	2,583	19.29

데이터베이스, XML 파일 등) /data 파티션에 저장된다. 따라서 애플리케이션과 관련된 대부분의 파일은 Ext4 파일 시스템 영역에 저장된다.

본 논문에서는 실질적인 테스트를 위해 실제로 약 15개월 정도 사용한 삼성 안드로이드폰인 갤럭시 S2(SHW-M250S)와 약 12개월 정도 사용한 갤럭시 노트(SHV-E160S)의 내부 데이터를 테스트 세트로 사용하였다. 갤럭시 S2에서는 총 6,074개, 갤럭시 노트에서는 총 7,319개 파일이 존재하였다. 〔표 3〕은 테스트 세트 내부의 파일들을 사용자 데이터와 관련된 파일, 애플리케이션 구동과 관련된 파일, 파일시스템의 메타데이터, 그래픽 관련 파일 4가지 분류로 정리한 것으로 일반 PC에 비해 파일의 종류가 한정적인 것을 알 수 있고 내부 데이터 중에서 사용자 데이터와 관련된 파일은 30.80% 정도임을 알 수 있다.

테스트 세트의 내부 데이터 중에서 조각나지 않은 데이터는 92.18%이며, 나머지 7.82%만 2개 이상의 블록으로 조각나서 존재한다. 조각난 데이터는 대부분 SQLite database 파일과 애플리케이션이 구동될 때 사용되는 DAT 파일이며, 〔표 4〕는 테스트 세트 내부의 데이터들 중에서 사용자 데이터와 관련된 파일(SQLite database, XML)과 그래픽 관련 파일의 크기 및 단편화 정도를 정리한 것이다. 〔표 4〕를 통해 XML과 그래픽 관련 파일(BMP, GIF, JPG, PNG)은 크기와 상관없이 단편화가 거의 발생하지 않는 것을 알 수 있고, SQLite 데이터베이스 파일은 대부분 단편화가 발생한다는 것을 알 수 있다.

5.2 파일 할당 및 단편화

/data 파티션의 data 디렉터리 하위에는 많은 디렉터리들이 존재하며 내부에 안드로이드 기기에 설치되어 있는 애플리케이션 데이터들이 존재한다. 각 디렉터리들의 이름은 애플리케이션의 패키지명과 같기 때문에 디렉터리의 이름을 통해 특정 애플리케이션에 대한 데이터를 확인할 수 있다. 디렉터리 하위에는 해당 애플리케이션의 사용 내역인 사용자의 데이터가 들어있는 SQLite 데이터베이스 파일과 사용자가 입력한 계정 및 설정정보들이 들어있는 XML 파일 등이 존재한다. 이러한 각각의 파일들은 애플리케이션이 설치될 때 같이 생성되므로 파일 시스템 내부의 같은 블록 그룹 내에 존재할 가능성이 크다.

〔그림 5〕는 EnCase7〔17〕를 사용하여 데이터 파티션의 data 디렉터리 바로 하위에 해당하는 디렉터리들의 정보(이름, 경로, inode번호, 섹터 번호, 크기)를 확인한 것이다. 'File Identifier' 항목이 inode번호이며, 'Physical Sector' 항목은 해당 데이터의 물리적 위치다. 디렉터리들은 모두 1번 블록 그룹에 존재한다는 것을 알 수 있고 이를 통해 같은 디렉터리 내의 데이터들은 같은 블록 그룹 안에 할당하는 Ext4 파일 시스템의 데이터 할당 정책을 확인할 수 있다.

〔그림 6〕은 문자 애플리케이션에 대한 데이터들의 정보를 확인한 것이다. 애플리케이션의 패키지명은 'com.sec.mms'로 〔그림 5〕에서 확인한 것과 같이 다른 애플리케이션의 디렉터리들과 마찬가지로 1번 블록 그룹에 존재하며 하위의 디렉터리들도 같은 블록

〔표 4〕 안드로이드 Ext4 파일시스템 내부의 파일 크기 및 단편화 여부

분 류	크 기(%)		단편화 여부(%)			
	1개 블록 이하	2개 블록 이상	1조각	2조각	3조각	4조각 이상
XML	97.48	2.52	100	-	-	-
SQLite database	1.56	98.44	8.04	64.06	15.40	12.5
BMP	18.75	81.25	100	-	-	-
GIF	74.42	25.58	100	-	-	-
JPG	2.30	97.70	100	-	-	-
PNG	46.00	54.00	98.97	1.03	-	-

Name	Item Path	File Identifier	Physical Sector	Logical Size
com.sec.android.app.smtsetup	GS2_data\/data\data\com.sec.android.app.smtsetup	8369	270,336	4,096
com.samsung.SMT	GS2_data\/data\data\com.samsung.SMT	8371	270,352	4,096
com.skt.skaf.CID00SID02	GS2_data\/data\data\com.skt.skaf.CID00SID02	8373	270,368	4,096
com.sec.samsung.skt.hidden	GS2_data\/data\data\com.sec.samsung.skt.hidden	8375	270,384	4,096
com.sec.android.sktadminsetting	GS2_data\/data\data\com.sec.android.sktadminsetting	8377	270,400	4,096
com.sec.android.skaflauncher	GS2_data\/data\data\com.sec.android.skaflauncher	8379	270,416	4,096
com.samsung.usim.contacts	GS2_data\/data\data\com.samsung.usim.contacts	8381	270,432	4,096
com.sec.mms	GS2_data\/data\data\com.sec.mms	8383	270,448	4,096
com.sec.readershub	GS2_data\/data\data\com.sec.readershub	8385	270,464	4,096
com.skt.RInstallAgent	GS2_data\/data\data\com.skt.RInstallAgent	8387	270,480	4,096

〔그림 5〕 /data/data 디렉터리 하위 데이터 정보

Name	Item Path	File Identifier	Physical Sector	Logical Size
lib	GS2_data\/data\data\com.sec.mms\lib	8384	270,456	4,096
shared_prefs	GS2_data\/data\data\com.sec.mms\shared_prefs	8652	272,648	4,096
mmsdata	GS2_data\/data\data\com.sec.mms\mmsdata	8657	272,496	4,096
attach	GS2_data\/data\data\com.sec.mms\attach	8666	272,568	4,096
deco	GS2_data\/data\data\com.sec.mms\deco	8671	272,608	4,096
databases	GS2_data\/data\data\com.sec.mms\databases	8675	272,640	4,096

〔그림 6〕 /data/data/com.sec.mms 디렉터리 하위 데이터 정보

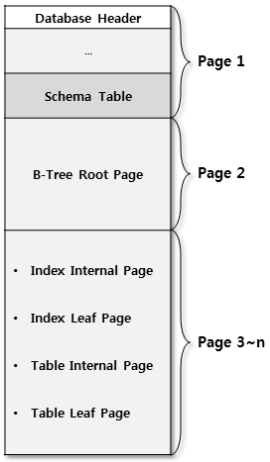
그룹에 있는 것을 알 수 있다. 이와 같이 directory entry나 애플리케이션에 대한 사용자의 설정 정보가 들어 있는 XML 파일처럼 크기가 작은 데이터는 (4KB) 일반적으로 같은 블록 그룹 내에 연속적으로 할당된다. 하지만 데이터의 크기가 큰 경우에는 같은 블록 그룹 내에 할당되지 못하는 경우도 있으며 할당되었더라도 새로운 데이터가 추가되면서 다른 블록 그룹으로 조각나 할당될 수 있다. 이러한 현상은대표적으로 SQLite 데이터베이스 파일에서 많이 발생하며 안드로이드 애플리케이션은 사용자 데이터를 SQLite 데이터베이스 파일을 사용하여 저장하기 때문에 이 파일에 대한 파일 시스템 내부에서의 할당 및 단편화 특징을 분석할 필요가 있다.

5.2.1 XML 파일

XML 파일은 각 애플리케이션에 대한 설정 정보들이 저장되어 있는데 그 중에는 사용자가 입력한 이름, 계정정보, 가장 최근에 동기화된 시간 정보 등도 포함될 수 있기 때문에 디지털 포렌식 관점에서 꼭 분석해야 할 데이터 중 하나이다. 〔표 4〕에서 확인했듯이, XML 파일은 크기가 작기 때문에 대부분 1개의 블록에 할당되어 저장되며 크기가 증가하더라도 Ext4 파일 시스템의 단편화 회피 정책에 따라 일반적으로 단편화가 발생하지 않다.

5.2.2 SQLite 데이터베이스 파일

SQLite 데이터베이스 파일은 페이지라는 최소 저장 단위를 사용하며 〔그림 7〕과 같은 구조를 갖는다



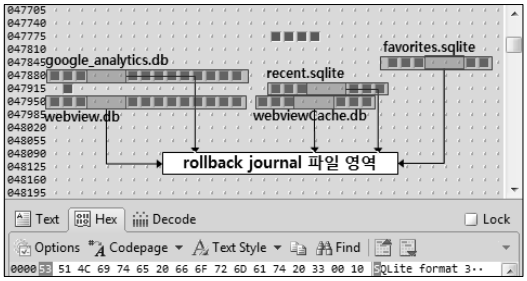
〔그림 7〕 SQLite 데이터베이스 구조

〔18〕.

첫 번째 페이지에 존재하는 데이터베이스 헤더의 16~17 byte, 28~31 byte offset에 페이지 크기와 전체 페이지 개수가 big-endian 형식으로 저장되며, 스키마 테이블 영역에는 데이터베이스의 스키마 정보가 저장되기 때문에 첫 번째 페이지로부터 데이터베이스의 전체적인 정보를 확인할 수 있다.

두 번째 페이지부터는 B-tree로 구성되며 두 번째 페이지에는 B-tree의 루트 페이지가 존재하며 세 번째 페이지부터는 B-tree의 포인터가 저장되어 있는 Index 페이지와 데이터의 내용이 저장되어 있는 Table 페이지가 존재하고 각 Index, Table 페이지는 하위 페이지의 포인터가 저장된 Internal 페이지와 실제 데이터가 저장된 Leaf 페이지를 갖는다. Table Leaf 페이지는 B-tree의 가장 하부에 존재하며 해당 영역에 데이터베이스의 레코드 데이터가 저장된다〔18〕. 따라서 SQLite 데이터베이스 파일의 식별 및 실제 레코드 정보를 획득하기 위해 첫 번째 페이지와 Table Leaf 페이지가 중요하다.

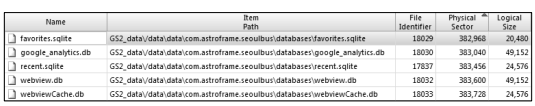
SQLite 데이터베이스 파일은 기본적으로 저널 기능을 사용하는데, 데이터베이스의 트랜잭션을 실행하기 전에 '데이터베이스 파일 이름-journal' 파일명으로 rollback journal 파일을 생성하고 변경되기 전의 데이터베이스의 내부 내용을 저장한다. rollback journal 파일은 트랜잭션이 완료되면 삭제되며 해당 사항은 다음과 같은 저널 모드를 통해 변경할 수 있다. 'PRAGMA journal_mode = DELETE | TRUNCATE | PERSIST | MEMORY | WAL | OFF' 저널 모드는 기본적으로 'DELETE'로 설정



〔그림 8〕 SQLite 데이터베이스 파일 단편화 정보

되어 있으며 'OFF'로 설정할 경우 저널 기능을 사용하지 않는다〔18〕. 저널모드가 'WAL'로 설정되어 있을 경우 트랜잭션 컨트롤 매커니즘인 WAL(The Write-Ahead Log)을 사용하는데, 이때 rollback journal 파일과 마찬가지로 '데이터베이스 파일 이름-wal' 파일과 wal-index 파일인 '데이터베이스 파일 이름-shm' 파일이 생성되고 트랜잭션이 완료되면 삭제된다〔18〕〔19〕.

〔그림 8〕은 EnCase7의 'Disk View' 기능을 사용하여 파일 시스템에서 단편화된 SQLite 데이터베이스 파일을 확인한 것으로, 파일의 최초 3개 블록 뒤에 rollback journal 파일 때문에 생성된 3개 블록 만큼의 비할당 영역이 존재하고 그 이후에 데이터들이 저장된 블록들이 존재하는 것을 확인할 수 있으며 저널 모드 설정에 따라 생성되는 wal, shm 파일도 rollback journal 파일과 마찬가지로 SQLite 데이터베이스 파일의 중간에 존재할 수 있다. 이러한 트랜잭션 도중에 생성 및 삭제되는 저널 파일들의 특성 때문에 SQLite 데이터베이스 파일의 크기와 상관없이 단편화가 발생하며 사용자가 자주 사용하는 애플리케이션이 사용하는 SQLite 데이터베이스 파일은 더욱 많은 단편화가 발생될 수 있다. 따라서 이러한 저널 파일들은 파일 카빙 과정에서 제외해야할 대상이다.



〔그림 9〕 SQLite 데이터베이스 파일 메타데이터 정보

〔그림 9〕는 EnCase7을 사용하여 〔그림 8〕에 해당하는 각 데이터베이스 파일들의 메타데이터를 확인한 것이고, 〔표 5〕는 〔그림 8〕의 데이터베이스 파일들의 단편화 정보를 정리한 것이다. 〔그림 9〕와 〔표 5〕를 통해서 각 5개의 SQLite 데이터베이스 파일은 같은

〔표 5〕 SQLite 데이터베이스 파일 단편화 정보

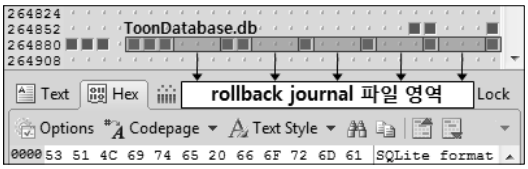
파일명	조각 번호	섹터 번호	블록 번호	블록 그룹	블록 개수
favorites.sqlite	1	382,968	47,871	1	3
	2	383,016	47,877	1	2
google_analytics.db	1	383,040	47,880	1	3
	2	383,088	47,886	1	9
recent.sqlite	1	383,456	47,932	1	3
	2	383,504	47,938	1	3
webview.db	1	383,600	47,950	1	3
	2	383,648	47,956	1	9
webviewCache.db	1	383,728	47,966	1	3
	2	383,776	47,972	1	3

경로('/data/data/com.astroframe.seoulbus/databases/')에 있는 파일이고, 같은 블록 그룹에 존재한다는 것을 알 수 있다. 이러한 현상은 Ext4 파일 시스템의 파일 할당 정책 때문이다. 이 파일들의 첫 번째 페이지에는 SQLite 데이터베이스의

스키마 정보를 비롯한 기본적인 메타데이터가 저장되고, 두 번째 페이지에는 B-Tree 루트 페이지가 존재하며 세 번째 페이지에는 B-Tree 루트 페이지에 해당하는 레코드가 저장된 Table Leaf 페이지가 존재한다. 그 후 추가되는 데이터들을 처리하기 위한 트랜잭션을 위해 3개 블록을 할당하여 저널 파일이 생성되고 SQLite 데이터베이스의 2번째 조각이 저널 파일 뒤로 연속적으로 할당되었다. 대부분의 SQLite 데이터베이스 파일들은 이러한 방식으로 단편화가 발생되며 첫 번째 페이지로부터 데이터베이스의 전체 크기 및 페이지 크기를 확인한 후 파일 중간에 존재하는 저널 파일을 식별 및 복구 대상에서 제외함으로서 2조각으로 단편화된 SQLite 데이터베이스를 온전히 복구할 수 있으며 이러한 프로세스를 통해 〔표 4〕에서 확인한 약 72.1%의 SQLite 데이터베이스 파일을 복구할 수 있다.

많은 트랜잭션이 발생된 SQLite 데이터베이스 파일의 경우 3조각 이상으로 단편화가 발생될 확률이 높으며 이러한 경우에도 위와 같은 프로세스를 적용할 수 있다. 〔그림 10〕은 6개의 조각으로 단편화가 발생된 'ToonDatabase.db'파일의 단편화 정보를 확인한 것으로 파일의 중간에 존재하는 저널 파일만 파일 복구 대상에서 제외함으로써 SQLite 데이터베이스 파일을 온전히 복구할 수 있다. 이와 같은 프로세스를 통해 3조각 이상으로 조각난 약 27.9%의 파일 중에

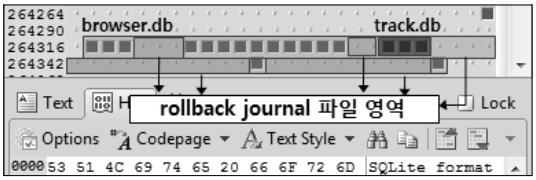
서 약 35.29%의 파일이 복구 가능하며 전체적으로는 약 81.94%의 SQLite 데이터베이스 파일의 복구가 가능하다.



〔그림 10〕 'ToonDatabase.db' 파일 단편화 정보

나머지 약 18.06%의 SQLite 데이터베이스 파일은 〔그림 11〕과 같이 3조각 이상으로 단편화된 SQLite 데이터베이스 파일 중 단편화된 조각 중간에 다른 파일이 존재하는 경우와 조각난 데이터가 같은 블록 그룹 안에 할당되지 못하는 경우다.

이러한 경우 해당 SQLite 데이터베이스 파일의 첫 번째 페이지에 존재하는 테이블 스키마 정보와 Table Leaf 페이지의 테이블 스키마 정보를 비교함으로써 같은 데이터베이스 파일의 레코드인지 확인할 수 있으며 다른 데이터베이스의 레코드인 경우 복구 대상에서 제외함으로써 오탐을 줄일 수 있다. 같은 데이터베이스 스키마를 갖는 레코드가 발견될 경우 해당 레코드를 추출하는 레코드 복구를 통해 삭제된 내용을 확인할 수 있다. SQLite 데이터베이스 파일 카빙에 대한 프로세스는 6.2에서 자세히 설명한다.



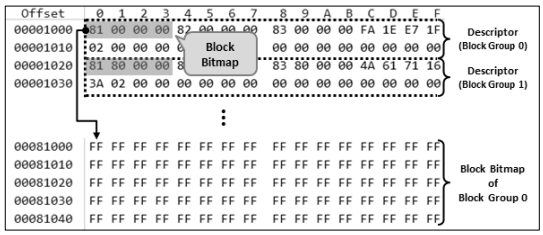
〔그림 11〕 'browser.db' 파일의 단편화 정보

VI. 파일 카빙 방법

Ext4 파일 시스템에서는 파일 삭제 시 복구에 필요한 정보가 초기화되기 때문에 삭제된 파일을 복구하기 위해서는 비할당 영역에서 파일 포맷별 카빙을 수행해야 한다. 따라서 먼저 블록 그룹별로 비할당 영역들을 수집한 뒤 블록 단위로 탐색하면서 파일 시스템의 특성과 파일 포맷별 고유한 특성을 이용하여 연속적인 파일 및 단편화가 발생된 파일에 대한 복구를 수행해야 한다.

6.1 비할당 영역 수집

비할당 영역은 할당되지 않은 블록들의 집합을 말하며 이 영역을 추출하기 위해서는 먼저 GDT 에서 총 블록 그룹의 개수와 각 블록 그룹당 block bitmap의 위치를 확인해야 한다. 〔그림 12〕는 GDT 영역에서 각 블록 그룹들에 대한 메타 정보가 저장되어 있는 descriptor 내부의 block bitmap 정보 저장 위치를 확인한 것이다.

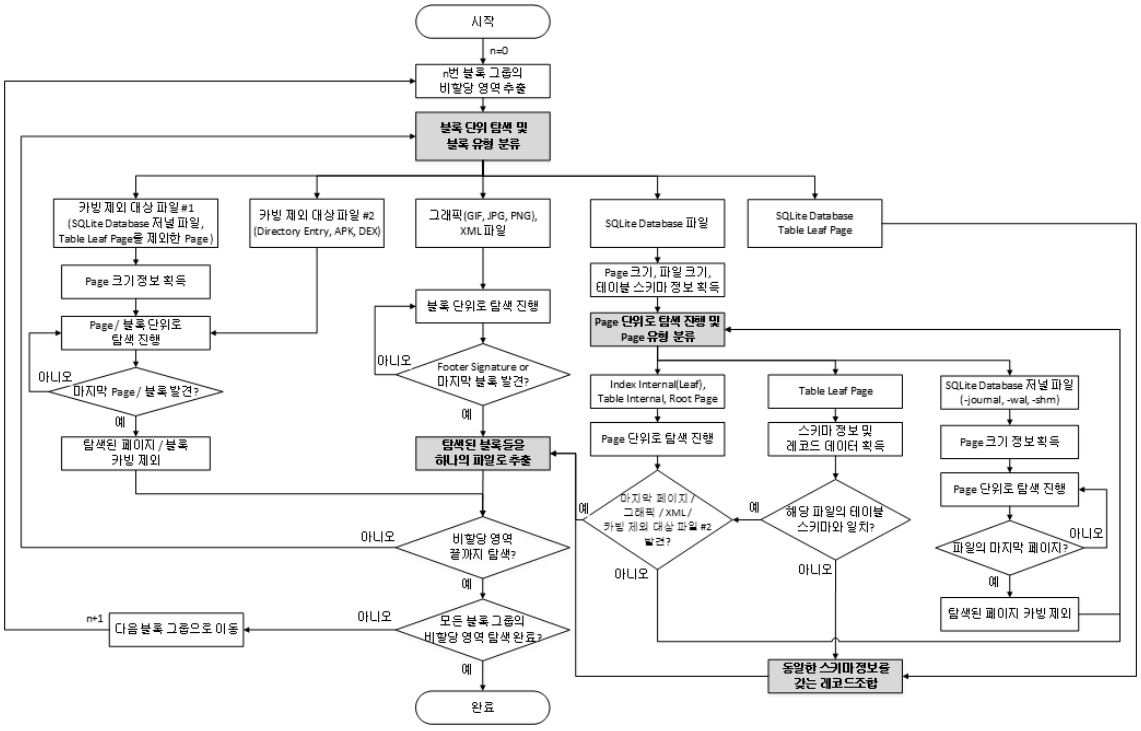


〔그림 12〕 각 블록 그룹들의 block bitmap 정보

각 블록 그룹에 존재하는 block bitmap의 위치로 차례로 이동하여 bitmap을 해석함으로써 비할당 블록을 파악한다. 모든 블록 그룹의 block bitmap을 해석하면 파일 시스템 전체의 비할당 영역을 수집할 수 있다. Ext4 파일 시스템은 5.2에서 확인했듯이 특정 애플리케이션과 관련 있는 파일들은 같은 블록 그룹에 존재할 가능성이 높고, 파일 단편화가 다른 블록 그룹으로 발생하더라도 최대한 인근의 블록 그룹으로 파일 조각을 할당하기 때문에 단편화된 파일 조각의 재조합 과정에서의 복구율을 높이기 위해서 비할당 영역을 각 블록 그룹별로 따로 수집해야 한다.

6.2 파일 카빙 프로세스

일반적인 리눅스 시스템에서 사용된 Ext4 파일 시스템에 대한 카빙을 수행할 경우는 다양한 포맷들을 고려해야 하지만 안드로이드폰에서 사용된 Ext4 파일 시스템의 경우는 〔표 3〕에 언급한 것처럼 다소 제한적인 포맷만이 존재한다. 이 파일들 중 디지털 포렌식 관점에서 중점적으로 카빙 대상이 되는 파일은 사용자 데이터가 저장되어 있는 SQLite 데이터베이스 파일과 단편화된 SQLite 데이터베이스의 실제 레코드가 존재하는 Table Leaf 페이지, 사용자가 입력한 계정 및 패스워드 등의 설정 정보들이 저장되어 있는 XML 파일이며 추가적으로 사용자가 애플리케이션을 사용하며 생성된 BMP, GIF, JPG, PNG 등의 그



〔그림 13〕 파일 카빙 프로세스

래픽 관련 파일에 대해서도 카빙을 수행할 필요가 있다. SQLite 데이터베이스 파일을 제외한 파일들은 일반적으로 크기가 작고 내용 수정 시 증가되는 데이터의 양도 작기 때문에 대부분 연속적으로 존재한다. 전체적인 파일 카빙 프로세스는 〔그림 13〕과 같다.

카빙 제외 대상 파일은 SQLite 데이터베이스와 관계된 파일(저널 파일, Table Leaf 페이지를 제외한 페이지)들과 파일 시스템의 메타데이터인 directory entry, 애플리케이션 실행과 관련된 APK, DEX 파일들이며 〔표 6〕, 〔표 7〕과 같은 시그니처로 분류할 수 있다. SQLite 데이터베이스의 저널 파일들은 SQLite 데이터베이스의 단편화를 유발하는 파일이므로 카빙 대상에서 제외하며, 파일 시스템의 블록 단위로 탐색 시 발견되는 B-Tree Root, Index Internal, Index Leaf, Table Internal 페이지들은 어떤 SQLite 데이터베이스 파일에 해당하는 메타데이터인지 구분하기 어렵기 때문에 카빙에서 제외한다. 이러한 '카빙 제외 대상 파일 #1'에 해당하는 파일들은 다른 파일의 시그니처가 발견되기 전까지 해당 파일의 페이지 크기 만큼 탐색을 진행함으로써 파일의 끝을 구별한다.

〔표 6〕 카빙 제외 대상 파일 #1

항목	시그니처			페이지 크기		
	오프셋	크기	값	오프셋	크기	값
rollabck journal	0	8	0xD9 D5 05 F9 20 A1 63 D7	24	4	가변적
WAL(Write-Ahead Log)	0	4	0x37 7F 06 82	8	4	가변적
shm(WAL-index)	0	3	0x18 E2 2D	-	-	-
B-Tree Root Page	0	1	0x01	-	-	-
Index Internal Page	0	1	0x02	-	-	-
Index Leaf Page	0	1	0x0A	-	-	-
Table Internal Page	0	1	0x05	-	-	-

〔표 7〕 카빙 제외 대상 파일 #2

항목	오프셋	크기	시그니처
directory entry	4	20	0x0C 00 01 02 2E 00 00 00 02 00 00 00
APK	0	4	0x50 4B 03 04
DEX	0	8	0x64 65 79 0A 30 33 35 00

'카빙 제외 대상 파일 #2'에 해당하는 파일들의 끝은 4.1에서 언급한 파일의 끝을 구분할 수 있는 특성을 이용하는 기존 카빙과 동일한 방법을 사용한다〔20〕〔21〕.

카빙 대상인 그래픽 파일과 XML 파일의 경우 〔표 8〕과 같은 각 파일의 헤더 시그니처를 이용하여 파일의 종류를 판단하고, 파일의 끝은 푸터 시그니처를 이용하거나 푸터 시그니처가 없는 포맷은 파일의 끝을 구분할 수 있는 특성을 이용한다〔20〕〔21〕.

SQLite 데이터베이스 파일의 경우 페이지 크기와 파일 크기, 테이블 스키마 정보를 획득한 후 페이지 단위로 탐색을 진행하며 이때 발견되는 각 페이지들은 해당 파일을 구성하는 페이지로 판단하여 해당 SQLite 데이터베이스 파일에 포함한다. 탐색을 진행하며 발견되는 SQLite 데이터베이스의 저널 파일들은 카빙 대상에서 제외하며, 발견된 Table Leaf 페이지의 레코드의 스키마 정보가 해당 데이터베이스 파일의 스키마 정보와 일치하지 않는 경우와 그 외의 파일의 시그니처가 발견되는 경우는 온전한 카빙이 불가능한 경우이므로 탐색을 중지하고 다시 파일 시스템의 블록 단위 탐색을 수행한다. 카빙이 정상적으로 완료되지 않은 SQLite 데이터베이스 파일은 레코드 재조합을 위해 스키마 정보에 따라 따로 분류한다.

블록 단위 탐색 도중 발견된 Table Leaf 페이지의 경우, 스키마 정보를 바탕으로 정상적으로 완료되지 않은 SQLite 데이터베이스 파일과 레코드 재조합을 수행할 수 있다. 이 때, 특정 컬럼에 해당하는 데이터가 시간 정보로 판단되는 경우(안드로이드 애플리케이션은 시간 정보를 주로 UnixTime 포맷을 사용한다.) 이 정보를 기반으로 레코드의 순서를 조합할 수 있다. 이러한 프로세스를 통해 정상적으로 카빙이 완료되지 않은 SQLite 데이터베이스 파일에 대해 레코드 단위의 카빙을 수행할 수 있다.

제안한 프로세스에 따라서 모든 블록 그룹의 비할당 영역에 대해서 카빙을 수행하면 카빙을 완료한다. 카빙된 파일들 중에서도 단편화가 많이 발생된 SQLite 데이터베이스 파일의 경우 정상적으로 카빙이 되지 않을 수도 있다. 따라서 카빙 수행 후 남은 블록 조각들을 안드로이드 기기에서 사용된 플래쉬 메모리의 페이지 단위로 데이터의 포맷을 분류한 후〔22〕, SQLite 데이터베이스의 페이지들을 재조합 하는 등의 분석도 필요 하다. 5.2에서 살펴본 것과 같이 조각난 블록들은 해당 블록 그룹에 존재할 가능성이 높기 때문에 블록 그룹별로 SQLite 데이터베이스의 페이지들을 수집한 뒤, 조각난 Table Leaf 페이지들로부터 추가적인 재조합을 수행한다. 또한, 카빙된 많은 XML, SQLite 데이터베이스 파일들을 효과적으로 분석하기 위해 파일 내부 문자열 추출을 수행하면 디지털 포렌식 분석에 더 많은 도움이 될 것이다〔23〕.

VII. 결론 및 향후 계획

안드로이드 스마트폰, 태블릿 PC등을 비롯한 안드로이드 운영체제를 기반으로 하는 여러 임베디드 기기들이 증가하고 있다. 이러한 임베디드 기기 내부에는 사용자의 일상생활과 관련된 많은 데이터들이 저장되어 있기 때문에 디지털 포렌식 조사 과정에서 매우 중요한 분석 대상이 되고 있다. 안드로이드 운영체제를 사용하는 임베디드 기기들의 플래쉬 메모리 내부의 파일 시스템으로 Ext4 파일 시스템이 널리 사용되고 있

〔표 8〕 카빙 대상 파일 시그니처

항목	헤더 시그니처			푸터 시그니처
	바이트	크기	값	
XML	0	5	0x3C 3F 78 6D 6C	-
SQLite database	0	16	0x53 51 4C 69 74 65 20 66 6F 72 6D 61 74 20 33 00	-
Table Leaf Page	0	1	0x0D	-
BMP	0	2	0x42 4D	-
GIF	0	6	0x47 49 46 38 37(39) 61	0x00 3B
JPG	0	10	0xFF D8 FF E0 ?? ?? 4A 46 49 46	0xFF D9
PNG	0	8	0x89 50 4E 47 0D 0A 1A 0A	0x49 45 4E 44 4E 42 60 82

기 때문에 디지털 포렌식 분석 관점에서 이에 대한 연구가 진행되어왔다. 하지만 Ext4 파일 시스템 특성상 사용자가 의도적으로 특정 애플리케이션 및 데이터를 삭제할 경우 삭제된 데이터에 대한 복구와 분석이 어려운 실정이며 이에 대한 방안 및 연구가 부족한 상황이다.

본 논문은 Ext4 파일 시스템을 분석함으로써 파일 할당 정책과 안드로이드 내부의 사용자 데이터가 존재하는 파티션, 애플리케이션 설치 시 생성되는 파일들의 종류 및 특성 등을 이용하여 애플리케이션 등의 데이터가 삭제된 경우 관련된 파일들을 효과적으로 복구 및 분석할 수 있는 방법을 제안하였다. 특히 제안한 방법을 통해 많은 사용자 데이터가 존재하는 XML파일과 SQLite 데이터베이스 파일을 등을 효과적으로 복구할 수 있도록 하였다. 이러한 연구 결과를 토대로 도구를 개발하고 많은 테스트를 통해 알고리즘을 수정해 나간다면 Ext4 파일 시스템을 사용하는 많은 임베디드 기기에 대한 디지털 포렌식 분석에 많은 도움을 줄 수 있을 것이다.

참고문헌

〔1〕 Kevin D. Fairbanks, "An analysis of Ext4 for digital forensics," Digital Investigation, Vol. 9, pp. 118-130, Aug. 2012.

〔2〕 Dohyun Kim, Jungheum Park, Keun-gi Lee, and Sangjin Lee, "Forensic Analysis of Android Phone using Ext4 File System Journal Log," Lecture Notes in Electrical Engineering, Vol. 164, pp. 435-446, June. 2012.

〔3〕 Val Henson, Zach Brown, Theodore Ts'o, and Arjan van de Ven, "Reducing fsck time for ext2 file systems," Proceeding of the Linux Symposium, Vol. 1, July. 2006.

〔4〕 Philip Craiger, "Recovering Digital Evidence from Linux Systems," IFIP The International Federation for Information Processing, Vol. 194, pp. 233-244, Feb. 2005.

〔5〕 SANS Information, Network, Computer Security Training, Research, Resources, http://www.sans.org.

〔6〕 Hal Pomeranz, "EXT3 File Recovery via Indirect Blocks," http://computer-forensics.sans.org/summit-archives/2011/EXT3-file-recovery.pdf.

〔7〕 Gregorio Narváez, "Taking advantage of Ext3 journaling file system in a forensic investigation," SANS Institute Reading Room, Dec. 2007.

〔8〕 ext3grep, http://code.google.com/p/ext3grep.

〔9〕 extcarve, http://freecode.com/projects/extcarve.

〔10〕 giis-ext4, http://www.giis.co.in.

〔11〕 Stellar Phoenix Linux Data Recovery, http://www.stellarinfo.com/linux-data-recovery.htm.

〔12〕 EaseUS, http://www.easeus.com/datarecoverywizard.

〔13〕 Brian Carrier, File Sysetm Forensic Analysis, Addison Wesley Professional, 2005.

〔14〕 Aneesh Kumar K.V, Mingming Cao, and Jose R Santos, "Ext4 block and inode allocator improvements," Proceeding of the Linux Symposium, July. 2008.

〔15〕 Avantika Mathur, Mingming Cao, and Suparna Bhattacharya, "The new ext4 filesystem: current status and future plans," Proceedings of the Linux Symposium, June. 2007.

〔16〕 Theodore, "Speeding up file system checks in ext4," The Linux Foundation, 2009.

〔17〕 Guidance Software, http://www.guidancesoftware.com.

〔18〕 SQLite Database File Format, http://www.sqlite.org/fileformat2.html.

〔19〕 SQLite Database File Format2, http://www.evolane.com/support/manuals/shared/manuals/tcltk/sqlite/fileformat.html.

〔20〕 Simson L. Garfinkel, "GarCarving contiguous and fragmented files with fast object validation," Digital Investigation, Vol. 4, pp. S2-S12, Sept. 2007.

〔21〕 Golden Richard III, Vassil Roussev and Lodovico Marziale, "In-Place File Carving," Digital Forensics III : IFIP The International Federation for Information Processing, Vol. 242, pp. 217-230, Jan. 2007.
〔22〕 Jungheum Park, Hyunji Chung, and Sangjin Lee, "Forensic analysis techniques for fragmented flash memory pages in smartphones," Digital Investigation, Vol 9, pp. 109-118, Nov. 2012.
〔23〕 Sangjun Jeon, Jungheum Park, Keun-gi Lee, and Sangjin Lee, "An Efficient Method of Extracting Strings from Unfixed-Form Data," Lecture Notes in Electrical Engineering, Vol.164, pp.425-434, June. 2012.

〈저자소개〉

김 도 현 (Dohyun Kim) 학생회원
2011년 2월: 서울과학기술대학교 정보통신대학 컴퓨터공학 공학사
2011년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 디지털 포렌식, 모바일 포렌식, 파일 시스템

박 정 흠 (Jungheum Park) 학생회원
2007년 2월: 한양대학교 정보통신대학 컴퓨터전공 공학사
2007년 3월~2009년 2월: 고려대학교 정보경영공학전문대학원 공학석사
2009년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 디지털 포렌식, 안티-안티 포렌식

이 상 진 (Sangjin Lee) 종신회원
1987년 2월: 고려대학교 수학과 학사
1989년 2월: 고려대학교 수학과 석사
1994년 8월: 고려대학교 수학과 박사
1989년 10월~1999년 2월: ETRI 선임 연구원
1999년 3월~2001년 8월: 고려대학교 자연과학대학 조교수
2001년 9월~현재: 고려대학교 정보보호대학원 교수
2008년 3월~현재: 고려대학교 디지털포렌식연구센터 센터장
〈관심분야〉 디지털 포렌식, 심층 암호, 해쉬 함수