석사학위 논문

Journaling of Journal 기반 SQLite 파일 복구 기법

Journaling of Journal based SQLite file recovery

2018년 12월

숭실대학교 대학원

컴퓨터학과

김찬규

석사학위 논문

Journaling of Journal 기반 SQLite 파일 복구 기법

Journaling of Journal based SQLite file recovery

2018년 12월

숭실대학교 대학원

컴퓨터학과

김 찬 규

석사학위 논문

Journaling of Journal 기반 SQLite 파일 복구 기법

지도교수 박 동 주

이 논문을 석사학위 논문으로 제출함

2018년 12월

숭실대학교 대학원

컴퓨터학과

김 찬 규

김 찬 규의 석사학위 논문을 인준함

심 사 위 원 장 황 규 백 인

심 사 위 원 이 정 진 인

심 사 위 원 박 동 주 인

2018년 12월

숭실대학교 대학원

목 차

표 목 차

그 림 목 차

국문초록

Journaling of Journal 기반 SQLite 파일 복구 기법

김찬규

컴퓨터학과

숭실대학교 대학원

디지털 포렌식에서 안드로이드 어플리케이션의 사용자 정보는 중요한 디지털 증거다. 안드로이드는 어플리케이션의 사용자 정보를 SQLite 파일로 저장한다. 또한 SQLite는 중요 데이터를 저널 파일이란 별개의 파일에 백업한다. 기존 기법은 안드로이드 상에서 SQLite 파일 트랜잭션이 가지는 Journaling of Journal 이상을 기반으로 단서를 확보하지 못했다. 본 논문은 SQLite 파일 복구를 효과적으로 수행하기 위한 단서를 확보하는 기법을 제안한다. 안드로이드 환경에서 발생하는 Journaling of Journal 이상은 SQLite 파일의 메타데이터를 백업해 ExT4의 저널 영역에 저장한다. 해당 메타데이터는 SQLite 파일 복구에 필요한 단서를 제공한다. 해당 단서를 활용해서 파일 카빙을 더 효율적으로 진행할 수 있다.

ABSTRACT

Journaling of Journal based SQLite file recovery

KIM, CHAN-GYU
Department of Computer
Graduate School of Soongsil University

Android application's user information is important digital evidence to digital forensic. Android store user information in SQLite file. Also, SQLite backup important data to a separate file from the journal file. Existing Technic can't get clue from the Journaling of Journal that occured by SQLite file transaction. In this paper, we propose a technique to clues about effectively perform SQLite file recovery. Journaling of Journal anomaly, that occurred in Android platform, duplicate SQLite file's metadata and store it on ExT4 journal block. That metadata give to clue about SQLite file recovery. Using that clue makes file recovery task more efficient.

제 1 장 서 론

1.1 연구 배경 및 동기

2014년 이후 사이버 범죄 발생 건수가 매년 100,000건을 넘어가고 있다. 그러나 매년 20000건을 넘는 사이버 범죄가 검거되지 못하면서 사이버 범죄 검거율에 대한 우려가 높아지고 있다. [표 1]은 경찰청 사이버 안전국에서 발표한 각종 사이버 범죄 발생 건수에 대한 통계자료다.

표 1-1 경찰청 사이버 안전국 사이버범죄 통계자료

구분	총계		정보통신망침해 범죄		정보 통신망 이용 범죄		불법컨텐츠범죄	
	발생 건수	검거 건수	발생 건수	검거 건수	발생건 수	검거건 수	발생건 수	검거건 수
2014	110109	71950	2291	846	89519	56451	18299	14643
2015	144679	104888	3154	847	118362	86658	23163	17388
2016	153075	127758	2770	842	121867	103172	28438	23539
2017	131734	107489	3156	1047	107721	88779	21307	17312

사이버 범죄가 늘어나면서 해당 범죄의 증거를 확보하는 기법인 디지털 포렌식에 대한 관심이 늘어나고 있다. 사이버 범죄는 디지털 컨텐츠를 이용해서 일어나는 범죄를 통칭하며, 이러한 사이버 범죄를 해결하기 위해서는 디지털 포렌식을 통해 얻은 디지털 증거가 필요하다. 사이버 범죄는 특히 안드로이드 플랫폼 기기에서 크게 증가하고 있다. [그림 1-1]은 Nokia 2017 Threat Intelligence Report에서 발췌한 플랫폼에 따라 나눈 사이버 범죄의 발생 비율로, 안드로이드가 전체 플랫폼 중에서 68퍼센트의 사이버 범죄의 대상이 되었다고 제시되었다. 이는 많은 사람들이 안드로이드 플랫폼 기기를 이용하고, 생활에 밀접한 연관을 가지고 있기 때문이라고 언급되었다.

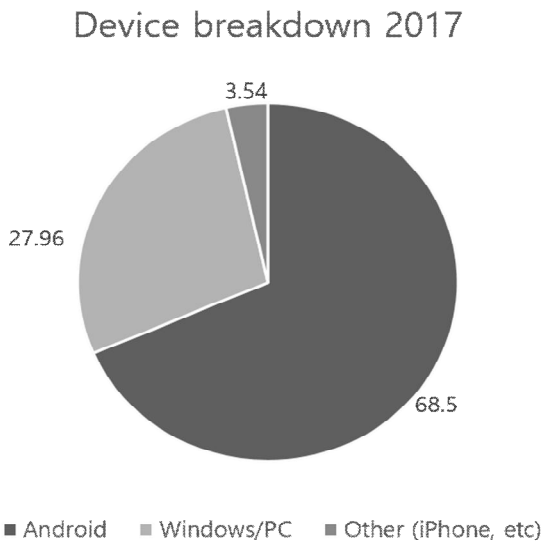

그림 1-1 Nokia 2017 Threat Intelligence Report

디지털 포렌식에서 중요한 것은 증거로써 효력이 있는 데이터를 복구하는 것이다. 이러한 데이터 확보를 위한 방법 중 하나가 전수조사다. 전

수 조사는 대상 영역, 집합 전체를 하나하나 전부 조사하는 조사방법이다. 그러나 전 영역을 조사하는 방법은 디지털 포렌식의 법적인 관점에서 위법수집증거배제법칙에 따라 문제의 소지가 있다. 위법수집증거배제법칙은 위법한 방법으로 수집한 증거의 증거능력을 인정하지 않는 법칙이다. 이는 디지털 포렌식에서도 개인 정보 보호를 위해 사전에 허가된 영역만을 조사하고 나머지는 조사해서 증거를 확보해도 효력을 인정하지 않는 것으로 적용되고 있다. 따라서 디지털 포렌식에서 사이버 범죄와 관련된 정보가 위치한 곳을 특정하는 것은 중요한 작업이다.

1.2 연구 목적 및 논문 구성

안드로이드 운영체제는 2008년부터 오픈 소스 프로젝트로 배포가 시작됐다. 2018년 현재에 이르러서는 윈도우 운영체제를 넘어 가장 많이 쓰이는 운영체제가 되었다. 이러한 안드로이드 운영체제는 스마트폰뿐만 아니라 태블릿, IoT 기기 등 다양한 플랫폼에서 이용되고 있다. 전 세계적으로 다양한 사용자의 정보들이 안드로이드 플랫폼 기기에 저장되고 있다.

보안에 대한 관심이 증가하면서 데이터 복구 방지 기법들도 나날이 발전하고 있다. ExT4 파일 시스템은 데이터의 복구를 방지하기 위해 메타데이터 구조와 변경과 삭제 방식을 변경했다. 시스템이나 설정의 변경으로 인해 데이터의 관리 방식이 달라져서 기존 기법으로 데이터를 복구할 수 없는 경우도 발생하고 있다. 이에 따라 ExT4와 SQLite에 존재하는 Journaling 데이터를 통해서 데이터를 복구하는 기법이 연구되었다. 해당 기법에서는 ExT4와 SQLite의 저널을 각각 분석, 데이터 복구에 활용했다. 그러나 각각의 저널을 독립적으로 분석하는 기법은 확보할 수 있는

데이터에 한계가 존재하거나 조건이 복잡했다.

안드로이드 운영체제에서 발생하는 Journaling of Journal 이상은 Journaling 작업 과정에서 데이터가 중복되는 현상이다. 안드로이드 운영체제에서 사용하는 SQLite와 ExT4가 작업 내용을 백업하는 Journaling 작업을 실행 할 때, ExT4와 SQLite의 Journaling 작업 횟수가 증가하는 문제다. 이는 성능 저하를 유발한다는 문제가 있으나, 해당 현상의 발생을 막지 않고 해당 현상으로 발생하는 작업 수, 작업량의 감소를 목표로 한다. ExT4와 SQLite 각자의 Journaling 작업이 백업한 데이터는 작업 중에 안드로이드 운영체제에 이상이 발생했을 때, 삭제, 변경된 데이터 복구를 위해서 존재하기 때문에 해당 현상 의 발생을 막는 것이 어렵기 때문이다. 따라서, Journaling of Journal 이상으로 발생한 데이터를 데이터 복구에 활용할 수 있다.

본 논문은 안드로이드 환경에서 발생하는 Journaling of Journal 이상에서 자동으로 생성되는 데이터를 기반으로 SQLite 파일 복구를 위한 단서를 확보하는 기법을 제시한다. 2장에서는 Journaling of Journal 이상과 해당 현상이 발생하는 안드로이드 운영체제의 배경지식을 살펴보고, 3장에서는 기존의 안드로이드 운영체제의 SQLite 복구 기법에 대한 분석과 비교를 기술한다. 4장에서는 제안 기법에 대한 분석과 기존 기법과의 기능 및 성능 비교, 제안 기법만의 장점이나 기능을 기술하고, 5장에서는 구현과 실험을 통해 성능 평가 및 기능 구현 여부를 확인한다. 마지막으로 6장에서는 본 논문의 결론을 기술한다.

제 2 장 배경 지식

2.1 ExT4

ExT4는 안드로이드 운영체제에서 공식적으로 채택한 파일 시스템이다. 기존에 존재하던 ExT2, ExT3에서 개량된 ExT4 파일 시스템은 하위 호환을 지원하면서 extents, 지연된 할당 등 새로운 기능들이 더해졌다. 새로운 기능들의 추가와 함께 기존에 ExT3에서부터 지원된 저널링 파일 시스템도 지원하며, 이에 따라 ExT4는 중요 작업이 발생하면 저널링 작업을 거쳐 데이터를 백업하게 되었다.

ExT4는 기존의 파일 시스템보다 대용량의 파일을 효율적으로 관리하기 위해서 extents라는 구조체를 이용한다. 이는 기존의 ExT2, ExT3에서 사용되던 블록 매핑 방식을 개선한 방식으로, I-node 테이블에서 관리하던 블록 매핑 데이터를 extents라는 구조체로 관리하는 방식이다. 이로 인해 대용량의 파일에 대한 작업이 더 빠르게 처리될 수 있다. 그러나 이러한 새로운 데이터 구조체가 적용되면서 기존의 ExT2, ExT3에서 사용되던 복구 기법을 사용할 수 없게 되었다. 또한 extents 구조체가 파일 삭제와 동시에 초기화되기 때문에 삭제된 파일에 관련된 단서를 확보하기 어렵다.

2.1.1 ExT4의 구조

ExT4는 파일 시스템의 메타데이터를 수퍼 블록이라는 곳에 저장한다. 수퍼 블록은 ExT4 파일 시스템의 설정 값, ExT4 파일 시스템이 관리하는 파일들의 메타데이터 위치 등이 있다. ExT4는 저장소를 블록 단위로 나누고, 이 블록들을 묶어서 블록 그룹으로 관리한다. 안드로이드는 기본

적으로 블록을 4KB 단위로 설정하고, 각각의 블록 그룹은 0x8000개(32768개)의 블록을 관리한다. ExT4의 수퍼 블록은 블록 그룹 0번째에 존재하며, 수퍼 블록 이후에는 각 블록 그룹들의 메타데이터가 있는 블록 그룹 디스크립터가 존재한다. 이 블록 그룹 디스크립터 내부에는 각 블록 그룹의 I-node 테이블 위치, 블록 비트맵 테이블 위치 등이 있다. 각 디스크립터가 전부 나오고 나면 이후에는 해당 블록 그룹의 블록 비트맵과 I-node 비트맵, I-node 테이블이 존재한다. 그 뒤로는 ExT4 저널 영역이 나온다. 저널 영역 이후에는 실제 데이터가 저장되는 데이터 영역이 나오게 된다.

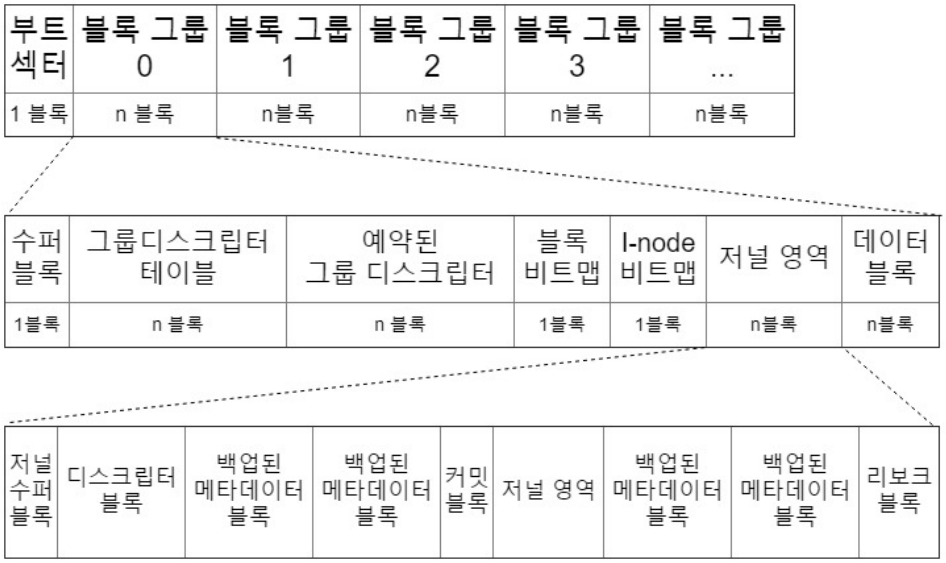

그림 2-1 ExT4의 메타데이터 구조

I-node 비트맵은 I-node 할당 여부를 알려주는 비트맵이다. I-node 비트맵을 통해 해당 파일에 어떤 I-node가 할당 되었는지 파악할 수 있다. I-node 테이블은 각각의 I-node 구조체를 관리한다. I-node 구조체는 해

당 I-node가 할당된 파일이나 디렉토리의 메타데이터가 존재한다. ExT2/3에서는 I-node 구조체에 해당 파일의 데이터 블록 위치도 기록되어있었다. 그러나 ExT4는 I-node 구조체에 데이터 블록 위치를 기록하지 않고 extents라는 구조체에 기록한다. I-node 구조체는 extents 구조체를 관리하고, extents를 통해서 데이터 블록으로 접근할 수 있다. extents는 트리 구조로 이루어져있으며, 리프 노드에 해당하는 extents가 데이터 블록의 위치 정보를 가진다.

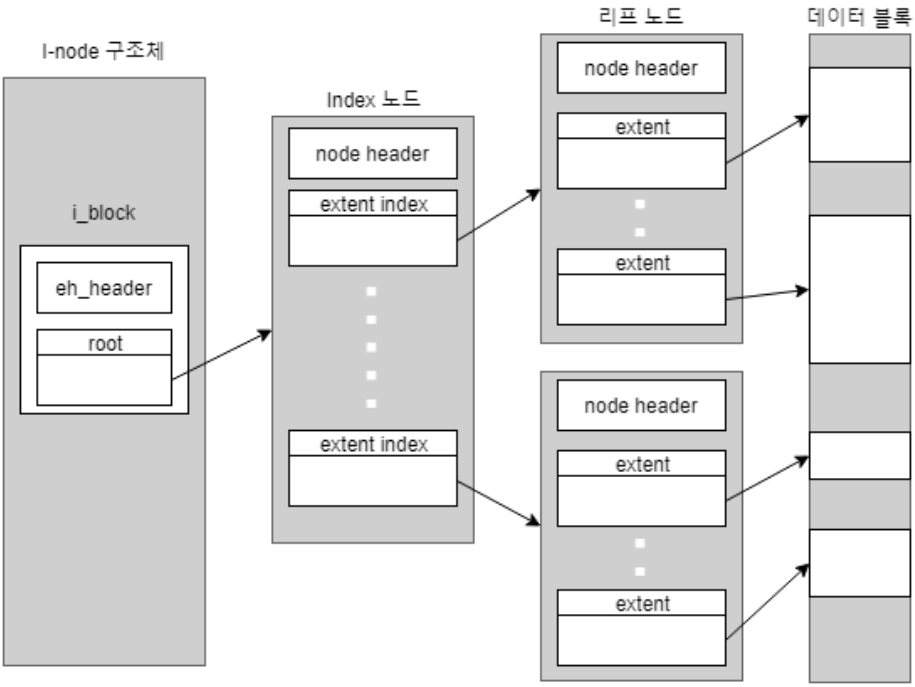

그림 2-2 Extents 트리의 구조

2.1.2 ExT4의 저널 블록

ExT4는 저널 영역에 저널링 된 데이터를 저장한다. 저널링은 중요 연산이 발생할 때 데이터를 백업하는 작업으로, 백업 되는 데이터의 종류

와 양은 ExT4의 설정에 따라 달라진다. 안드로이드 운영체제에서는 기본적으로 ordered 모드로 설정되는데, 이는 저널링 작업 때 해당하는 파일의 메타데이터만 백업하고 실제 파일 데이터는 백업하지 않는 모드다. 저널 영역 역시 블록 단위로 관리되며, ExT4 저널링의 설정과 저널 영역에 대한 메타데이터를 가진 저널 슈퍼 블록이 저널 영역 맨 앞에 위치한다. 저널 수퍼 블록은 사전에 약속된 고유한 데이터 값인 시그니처를 가지므로, 이를 통해서 저널 수퍼 블록의 위치를 파악할 수 있다. 저널 수퍼 블록 이후에는 저널 블록들이 나열되어 있는데, 이들은 각각 저널 로그로 묶일 수 있다. 저널 로그는 각각 한번의 저널링 작업을 표현한다. 저널 블록은 3종류로 나뉜다. 디스크립터 블록은 저널 로그의 시작을 가리키며, 저널 로그에 대한 정보를 가진다. 이후에는 data 블록이 나온다. data 블록은 저널링 작업으로 백업된 데이터로, 안드로이드 운영체제에서는 기본 설정에 따라 파일의 메타데이터만 백업된다. 이후에 revoke 블록이나 commit 블록이 나온다. revoke 블록은 저널링 작업을 발생시킨 중요 작업 수행 중에 문제가 생겼다는 것을 가리키며, 따라서 백업된 메타데이터를 통해서 복구 작업을 해야 할 수도 있다. commit 블록은 중요 작업 수행이 정상적으로 끝났다는 것을 의미한다. 이렇게 디스크립터 블록으로 시작해서 revoke/commit 블록으로 끝맺음한 하나의 블록 묶음이 저널 로그다.

2.2 SQLite

SQLite는 안드로이드 운영체제뿐만 아니라 iOS, 파이어폭스 등 다양한 플랫폼이 채택한 경량화된 관계형 데이터베이스다. SQLite는 ACID 특성을 지닌 트랜잭션을 통해 데이터를 관리하며, 데이터베이스를 하나의 파일로 다루어서 어플리케이션에서도 쉽게 이용할 수 있다. 안드로이

드 운영체제는 어플리케이션의 중요 데이터를 SQLite 데이터베이스 파일에 저장한다. 안드로이드 운영체제가 SQLite를 사용하는 이유는 데이터베이스의 특징인 안정성을 확보하기 위해서다. SQLite를 통해 어플리케이션의 중요 데이터를 관리하면 XML, JSON등의 단순 파일로 관리하는 것보다 오류나 이상으로 인한 데이터 변질, 유실을 방지할 수 있다.

2.2.1 SQLite의 구조

SQLite는 하나의 데이터베이스를 하나의 파일로 표현한다. 이 파일을 메인 데이터베이스 파일이라고 한다. SQLite에서 사용하는 메인 데이터베이스 파일은 하나 이상의 페이지들로 구성되며, 페이지의 크기는 512바이트에서 65536바이트로, 페이지의 크기 값은 메인 데이터베이스 파일의 헤더에 존재한다. 페이지는 다시 최소 저장단위인 셀로 나뉘게 된다. 페이지와 셀의 구조는 [그림 2-3]과 같다. 페이지의 메타데이터를 가진 헤더 이후에 2Byte의 셀 포인터가 존재하고, 셀 포인터는 해당 셀이 존재하는 오프셋을 가리킨다. 셀은 최하단에서 부터 차례대로 사용되며, 셀 포인터와 셀 사이에는 아직 사용되지 않는 빈 구역이 존재한다. 이렇게 사용되지 않는 구역을 Unallocated space라고 한다.

Unallocated space는 셀과 셀 포인터 사이에만 존재하지 않고 다른 영역에도 존재할 수 있다. 이를 관리하는 것이 Freeblock이다. Freeblock은 빈 셀을 가리키는 체인 구조로, 첫 번째 Freeblock은 페이지의 헤더에 오프셋이 존재한다. 오프셋에 따라 첫 번째 Freeblock을 탐색하면 다음 Freeblock을 가리키는 오프셋이 존재한다. 이런 체인 구조를 통해서 그 페이지가 가진 Unallocated space를 확인할 수 있으며, 이후에 셀의 데이터가 삭제되면 이를 체인 구조에 추가시켜서 Unallocated space에 해당

셀을 추가하는 방식으로 관리한다.

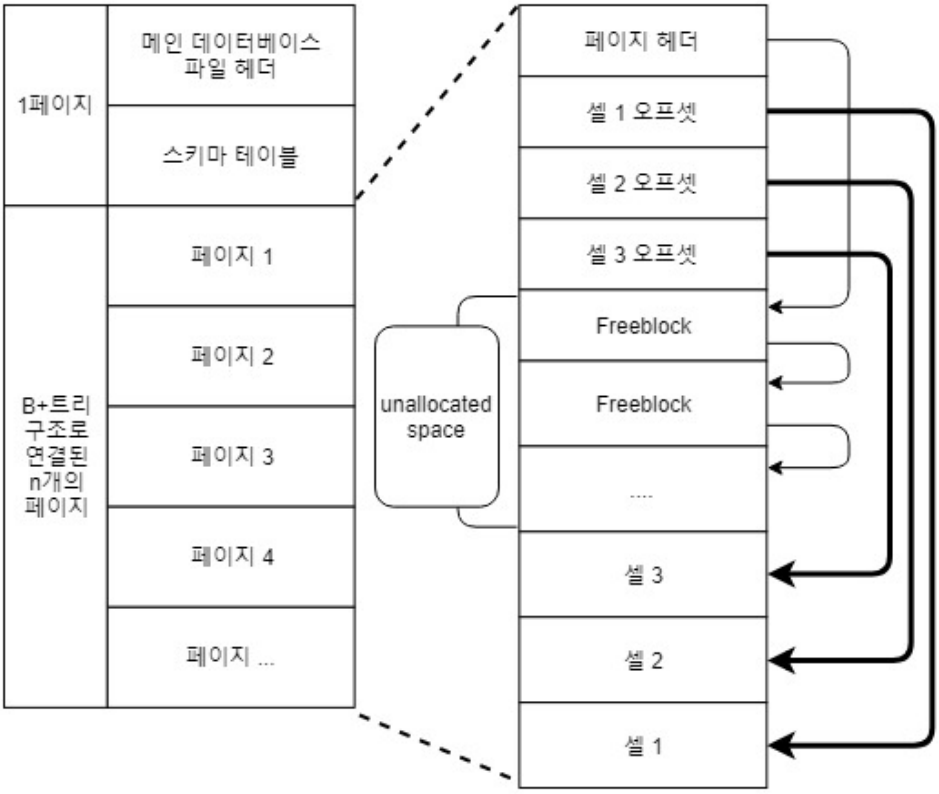

그림 2-3 SQLite 메인 데이터베이스 파일의 구조

2.2.2 SQLite의 저널링

SQLite는 데이터베이스이기 때문에 ACID 특성을 가진 트랜잭션 연산을 통해서 데이터를 조작한다. 데이터의 안정성과 관련된 ACID 특성을 얻기 위해서 SQLite는 저널링 작업을 트랜잭션 작업마다 진행한다. SQLite의 저널링은 크게 2가지 방식이 존재한다. 저널 파일이라는 백업 파일을 만들어 관리하는 방식과 트랜잭션 작업을 로그로 작성해서 로그 파일에 기록하는 WAL(Write-Ahead Log) 방식이다. 저널 파일은 상대

적으로 느리지만 더 나은 안정성을 가지고, WAL 방식은 상대적으로 빠르지만 다수의 데이터베이스가 동시에 작업할 때에는 안정성이 제대로 보장되지 않는다.

저널 파일 방식은 저널 파일을 어떻게 다루느냐에 따라서 5가지의 모드로 나뉜다. OFF는 저널 자체를 사용하지 않는 모드로, ACID가 보장되지 않는다. DELETE는 모든 트랜잭션이 끝난 후에 저널 파일을 삭제한다. 이로 인해 트랜잭션 작업마다 시간이 많이 소모된다. TRUNCATE는 저널 파일을 삭제하는 대신 저널 파일의 크기를 0으로 바꾼다. 즉, 저널 파일 내부의 데이터를 실제로 삭제하지 않는다. PERSIST는 트랜잭션 후에 저널 파일을 삭제하지 않고 유지한다. 대신 다음 트랜잭션 작업 때 유지한 저널 파일을 사용할 수 있도록 저널 파일의 헤더를 0으로 덮어쓴다. MEMORY는 저널 파일을 메인 메모리에만 일시적으로 저장한다. 따라서 메인 메모리에 이상이 생기면 저널 파일의 데이터를 잃게 된다. 안드로이드는 과거 DELETE나 TRUNCATE 모드가 기본 모드였지만, 안드로이드 4.1.1(API 16)을 기준으로 PERSIST 모드가 기본 모드가 되었다.

저널 파일은 데이터베이스의 트랜잭션이 일어났을 때, 트랜잭션에 영향을 받는 페이지들을 백업한다. 백업된 페이지들은 트랜잭션 작업 이전의 데이터를 가지며, 해당 페이지들을 백업하는 작업이 트랜잭션 보다 먼저 발생한다. 만약 저널 파일이 존재하지 않으면 저널 파일을 새로 생성한다. 저널 파일의 이름은 메인 데이터베이스 파일과 동일하고 확장자만 차이가 있다. 따라서 메인 데이터베이스 파일 이름과 동일한 이름의 저널 파일을 찾아야 맞는 저널 파일을 찾은 것이다.

2.3 Journaling of Journal

Journaling of Journal은 안드로이드에서 발생하는 이상으로, SQLite 트랜잭션 작업 과정에서 발생한다. SQLite는 트랜잭션을 시작하기 전에 저널 파일을 생성한다. 이 저널 파일에는 SQLite 메인 데이터베이스 파일의 페이지 중, 트랜잭션에 영향을 받는 페이지들의 데이터가 백업된다. 이때, SQLite는 ExT4의 파일 입출력 연산을 사용하고, ExT4는 해당 연산들이 적용되기 전에 저널 작업을 처리한다. 따라서, 안드로이드 운영체제는 하나의 트랜잭션 작업을 시행할 때 마다 저널링 작업을 처리해야하며, 이는 연산의 중복과 데이터의 중복을 발생시킨다. 이러한 중복 작업은 안드로이드의 성능을 크게 저하시키는 요인이지만, 이를 해결하는 기법들도 제시되고 있다. 그러나 저널링 모드 자체를 사용하지 않는 것이 아니라 파일 입출력의 처리량을 줄이는 방식으로 작용하고 있다.

2.3.1 Journaling of Journal 절차

Journaling of Journal의 절차는 다음과 같다.

1. SQLite가 트랜잭션 연산을 수행하기 전에 저널 파일을 생성하는데, ExT4가 메타데이터를 ExT4 저널 영역에 백업한다.
2. SQLite가 트랜잭션 연산을 시작하기 전에 저장 매체에 저널 파일을 생성하고 페이지를 백업한다.
3. SQLite 트랜잭션 연산이 적용되기 전에 ExT4가 메타데이터를 ExT4 저널 영역에 백업한다.
4. SQLite가 트랜잭션 연산을 수행하고 디스크에 변경사항을 적용한다.

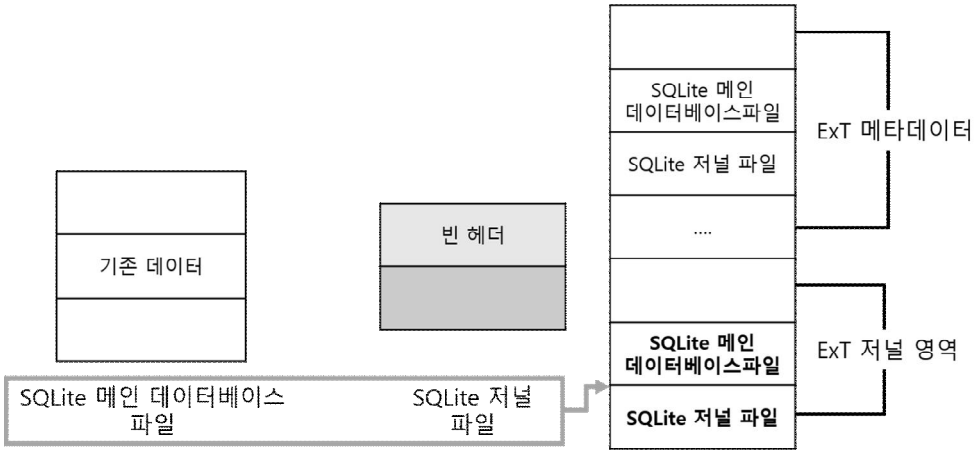

그림 2-4 Journaling of Journal 1단계

[그림 2-4]에서 알 수 있듯이 Journaling of Journal은 SQLite의 트랜잭션에서 발생한다. SQLite는 트랜잭션 연산을 시작하기에 앞서 먼저 SQLite 메인 데이터베이스 파일의 페이지를 백업할 저널 파일을 생성하거나 기존 저널 파일을 수정해 저널링을 한다. [그림 2-4]는 SQLite의 저널 파일 생성 작업이 처리되기 전에 해당 저널 파일의 메타데이터가 ExT4 저널 영역에 저널링 되는 작업을 나타낸다. SQLite 저널링 모드 PERSIST는 저널 파일 생성 작업이 없는 경우도 있으나 저널 파일에 새로운 헤더 값을 입력하기 때문에 보다 작은 양의 메타데이터가 ExT4에 저널링될 뿐, 저널링 작업 자체는 이루어진다. 실제 파일 생성, 수정 작업이 처리된 것은 [그림 2-5]다.

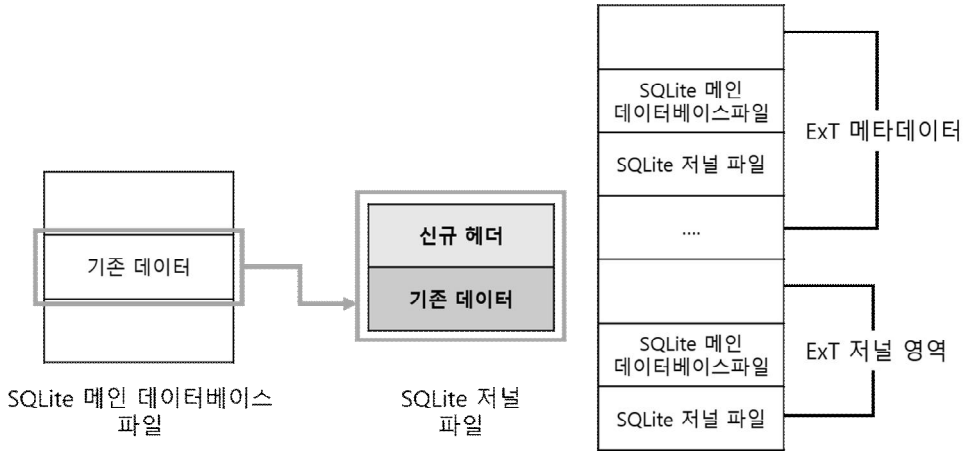

그림 2-5 Journaling of Journal 2단계

이후 SQLite 메인 데이터베이스 파일에 적용될 트랜잭션 연산에 대한 작업이 시작되면 [그림 2-6]와 같이 그 이전에 SQLite 메인 데이터베이스 파일에 대한 저널링 작업이 이루어지고 [그림 2-7]에서 실제 트랜잭션 연산이 처리된다.

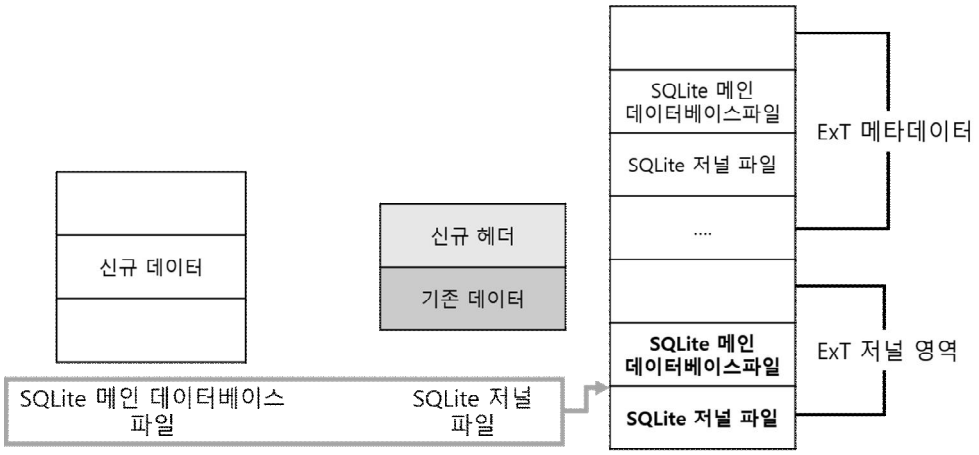

그림 2-6 Journaling of Journal 3단계

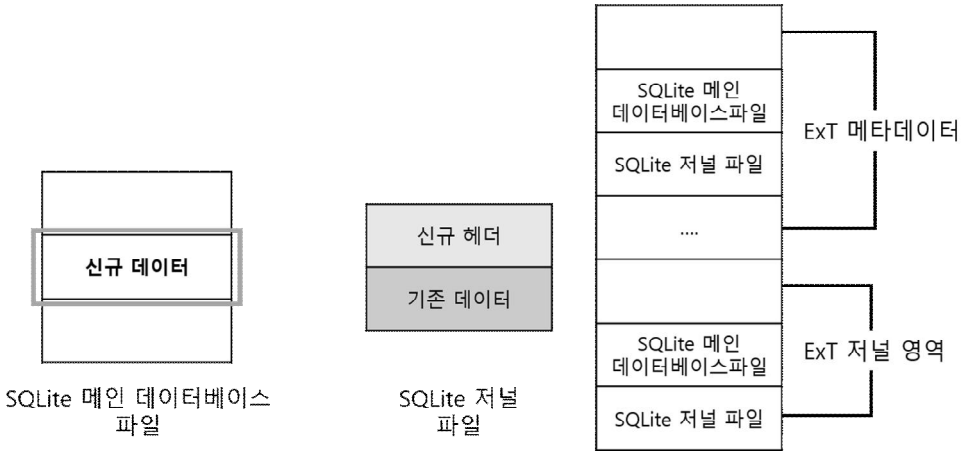

그림 2-7 Journaling of Journal 4단계

2.4 디지털 포렌식

디지털 포렌식은 디지털 증거를 확보하는 기법을 말한다. 디지털 포렌식의 최종 목표가 디지털 증거인 만큼 증거로써 효력이 있는 디지털 증거를 확보해야만 한다. 법적으로 효력이 있는 디지털 증거는 위법증거수집배제원칙을 준수해야한다. 따라서 사전에 법적으로 허가된 영역에서 확보한 디지털 증거만이 인정되고 그 외의 영역에서 발견된 증거는 위법하게 수집한 증거가 되어 효력을 잃는다. 또한 전문증거 문제로 인해 사건에 대한 직접적인 증거가 아닌, 단순히 관련 내용이 사용자의 자의로 작성된 경우는 예외적인 경우를 제외하면 전문증거가 되어 직접적인 증거가 되지 못한다.

디지털 포렌식을 위한 삭제 데이터 복구 기법은 크게 2가지로 나누어서 생각할 수 있다. 파일 시스템에 존재하는 파일의 메타데이터를 기반으로 복구하는 것과 파일 또는 데이터 자체가 가지는 고유한 특징, 시그니처 등을 기반으로 탐색, 복구하는 것이다. 파일의 메타데이터를 기반으

로 복구하는 방법은 파일 시스템 영역과 메타데이터가 가리키는 영역만 조사하면 되기 때문에 오탐지 가능성이 적고 더 신속한 복구가 가능하다. 그러나 파일의 메타데이터를 인위적으로 변조하거나, 파일 시스템에서 파일을 삭제할 때 메타데이터까지 같이 삭제하면 복구가 어려워진다. 파일 또는 데이터의 고유한 특징, 시그니처를 기반으로 분석하는 것은 탐색 영역이 넓지 않으면 해당하는 파일이나 데이터를 찾지 못할 수 있기 때문에 저장매체의 전 부분을 조사해야한다. 이렇게 탐색 영역을 대상 또는 집합 전체로 설정하고 조사하는 것을 전수 조사라고 한다. 전수 조사는 전체 영역을 조사하기 때문에 많은 시간이 소모되고, 오탐지의 가능성도 커진다.

제 3 장 관련 연구

안드로이드 운영체제에서 데이터를 복구하기 위해 여러 기법들이 제시되어왔다. 안드로이드 운영체제가 채택한 ExT4와 SQLite의 특징을 분석해서 기법들이 제시되어 왔고, 그에 따라 ExT4의 파일 카빙 기법을 개선하거나 저널 영역을 분석해서 최신 메타데이터와 저널 영역의 백업된 메타데이터를 비교, 복구하는 기법도 제시되었다. SQLite가 가진 특징인 데이터베이스가 파일로 표현된다는 점을 이용해, SQLite 파일 내부 구조를 분석해서 삭제된 데이터를 복구하는 기법도 제시되었다.

3.1 파일 카빙 개선

파일 카빙은 복구 기법 중에서 메타데이터가 아닌 파일, 데이터의 고유 데이터나 구조를 통해서 복구를 하는 기법이다. 파일이 가지는 고유 데이터를 시그니처라고 하는데, 이 시그니처를 통해서 파일의 시작, 끝을 파악해서 복구하는 것이다. 파일 카빙의 문제점은 2가지다. 첫 번째는 파일 데이터가 이어져서 저장되지 않고 단편화 되었을 때, 복구가 어려워진다. 두 번째는 파일 카빙에 소요되는 시간이 길다는 것이다. 이러한 두 가지 문제점을 안드로이드 운영체제로 복구 대상 플랫폼을 제한해서 연구한 논문이 있다. 해당 논문은 안드로이드 운영체제에서 복구 대상이 될 주요 파일, 데이터를 선별하고, 이에 제외되는 파일들과 데이터를 단편화를 일으키는 것으로 파악해서 추가로 분석한다. 사용자 정보를 담는 SQLite 메인 데이터베이스 파일은 분석 대상이지만 SQLite 저널 파일은 분석 대상이 아니며, 오히려 SQLite 메인 데이터베이스 파일의 단편화를 일으킨다. 따라서 단편화를 일으키는 부분을 무시하고 이후에 이어지는 내용을 분석해서 SQLite 메인 데이터베이스 파일을 복구한다.

이후 후속 연구를 통해서 SQLite 저널 파일을 복구하는 기법도 제시되었다. SQLite 저널 파일의 헤더를 분석해서 SQLite 저널 파일을 복구하는 것이다. 그러나 저널 파일의 헤더를 분석하는 것은 안드로이드의 SQLite 저널 모드의 기본 값이 TRUNCATE에서 PERSIST로 바뀌면서 파일 삭제가 아닌 헤더 덮어쓰기가 되었기 때문에 해당 기법을 사용할 수 없다. SQLite 저널 파일은 헤더에 시그니처가 존재하기 때문에 헤더가 덮어 쓰여지면 시그니처도 삭제된다. 파일 카빙의 가장 큰 문제점인 단서가 없어 전체 영역을 분석하는 전수 조사로 파일을 카빙하는 부분도 단점이다. ExT4의 메타데이터나 저널 영역을 분석해서 단서를 확보할 수 있으나, 기존 기법은 그러한 기능이 존재하지 않는다.

3.2 ExT4 저널 영역 분석

ExT4의 저널 영역에 백업된 데이터를 통해서 데이터를 복구하는 기법도 제시되었다. 저널 영역의 크기가 작아 복구할 데이터가 적다는 논문도 있었으나, 안드로이드 운영체제의 경우 ExT4 저널 영역에서 백업하는 데이터가 메타데이터로 한정되기 때문에 해당 메타데이터를 통해서 데이터 복구의 단서를 확보할 수 있다. 저널 영역 분석 기법은 메타데이터와 저널 영역에 백업된 메타데이터를 비교하는 것이다. 파일 시스템이 관리하고 있는 최신 메타데이터와 저널 영역에 백업된 메타데이터의 차이점을 비교해서 실제 데이터 영역의 삭제, 변경된 데이터를 추적한다. 이를 위해서 저널 영역의 저널 로그를 분석해야하는데, 하나의 저널 로그가 파일 시스템 작업 한 번의 저널 로그이기 때문이다. [그림 3-1]은 해당 논문에서 분석한 저널 로그의 구조를 나타내며, 이 내부에서 필요한 메타데이터를 확보해야한다.

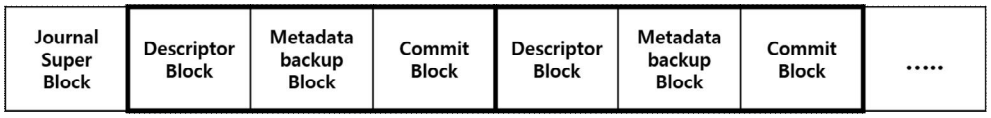

그림 3-1 저널 로그의 구조

복구 기법의 절차는 다음과 같다.

1. 저널 영역을 전부 확보하기위해, 저널 Super블록의 내용을 무시하고 이후에 이어지는 디스크립터 블록을 전부 확보.
2. 하나의 저널 로그는 하나의 디스크립터 블록으로 시작하고, 그 후에 하나의 commit 블록이나 revoke 블록으로 끝난다.
3. 각 저널 로그를 위의 방식으로 확보.
4. 각각의 데이터 블록을 확보하기 위해 디스크립터 블록에서 가리키는 디스크립터 엔트리를 분석, 데이터블록을 확보.
5. 이후에는 저널 영역을 분석하며 메타데이터 블록을 확보
6. 마지막으로 commit 블록이나 revoke 블록을 확보하면 하나의 저널로그 확보 완료.
7. 이를 저널 영역이 끝날 때까지 반복.
8. 확보한 디스크립터 블록의 블록 번호와 메타데이터 블록의 번호를 비교해서 유효성 확인
9. 만약 두 블록의 데이터 중, 디렉토리 엔트리가 변경되었다면 해당 데이터의 변경이 있었다고 확인할 수 있음.
10. 해당 디렉토리 엔트리의 블록 비트맵을 비교, 변경이 있다면 데이터의 변경이 있었다고 예상 가능
11. 확보된 블록 비트맵 정보들 중 해당 디렉토리 엔트리의 비트맵 정보를 통해 삭제된 블록이 있는지 확인.

12. 삭제, 변경된 블록을 조사해서 파일 카빙.

기존 기법은 하나의 저널 로그에서 디렉토리 엔트리, 블록 비트맵, I-node 비트맵, I-node 테이블을 확보하면 데이터를 복구할 수 있다고 제시한다. 또한 저널 로그가 단순히 메타데이터의 백업이기 때문에 사용자가 어플리케이션을 사용해서 발생하는 저널과 시스템이 유지, 관리를 위해 발생하는 저널의 차이를 구분할 수 없다고 밝혔다. 해당 부분들을 제안 기법에서 개선할 수 있다.

3.3 SQLite 파일 내부 삭제 데이터 복구

SQLite 메인 데이터베이스 파일의 페이지 내부에는 이후에 데이터가 입력되거나 저장되는 Unallocated space가 존재한다. 기본적으로 Cell과 Cell pointer 사이에 존재하는 Unallocated space는 SQLite의 작업에 따라 확장될 수 있다. 이러한 확장 중에서 삭제 연산에 따라서도 확장이 일어난다. 페이지 내부의 셀이 삭제되면 해당 셀이 가진 내부 데이터를 직접 삭제하거나 덮어쓰는 것이 아니라, 해당 셀의 오프셋이 Free block 체인 구조에 포함된다. 이후, 페이지 내부에 데이터가 입력, 변경되어서 새로운 셀이 필요할 때 Freeblock 체인 구조를 통해 셀을 확보한다. 따라서 삭제 작업 이후에도 작업이 진행되지 않았다면 셀을 분석해서 데이터를 복구하는 것이 가능하다. [그림 3-2]는 SQLite 메인 데이터베이스 파일 내부의 삭제된 데이터를 복구하는 절차를 나타낸다. 먼저 Free block의 오프셋을 전부 확보하고 이를 통해서 Unallocated Area를 확보한다. Unallocated Area를 셀로 나누어 데이터를 분석하면 삭제, 변경된 데이터를 추적해서 복구할 수 있다.

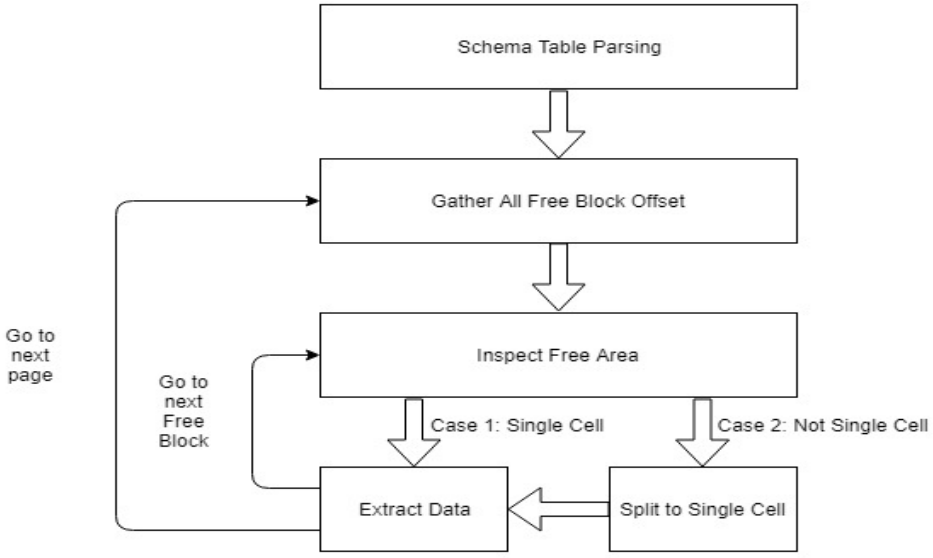

그림 3-2 SQLite 파일 내부 레코드 복구 절차

그러나 내부의 데이터를 사용자가 직접 삭제하면 해당 데이터는 복구할 수 없다. 해당 방식의 삭제 방지 기법이 존재하며, 삭제 방지 기법은 마찬가지로 Free block을 추적해서 Free block 내부에 데이터가 존재하면 이를 0으로 덮어쓰는 것으로 복구를 방지한다. 본 논문의 제안 기법은 이러한 SQLite 메인 데이터베이스 파일의 변조에도 영향을 받지 않는 SQLite 저널 파일을 기반으로 삭제 데이터를 복구한다. SQLite 저널 파일은 작업에 영향을 받는 페이지 데이터를 백업하기 때문에 메인 데이터베이스 파일의 데이터와 백업된 페이지 데이터를 비교하는 것으로 삭제, 변경된 데이터를 추적해서 복구할 수 있다.

3.4 각 기법에 대한 비교

파일 카빙 개선, 저널 영역 분석, SQLite 삭제 데이터 복구 모두 탐색하는 영역에서 차이가 있다. 파일 카빙 개선 기법은 안드로이드 운영체

제가 관리한 데이터 영역 전체가 대상이고, 저널 영역 분석 기법은 ExT4 메타데이터와 저널 영역이 분석 대상이다. SQLite 삭제 데이터 복구 기법은 SQLite 메인 데이터베이스 파일 내부가 분석 대상이다. 그러나 각 기법들 모두 대상 영역 외부에서 단서를 확보하는 기능이 없었다. 이로 인해 복구 대상 영역이 변조 되면 복구가 불가능한 문제가 있다. [표 3-1]은 각 기법들이 가진 기능에 대한 정리다.

표 3-1 기존 기법 간의 기능 비교

	파일 카빙 알고리즘개선	저널 로그 기반 복구	SQL 삭제 레코드 복구
삭제/변경 DB 파일 복구	O	O	X
트랜잭션 이전 데이터 확보	O	O	O
저널 영역 활용 여부	X	O	X
저널 로그 기반 타임 라인 파악	X	X	X
최신 기종 저널 파일 복구	X	X	X

저널 영역을 활용하지 않기 때문에 파일 카빙 기법이 많은 시간이 소모되며, 전체 영역을 조사해야하기 때문에 디지털 포렌식에서 중요한 개인정보 침해 방지 부분에서 법적 문제가 발생할 수 있다. 또한 디지털 포렌식에서 사용자의 작업 여부와 운영체제가 자동으로 실행한 작업, 사용자의 작업 내용을 분석하는 것은 중요한 작업이지만 제안 기법들은 저널 로그에서 해당 단서를 확보할 수 없었다.

제 4 장 Journaling of Journal 기반 SQLite 파일 복구 기법

4.1 Journaling of Journal 분석

Journaling of Journal이 발생한 후에 저장매체의 데이터를 확인하면 [그림 4-1]와 같이 메타데이터가 ExT4의 저널 영역에 백업되어 있는 것을 확인할 수 있다. [그림 4-1]에서 SQLite 메인 데이터베이스 파일과 저널 파일이 삭제되었을 때를 나타내고 있다.

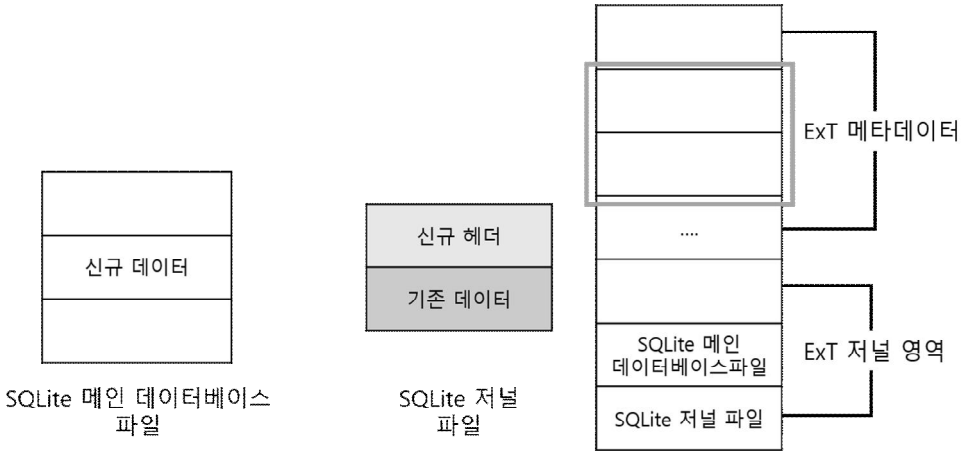

그림 4-1 Journaling of Journal 이상 이후 삭제 작업을 실행한 경우

ExT4에서 파일을 삭제하면 ExT4에서 관리하던 해당 파일의 메타데이터는 삭제되지만 파일 데이터는 삭제되지 않기 때문에 SQLie 메인 데이터베이스 파일과 저널 파일 자체는 저장매체 상에 남아있다. 메타데이터가 존재하지 않기 때문에 SQLite 메인 데이터베이스 파일을 복구하기 전까지 SQLite 파일에 접근해 내부의 레코드를 복구하는 것은 불가능하

다. SQLite 메인 데이터베이스 파일을 확보하기 위해서는 SQLite 파일이 가진 시그니처를 기반으로 파일 카빙을 시도해야하지만, 특별한 단서가 없기 때문에 데이터 영역 전체가 복구 대상이 된다. 이 상태에서 저널 영역은 앞서 언급된 삭제 연산에 영향을 받지 않고 이전의 백업된 메타데이터를 그대로 가지고 있다. 백업된 메타데이터는 삭제된 SQLite 메인 데이터베이스 파일과 저널 파일의 복구에 사용될 수 있다. 기존의 저널 로그 기반 복구 기법은 복구를 위해서 디렉토리 엔트리, 블록 비트맵, I-node 비트맵, I-node 테이블을 확보해서 해당 작업으로 변경된 메타데이터를 추적해서 변경 이전의 데이터를 복구했다. 그러나 SQLite는 트랜잭션의 변경된 데이터를 저널 파일에 백업하기 때문에 저널 파일을 확보하면 변경 이전의 데이터를 확보할 수 있다. 파일 복구에 필요한 메타데이터는 디렉토리 엔트리로, 이는 저널 로그에서 메인 데이터베이스 파일, 저널 파일의 이름이나 메인 데이터베이스 파일의 확장자 “.db”, 저널 파일의 확장자 “.db-journal”을 검색하면 찾을 수 있다. 디렉토리 엔트리는 [표 4-1]에 나오듯이 해당하는 파일의 이름과 디렉토리 엔트리 번호를 가진다.

표 4-1 디렉토리 엔트리 구조

오프셋	설명
0x0	해당 디렉토리 엔트리가 가리키는 아이노드
0x4	디렉토리 엔트리의 길이
0x6	파일 이름의 길이
0x8	파일 이름 데이터

디렉토리 엔트리는 블록 그룹별로 할당된다. 따라서 디렉토리 엔트리

의 번호를 분석하면 블록 그룹을 파악할 수 있다. 블록 그룹을 파악하면 해당 파일의 데이터가 존재하는 영역을 파악할 수 있다. 각 블록 그룹의 크기와 I-node 개수는 ExT4 슈퍼 블록에 명시되어있다.

표 4-2 ExT4 슈퍼 블록 구조의 일부

오프셋	설명
0x0	총 I-node 개수
0x4	총 블록 개수
0x8	super-user만 관리 가능한 정보
0xC	할당되지 않은 블록 개수
0x10	할당되지 않은 I-node 개수
0x14	첫 번째 데이터 블록
0x18	블록의 크기
0x1C	클러스터의 크기
0x20	블록 그룹 당 블록 개수
0x24	블록 그룹 당 클러스터 개수
0x28	블록 그룹 당 I-node 개수
...	....

해당하는 블록 그룹을 파악했다면 이후 파일 카빙을 시도할 때, 해당 구역을 분석한다. 특정 어플리케이션의 정보를 얻어야 할 때에는 해당 어플리케이션이 관리하는 메인 데이터베이스 파일의 이름을 파악해서 저널 로그에서 검색하고 분석한다. 최대한 많은 파일들을 확보할 때에는 저널 로그에서 모든 디렉토리 엔트리를 확보해서 분석한다.

4.2 Journaling of Journal 기반 SQLite 파일 복구절차

Journaling of Journal 기반 SQLite 파일 복구의 절차는 다음과 같다.

1. ExT4 저널 영역 및 저널 로그 확보
2. ExT4 저널 로그에서 SQLite 메인 데이터베이스 파일, 저널 파일의 디렉토리 엔트리 확보
3. 디렉토리 엔트리의 엔트리 번호를 통해서 속한 블록 그룹 추적
4. 해당 블록 그룹 내부에서 SQLite 메인 데이터베이스 파일을 카빙
5. 저널 파일은 디렉토리 엔트리에서 I-node를 확보, 이를 토대로 복구
6. 저널 로그를 분석해서 SQLite 데이터베이스의 작업 내역을 추적

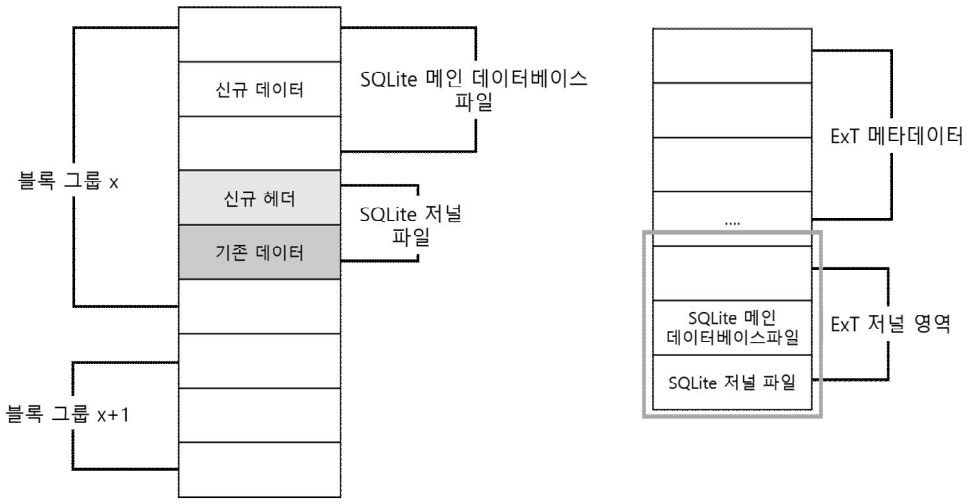

그림 4-2 Journaling of Journal 기반 SQLite 파일 복구 기법 1단계

1단계에서는 ExT4 저널 영역과 저널 로그를 확보한다. 저널 영역은 저널 블록들로 이루어진다. 저널 블록은 모두 동일한 헤더를 가지고, 헤더 내부의 플래그 값으로 종류를 구분한다. 저널 블록의 헤더는 [표

4-2]의 구조로 이루어져있다.

표 4-3 ExT4 저널 블록 헤더 구조

offset	데이터
0x0	저널 블록의 시그니처 “0xC03B3998”이 존재한다.
0x4	블록 종류를 나타내는 플래그 값이 존재한다. 블록 종류로는 저널 슈퍼 블록, 디스크립터 블록, 커밋 블록, 리보크 블록이 있다.
0x8	해당 블록이 저널링한 트랜잭션의 ID

저널 영역은 저널 수퍼 블록으로 시작하기 때문에 저널 수퍼 블록을 가장 먼저 찾는다. 저널 수퍼 블록을 확보하면 그 다음부터 저널 로그를 확보한다. 저널 로그는 저널 디스크립터 블록으로 시작해서 저널 커밋 블록이나 저널 리보크 블록으로 끝난다. 저널 디스크립터 블록이 발견될 때까지 블록을 순차적으로 검사하고, 저널 디스크립터 블록이 발견되면 저널 로그의 트랜잭션 ID를 확보한다. 트랜잭션 ID를 이용해서 저널 로그의 따라서, 저널 커밋 블록이나 저널 리보크 블록 이후에 저널 디스크립터 블록이 나오지 않으면 저널 영역이 끝났다는 것을 알 수 있다.

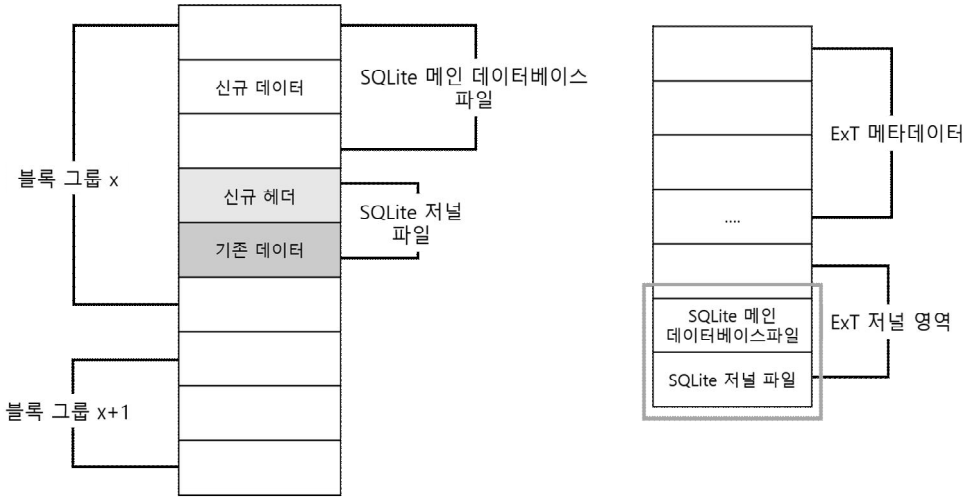

그림 4-3 Journaling of Journal 기반 SQLite 파일 복구 기법 2단계

2단계에서는 확보한 저널 로그들에서 SQLite 메인 데이터베이스 파일, SQLite 저널 파일의 디렉토리 엔트리를 찾는다. 디렉토리 엔트리는 확장자를 포함한 파일의 이름 데이터를 가지고 있기 때문에, 저널 로그의 데이터를 SQLite 메인 데이터베이스 파일의 확장자 “.db”, SQLite 저널 파일의 확장자“.db-journal”로 탐색하면 디렉토리 엔트리를 탐색할 수 있다. 오탐색을 방지하기 위해, 이름 데이터가 탐색 될 때 마다 [표 4-1]의 디렉토리 엔트리 구조에 맞는지 검사한다. 정확한 위치에 디렉토리 엔트리 길이 값이 있는지, 파일명 길이 값과 실제 파일명 길이가 같은지를 검사해서 디렉토리 엔트리가 맞는지 확인한다. 디렉토리 엔트리의 파일명 길이가 255를 넘어간다면 이는 디렉토리 엔트리가 아니므로 오탐색에 해당하니 무시한다.

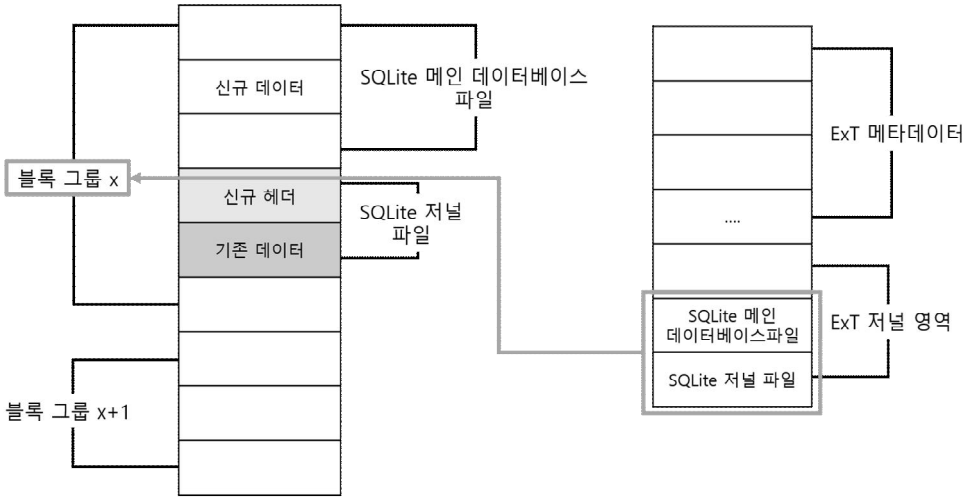

그림 4-4 Journaling of Journal 기반 SQLite 파일 복구 기법 3단계

3단계에서는 확보한 디렉토리 엔트리에 있는 I-node 정보를 토대로 해당 디렉토리 엔트리가 속한 블록 그룹을 추적한다. ExT4의 블록 그룹은 각각 일정한 수의 I-node를 할당받는다. 이 값은 ExT4 수퍼 블록에 저장되어 있다. ExT4 수퍼 블록은 저널 영역 앞에 존재하며, ExT4 슈퍼 블록의 시그니처는 “0xEF53”이다. 이 값을 추적해서 ExT4 수퍼 블록을 확보한다. ExT4 슈퍼 블록에서 블록 그룹별로 할당 받는 I-node 개수를 확인하면 디렉토리 엔트리의 I-node 정보를 분석해서 디렉토리 엔트리가 어느 블록 그룹에 속하는지 파악한다.

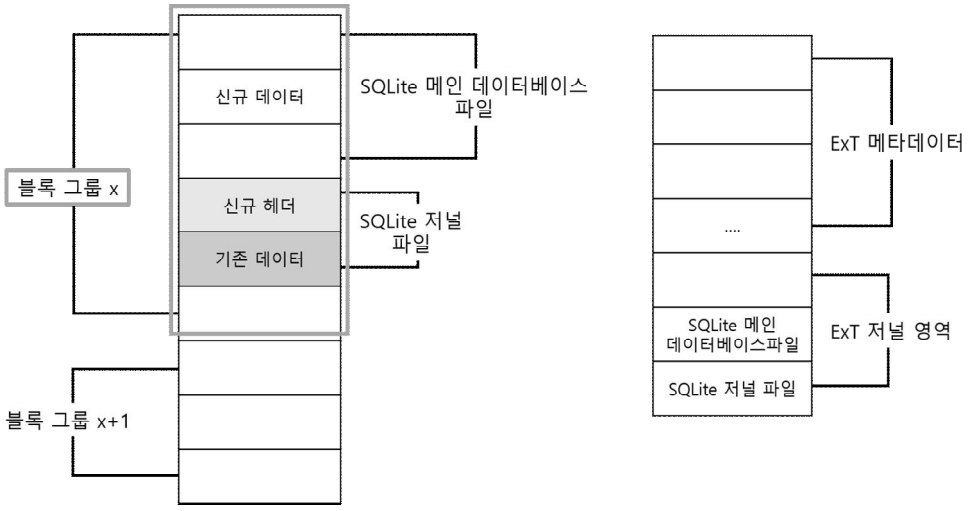

그림 4-5 Journaling of Journal 기반 SQLite 파일 복구 기법 4단계

4단계에서는 해당하는 블록 그룹에서 SQLite 메인 데이터베이스 파일을 카빙한다. 기존의 전체 영역을 카빙하는 방식과는 달리, 특정 블록 그룹만을 카빙하기 때문에 카빙 영역이 크게 줄어든다. 기존 기법은 데이터 영역 전체를 분석해야 하지만, Journaling of Journal 기반 SQLite 파일 복구 기법은 3단계에서 분석 후 파악된 파일의 디렉토리 엔트리가 할당된 블록 그룹의 영역만 분석해서 분석 영역의 크기가 줄어든다. SQLite 파일은 전부 시그니처로 파일을 나타내는 매직 넘버가 존재한다. SQLite 메인 데이터베이스 파일 카빙은 이 시그니처를 기반으로 SQLite 헤더를 확보하고, 이후에 SQLite 페이지들을 순차적으로 확보하는 것을 목표로 한다. 그러나 저널 파일은 시그니처 기반 파일 카빙으로 복구하기 어려운데, 최신 안드로이드가 SQLite에 설정한 저널링 모드에서는 시그니처가 주기적으로 삭제되기 때문이다. 따라서 Journaling of Journal 기반 SQLite 파일 복구 기법에서는 디렉토리 엔트리의 I-node 데이터를 I-node 구조체에 접근하는 데에 활용한다.

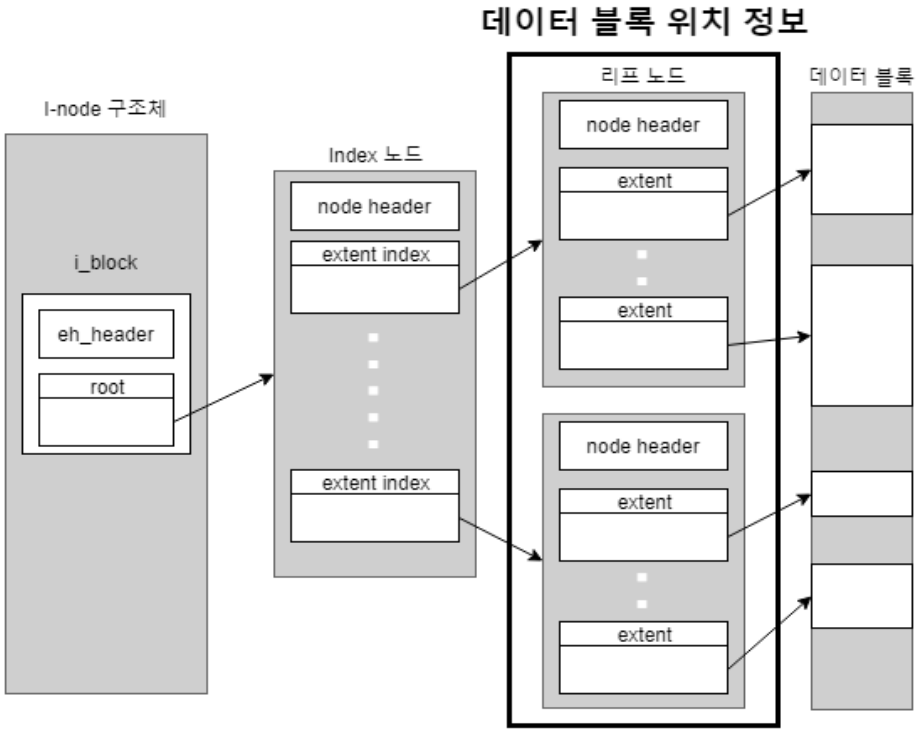

그림 4-6 Extents 트리 구조와 데이터 블록 위치 정보의 위치

5단계에서는 SQLite 저널 파일을 I-node 값을 기반으로 분석해서 복구한다. I-node 값 또한 디렉토리 엔트리에 존재하므로, 디렉토리 엔트리에서 I-node를 확보해 해당하는 블록 그룹의 I-node 테이블, I-node 비트맵 테이블 등을 활용해서 복구한다. I-node 구조체는 파일이 할당받은 데이터 블록의 위치를 가진 extents 구조체를 가지고 있다. extents 구조체들은 트리 구조로 관리되고 있으며, 리프 노드에 해당하는 extents 구조체가 데이터 블록의 위치를 가진다. 기존의 파일 카빙 기법은 저널 파일을 시그니처 기반 파일 카빙 기법으로 복구했으나 최신 기종의 안드로이드에서는 저널 헤더가 트랜잭션 종료 후에 0으로 덮어써지기 때문에 시그니처 기반 파일 카빙 기법은 적용할 수 없다.

6단계에서는 ExT4 저널 로그를 분석해서 SQLite 파일이 어떤 작업을 했는지 파악한다. 만약 ExT4 저널 로그에 SQLite 메인 데이터베이스 파일과 SQLite 저널 파일의 디렉토리 엔트리가 동시에 존재한다면 이는 데이터베이스에 트랜잭션이 있었다고 예측할 수 있다. 반대로 SQLite 메인데이터베이스 파일만 존재하고 SQLite 저널 파일이 존재하지 않으면 이는 트랜잭션 작업이 아닌, 다른 작업으로 인해 메타데이터가 백업되었다는 것을 알 수 있다. 기존 기법은 저널 로그가 사용자의 작업이나 운영체제의 작업을 구분하지 않고 저널링한다. 이로 인해 기존 기법은 특정 저널로그가 사용자의 작업에 대한 저널 로그인지, 운영체제가 자동으로 실행하는 작업에 대한 저널 로그인지 구분할 수 없다. Journaling of Journal 이상을 고려하면 특정 저널 로그가 트랜잭션으로 인해 작성되었는지, 다른 작업으로 인해 작성되었는지 파악할 수 있다.

4.3 Journaling of Journal 기반 SQLite 분석

Journaling of Journal 기반 SQLite 파일 복구 기법은 삭제/변경 된 데이터베이스 파일을 파일 카빙 기법으로 복구할 수 있으며, 복구 대상 영역이 축소되기 때문에 카빙 속도 또한 빨라지리라 기대할 수 있다.

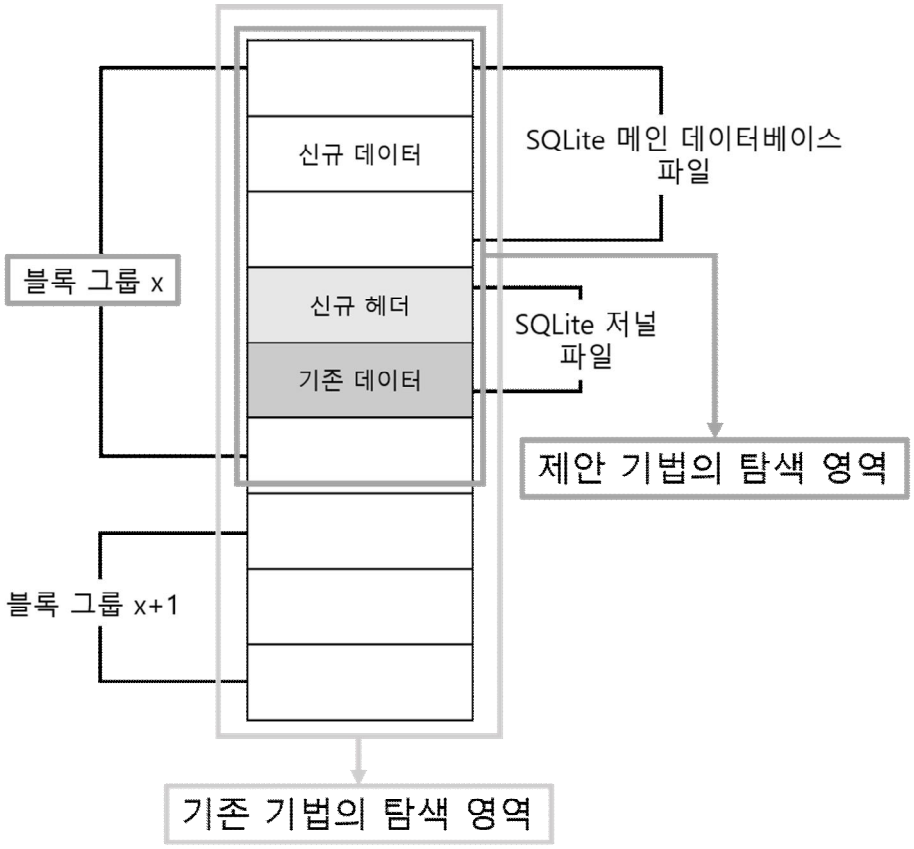

그림 4-7 기존 파일 카빙 기법과 제안 기법의 탐색 영역 비교

트랜잭션 이전의 데이터는 저널 파일을 복구해 저널 파일이 백업한 트랜잭션 이전의 데이터를 확보한다. 기존의 삭제 레코드 복구 기법은 SQLite 메인 데이터베이스 파일에 복구 방지 기법으로 파일 내부 구조와 데이터가 변경되면 복구를 할 수 없었지만, 저널 파일을 기반으로 한

복구는 해당 기법에 영향을 받지 않으므로 데이터 복구가 가능하다.

저널 로그 내부의 Journaling of Journal 이상 발생 여부를 분석해서 해당 파일의 작업 종류를 유추할 수 있다. Journaling of Journal 이상이 발생하면 저널 로그에 SQLite 메인 데이터베이스 파일과 저널 파일이 동시에 기록된다는 점을 이용해서 SQLite의 트랜잭션 여부를 분석할 수 있으며, 이를 통해 사용자의 작업을 유추할 수 있다.

저널 파일 확보의 경우, 최신 기종들이 저널 모드의 기본 모드를 TRUNCATE에서 PERSIST로 변경했기 때문에 기존의 파일 카빙 기법으로는 복구할 수 없었다. Journaling of Journal 기반 SQLite 파일 복구 기법에서는 디렉토리 엔트리를 확보, I-node 데이터를 기반으로 복구한다.

표 4-3 기존 기법과 제안 기법 간의 기능 비교

	파일 카빙 알고리즘개선	저널 로그 기반 복구	SQL 삭제 레코드 복구	제안 기법
삭제/변경 DB 파일 복구	O	O	X	O
트랜잭션 이전 데이터 확보	O	O	O	O
저널 영역 활용 여부	X	O	X	O
저널 로그 기반 타임 라인 파악	X	X	X	O
최신 기종 저널 파일 복구	X	X	X	O

제 5 장 실험 및 평가

5.1 실험 환경

실험 환경은 다음과 같았다.

먼저 안드로이드 환경을 구현하기 위해서 가상 머신 플랫폼 VMware Player14로 만든 가상머신에 Android-x86 이라는 오픈 소스 프로젝트의 데스크톱 용 안드로이드 운영체제를 설치했다. 가상 머신은 모두 동일한 환경을 구축하되, 저장소의 크기만 다르게 해서 총 3가지의 가상 머신이 존재했다. 가상 머신은 각각 10GB, 15GB, 30GB의 저장소를 가지고, 2GB의 메인 메모리를 가지도록 설정했다. 저장소의 포맷은 파티션을 나누지 않고 전 영역을 ExT4가 설치된 하나의 파티션이 존재하도록 설정했다. 이 가상 머신들에서 dd 연산을 사용해 파일 이미지를 확보했다.

파일 이미지가 확보된 후에, 이 이미지를 호스트 컴퓨터에서 기존의 파일 카빙 기법과 제안 기법을 각각 실행해 결과를 비교하고, 제안 기법에서 출력한 추가 데이터를 분석했다. 파일 복구는 발견되는 모든 SQLite 파일을 대상으로 했으며, 각 파일의 발견 시간을 별개로 작성해서 각 파일 별 탐색 시간을 확인했다. 호스트 컴퓨터의 사양은 CPU는 Intel ® CoreTM i5-8600 CPU @ 3.10 GHz이고 메인 메모리는 16GB다.

5.2 실험 결과

먼저 분석한 것은 복구 속도와 영역이었다. [그림 5-1]을 ExT4의 저장소 범위인 블록 그룹(그룹 당 8MB)을 기준으로 분석했을 때, 제안 기법은 기존 기법 보다 탐색 영역을 1/40에서 1/60까지 축소했다. [그림 5-1]

에서 확인할 수 있듯이 탐색 영역이 축소되면서 탐색 시간은 1/2에서 최대 1/3까지 줄어들었다.

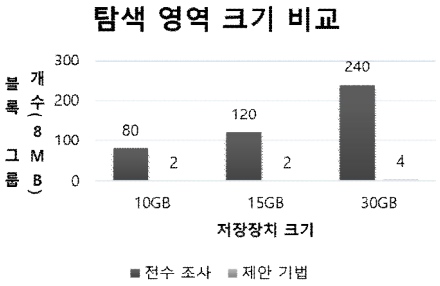

그림 5-1 탐색 영역 크기 비교

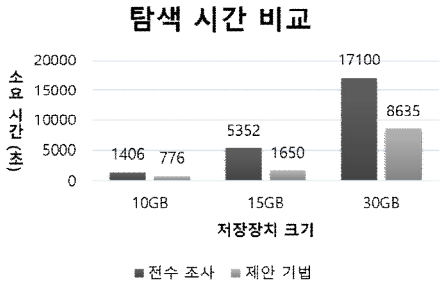

그림 5-2 탐색 시간 비교

탐색 영역의 축소로 인해 누락되는 파일이 존재하는지 확인해본 결과, 전체 파일 중에서 제안 기법이 기존 기법에 비해 평균 6.33%의 파일 탐색을 누락 했다.

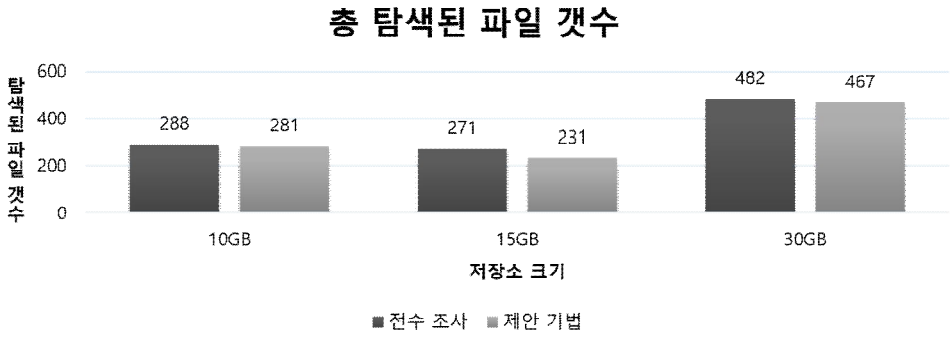

그림 5-3 총 탐색된 파일 갯수

[그림 5-3]은 특정 파일을 탐색에 걸린 시간을 확인했다. 탐색 대상 파일인 locksettings.db 파일을 탐색하는 데 걸린 시간을 측정했다. 실험 결

과 제안 기법이 기존 기법보다 약 42배에서 약 78배까지 더 빠르게 탐색했다.

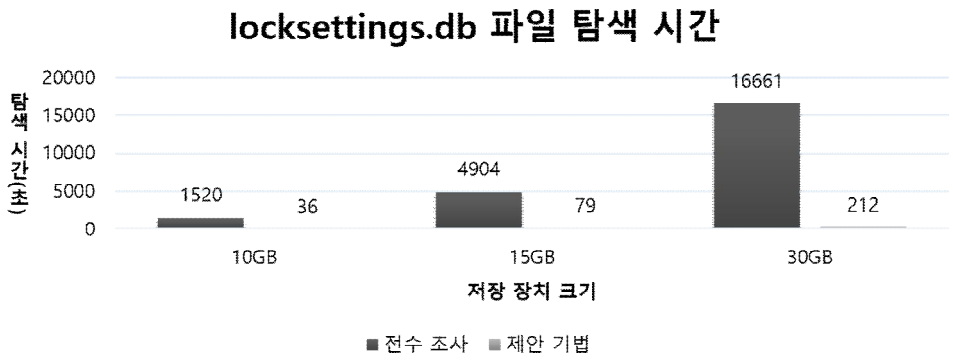

그림 5-4 locksettings.db 파일 탐색 시간

저널 영역에서 벗어나기 쉬운 오래된 파일의 경우, 환경 설치 후 8시간 뒤에 확보한 이미지에서 이루어진 카빙 실험 결과 전 이미지에서 해당 파일이 복구 되었다. 오래된 파일의 복구 시간도 제안 기법이 기존 기법보다 약 5배에서 약 22배 더 빠르게 복구 되었다.

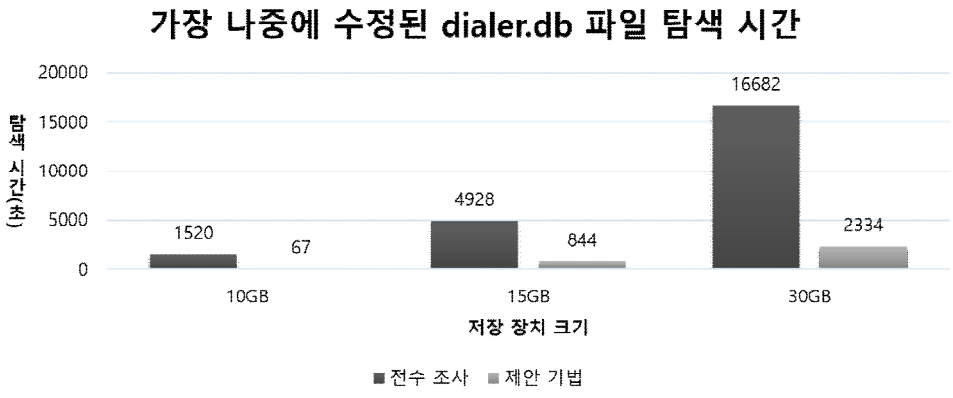

그림 5-5 가장 나중에 수정된 dialer.db 파일 탐색 시간

저널 로그의 데이터 분석은 실제 저널 로그에서 확보된 데이터를 기반으로 확인할 수 있었다. [표 5-1]은 저널 로그에서 확보된 데이터로, 이는 실제 사용자가 작업한 내용과 일치했다.

표 5-1 10GB 이미지에서 확보한 저널 로그의 일부 발췌

저널 로그 번호	저널링된 SQLite 메인 데이터베이스 파일/저널 파일	확인된 정보	실제 작업
5710	search_index DB/저널 파일	DB&저널 파일 저널링 -> 트랜잭션 확인	파일 탐색
5697	search_index DB/저널 파일	DB&저널 파일 저널링 -> 트랜잭션 확인	파일 탐색
5640	notification_log DB/저널 파일 locksettings DB 파일	DB&저널 파일 저널링 -> 트랜잭션 확인	안드로이드 notification
....			

10GB 이미지 파일에서 확보한 저널 로그에서 5710 저널과 5697 저널은 search_index.db 파일과 search_index.db-journal 파일의 디렉토리 엔트리가 존재했다. 저널 로그에서 SQLite 메인 데이터베이스 파일과 저널 파일의 디렉토리 엔트리가 나온 것을 통해 트랜잭션이 존재했음을 유추할 수 있다. 이 때 실제 작업은 안드로이드 내부의 파일 탐색 및 복사 작업이었다. search_index.db 파일은 안드로이드 상에서 파일 검색 기능을 사용할 때, 해당 검색 내역이 저장되는 파일이다. 따라서 검색 내역이

저장되면서 트랜잭션이 있었음을 알 수 있다. 반대로 5640저널은 notification_log.db 파일과 notification_log.db-journal 파일, 그리고 locksettings.db 파일의 디렉토리 엔트리만 확인되었으며, 대응하는 locksettings.db-journal 파일은 확인 되지 않았다. notification_log.db 파일은 저널 파일과 동시에 디렉토리 엔트리가 존재하기 때문에 트랜잭션이 존재했음을 유추할 수 있다. 반대로 locksettings.db 파일은 대응하는 저널 파일이 저널 로그에 존재하지 않기 때문에 트랜잭션이 없었으리라 유추할 수 있다. 이는 실제 작업과 일치한다. notification._log.db는 안드로이드에서 발생한 알림 로그를 관리하는 데이터베이스다. 실제 작업은 탐색 작업을 위해 터미널 어플리케이션을 실행하는 작업이었으며, 터미널 어플리케이션이 실행되었다는 알림이 나타났다. locksettings.db는 안드로이드에서 설정된 PIN 등의 암호를 관리하는 데이터베이스다. 그러나 해당 작업은 암호를 변경하는 것이 아니라 단순히 암호를 확인하고 입력된 값과 비교하는 작업이며, 해당 디바이스에는 PIN 등의 암호가 설정되어있지 않았다. 실제 데이터베이스 내부에 데이터를 변경하는 트랜잭션을 발생하지 않는다. 따라서 저널 로그의 정보를 토대로 실제 작업을 유추하는 것이 가능하다.

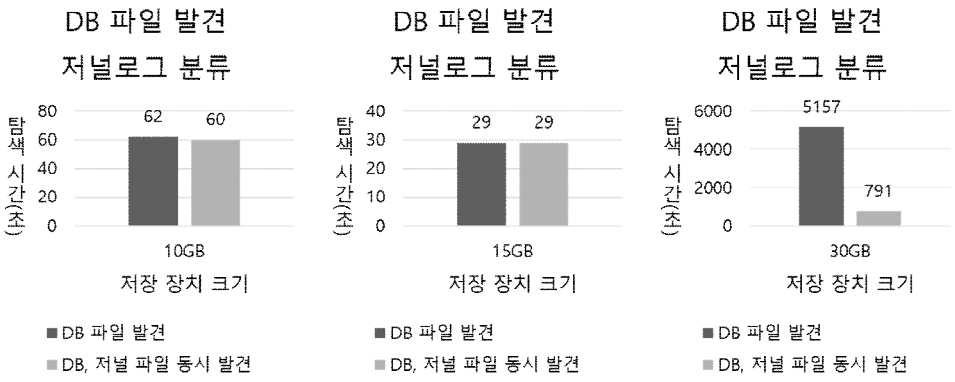

그림 5-6 DB 파일 발견 저널 로그 분류

저널 파일 복구의 경우, 기존 기법은 SQLite 저널 파일의 헤더에 존재하는 시그니처를 기반으로 복구한다. 그러나 최근 안드로이드 운영체제는 트랜잭션이 끝나면 저널 파일의 헤더를 0으로 덮어써서 다음 트랜잭션 작업 때 저널 파일을 재활용할 준비를 한다. 이로 인해 시그니처도 유실되기 때문에 기존 기법은 저널 파일을 복구할 수 없다. Journaling of Journal 기반 SQLite 파일 복구 기법은 저널 로그를 통해서 확보한 저널 파일에서 트랜잭션 이전의 데이터를 확보한다. SQLite 메인 데이터베이스 파일이 발견된 저널로그 중에서 SQLite 메인 데이터베이스 파일과 저널 파일이 동시에 발견되는 경우가 확인된다. 해당 상황에서는 SQLite 메인 데이터베이스 파일에 맞는 저널 파일을 확보해 I-node 비트맵, I-node 테이블을 활용해서 트랜잭션 적용 전의 데이터를 확보할 수 있다.

5.3 총평

실험한 기능 및 성능 부분은 다음과 같았다.

1. 삭제/변경 DB 파일 복구가 가능하면서 이전 기법보다 나은 성능을 보였는가?

2. 저널 영역을 활용해서 복구에 단서가 되는 데이터를 확보할 수 있었는가?
3. 저널 로그에서 작업 내역이나 타임라인을 확보할 수 있었는가?
4. 트랜잭션 이전의 데이터를 확보할 수 있었는가?
5. 최신 기종(안드로이드 7.0)에서 저널 파일을 확보할 수 있었는가?

해당 부분의 구현 및 성능을 확인한 결과, 1번 기능의 경우 기존 기법보다 탐색 영역을 1/40에서 1/60까지, 탐색 시간은 1/2에서 1/3까지 감소했다. 탐색 영역이 감소하면서 발생할 수 있는 누락 파일 문제의 경우, 전체 파일 중 평균 6.33퍼센트의 누락이 있었다. 2번 기능의 경우, 저널 영역을 활용해서 복구에 단서가 되는 데이터인 디렉토리 엔트리를 확보할 수 있었으며, SQLite 메인 데이터베이스 파일과 저널 파일의 디렉토리 엔트리가 같이 백업 됐는가의 여부를 검사해서 3번 기능인 트랜잭션 작업 여부를 파악할 수 있었다. 또한 4번 기능인 트랜잭션 이전의 데이터를 확보하는 기능은 SQLite 메인 데이터베이스 파일에 대응하는 저널 파일의 디렉토리 엔트리를 확보하는 것으로 해결했다. 또한 이렇게 저널 영역에서 메타데이터를 확보하는 방식은 최신 안드로이드 운영체제가 저널 파일의 헤더를 변경하는 것에 상관없이 저널 영역에 메타데이터가 남는다는 것이 확인 되었다.

표 5-2 각 기능 구현 여부

	구현 여부
삭제/변경 DB 파일 복구	O
저널 영역 활용 여부	O
저널 로그 기반 타임 라인 파악	O
트랜잭션 이전 데이터 확보	O
최신 기종 저널 파일 복구	O

Journaling of Journal 기반 SQLite 파일 복구 기법은 삭제되거나 변경된 SQLite 메인 데이터베이스 파일을 확보할 수 있다는 것을 실험을 통해 확인했다. 해당 복구 과정에서 저널 영역을 활용해서 탐색 영역과 탐색 시간을 줄일 수 있는 것도 확인되었다. 또한 저널 로그 기반 타임라인 파악이 실제 작업과 일치하는 것을 실험을 통해 확인할 수 있었다. I-node 테이블 기반 복구 작업을 위한 디렉토리 엔트리가 저널 로그에 존재하는 것을 확인했으며, 해당 저널 로그를 통해 저널 파일에 맞는 SQLite 메인 데이터베이스파일을 확보할 수 있다는 것도 실험을 통해 확인할 수 있었다. Journaling of Journal 기반 SQLite 파일 복구 기법은 트랜잭션으로 인해 삭제/변경된 SQLite 데이터를 복구하는 데에 SQLite 저널 파일을 사용해서 SQLite 메인 데이터베이스 파일에 복구 방지 기법이 적용되어도 저널 파일만 확보되면 트랜잭션 이전의 데이터를 확보할 수 있다.

제 6 장 결론 및 향후 계획

모바일 디바이스에 대한 사이버 범죄가 증가하면서 이에 대응하는 디지털 포렌식에 대한 관심도 증가했다. 그러나 기존 기법들이 최근에 업데이트된 안드로이드 운영체제에 대한 복구 기능이 문제가 있었고, Journaling of Journal 이상을 분석해서 추가로 데이터를 확보하는 기능이 없었다. 본 논문에서는 Journaling of Journal 이상을 분석해서 안드로이드 운영체제의 사용자 정보를 보관하는 SQLite 데이터베이스의 파일을 복구하는 기법을 제안했다. 전 영역을 조사하던 기존 파일 카빙 기법과 다르게 저널 영역에서 SQLite 데이터베이스의 파일 위치를 추적해서 파일을 복구하기에 더 빠른 탐색 속도를 보였고, 저널 영역의 데이터를 Journaling of Journal 이상의 특성을 통해서 분석, 트랜잭션의 유무를 유추하는 것이 가능하고 SQLite 메인 데이터베이스 파일에 대응하는 저널 파일을 확보해서 트랜잭션 이전의 데이터도 확보할 수 있다. 또한 저널 파일의 데이터를 저널 영역에서 확보하기에 최신 안드로이드 운영체제에서 발생하는 저널 파일에 대한 복구 방지에 상관없이 저널 파일을 복구 할 수 있었다. 이러한 개선점을 토대로 기존의 메타데이터의 삭제나 변조 등의 복구 방지 기법에 대응하는 복구가 가능할 것으로 보이며, 이를 통해 디지털 포렌식에서 더 효율적으로 중요한 정보를 확보하는 것이 가능하리라 기대한다.

참고문헌

[1] 손성배, 노연진, 이도근, 박성순, 원유집. “커널 버전 별 Ext4 파일 시스템의 Fsync()에 대한 고찰.”. 정보과학회논문지 44 4 (2017): 363-73.

[2] 이규원, 양승제, 황현욱, 김기범, 장태주, 손기욱. “Sqlite 데이터베이스의 삭제된 오버플로우 데이터 복구 기법.”. 한국정보기술학회논문지 10 11 (2012): 143-53.

[3] 정준석, 정원용. “임베디드 개발자를 위한 파일시스템의 원리와 실습”. 한빛미디어 (2006): 304-390

[4] 정수만. “I/O Stack Optimization for Smartphones.”. 한양대학교 대학원, 2014.

[5] 김도현. “안드로이드 운영체제의 Ext4 파일 시스템에서 삭제 파일 카빙 기법”. 고려대학교 정보보호대학원, 2013

[6] 정영훈. “Sqlite 데이터베이스에서 삭제된 레코드 복구 방지 기법.”. 숭실대학교 대학원, 2016.

[7] "사이버 범죄 통계 자료.". 경찰청 사이버안전국, cyberbureau.police.go.kr/share/sub3.jsp?mid=030300. 2018년 12월 13일 접속.

[8] Dewald, Andreas, and Sabine Seufert. "Afeic: Advanced Forensic Ext4 Inode Carving.". Digital Investigation 20 (2017): S83-S91

[9] Kim, Dohyun, Park, Jungheum, Lee, Keun-gi, Lee, Sangjin. “Forensic Analysis of Android Phone Using Ext4 File System Journal Log.”. Future Information Technology, Application, and Service. Springer, 2012. 435-46.

[10] Fairbanks, Kevin D. “An Analysis of Ext4 for Digital Forensics.”.

Digital Investigation 9 (2012): S118-S30.
[11] Cao, Mingming, Suparna Bhattacharya, and Ted Ts'o. "Ext4: The Next Generation of Ext2/3 Filesystem.". LSF. 2007.
[12] Fairbanks, Kevin D., Christopher P. Lee, and III Henry L. Owen. "Forensic Implications of Ext4.". Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. Ed. 1852691: ACM.
[13] Kang, Woon-Hak, Lee, Sang-Won, Moon, Bongki, Oh, Gi-Hwan, Min, Changwoo. "X-FTL: transactional FTL for SQLite databases.". Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.
[14] Kim, Sungmin, Kim, Minseok, Tuan, Dam Quang, Won, Youjip. "Atomic Multi-Database Transaction of Wal Joumaling Mode in Sqlite.". Advanced Communication Technology (ICACT), 2017 19th International Conference on. 2017. IEEE.
[15] Lee, Wongun, Lee, Keonwoo, Son, Hankeun, Kim, Wook-Hee, Nam, Beomseok, Won, Youjip. "Waldio: Eliminating the filesystem journaling in resolving the journaling of journal anomaly". Usenix, 2015.
[16] Mathur, Avantika, Cao, Mingming, Bhattacharya, Suparna, Dilger, Andreas, Tomas, Alex, Vivier, Laurent. "The new ext4 filesystem: current status and future plans.". Proceedings of the Linux symposium. Vol. 2. 2007.
[17] Shen, Kai, Stan Park, and Meng Zhu. "Journaling of journal is (almost) free.". FAST. 2014.

[18] Nokia. “Nokia 2017 Threat Intelligence Report”. 2017.

[19] “Database File Format.". SQLite, www.sqlite.org/fileformat2.html. 2018년 12월 13일 접속.

[20] “ExT4 Disk Layout." ExT4 Wiki, 2018년 8월 16일, ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout. 2018년 12월 13일 접속.