SQLite 활용 실태 조사 및 삭제된 레코드 복원 기법 분석

이성현[O1] 김나경[1] 최연규[1] 김민성[1] 이인수[2] 박수영[2] 최종무[1]
단국대학교[1], 대검찰청 디지털수사과[2]
{leesh0812, nkkim, choiyg, kms0509}@dankook.ac.kr, {insoo21 hilda01}@spo.go.kr, choijm@dankook.ac.kr

Investigation of SQLite Usage and Techniques for Recovering Deleted Records

Seonghyeon Lee[O], Nakyeong Kim, Yeongyu Choi, Minseong Kim
Insoo Lee, Sooyoung Park, Jongmoo Choi
Dankook University, Supreme Prosecutors` Office

요 약

SQLite는 애플리케이션 데이터를 저장하기 위해 광범위하게 사용되는 데이터베이스로, 특히 스마트폰 환경에서 최근 몇 년간 꾸준히 높은 사용률을 보인다. 이는 포렌식 수사에서 SQLite에 저장된 데이터를 살펴볼 필요성을 증가시켰으며, SQLite를 사용하는 애플리케이션에 대한 조사와 SQLite 포렌식 분석을 위한 Undark, SQLite Deleted Record Parser, FQlite 등과 같은 도구의 등장을 촉진했다. 그러나 기존의 연구들은 주로 특정 애플리케이션이나 해외 사례에 집중되어 있었으며, 다양한 복원 기법들이 실제로 어느 정도의 복원율을 보장하는지에 대한 명확한 정보를 제공하지 않았다. 본 연구에서는 국내 애플리케이션에서 SQLite의 사용 실태를 조사하고, 실제 로그에 기반하여 중요성을 확인한다. 또한 대표적인 복원 기술을 선택하여 실제 복원 가능 여부를 검증하고, 도구를 선택하여 SQLite 포렌식 코퍼스에 따른 도구의 복원율 및 프리 페이지 리스트(Free-Page List), 아티팩트 카빙(Artifact Carving)과 같은 기법별 복원율 및 성능을 평가한다.

1. 서 론

오늘날 SQLite는 세계에서 가장 널리 사용되는 데이터베이스 관리 시스템(DBMS) 중 하나이다. 특히 SQLite는 가볍고, 독립적인 특징 덕분에 다양한 애플리케이션에서 채택되었으며, 메시징 애플리케이션 및 검색 엔진에서 높은 사용률을 보여준다[1, 2, 3]. 예를 들어, 브라우저는 방문 기록, 쿠키, 오프라인 데이터 등을 저장하는 데 주로 사용되며, 스마트폰 애플리케이션은 사용자 설정, 메시지 기록 등을 저장하는 데 주로 사용된다. 이러한 광범위한 사용은 포렌식 수사에서 SQLite에 저장된 데이터를 살펴볼 필요성을 증가시켰고, Undark, SQLite Deleted Record Parser, FQlite 등과 같이 다양한 SQLite 포렌식 도구의 등장을 촉진했다[4, 5].

그러나 많은 연구가 해외 사례와 특정 애플리케이션에 집중되어 있어, 국내 사례에 적용하기에는 어려움이 있다. 또한 SQLite 포렌식 코퍼스에서 복원율 평가가 이루어졌으나, 어떠한 복원 기법이 실제 어느 시나리오에서 복원할 수 있는지 알기 어렵다. 이에 본 연구에서는 국내에서 사용 빈도가 높은 애플리케이션에서 SQLite의 사용 여부를 확인하기 위한 데이터베이스 추출을 진행하고, 실제 로그에 기반하여 저장 데이터를 분석한다. 또한 표준화된 포렌식 코퍼스에 따른 도구 및 복원 기법별 레코드 복원 가능 여부를 직접 확인하고 분석을 진행한다.

본 논문의 구조는 다음과 같다. 2장에서는 국내 애플리케이션에서의 사용 실태 조사 및 SQLite 구조 분석을 진행하고, 3장에서는 복원 기법에 대한 설명 및 검증을 진행한다. 4장에서는 표준화된 포렌식 코퍼스에서 기존 도구의 복원을 진행한 뒤, 적용된 복원 기법을 이용해 직접 가능 여부 및 복원율을 평가한다. 이후 5장에서 결론 및 향후 연구 계획으로 마무리한다.

2. 배 경

2.1 모바일 애플리케이션에서 SQLite 사용 실태

본 논문에서는 해외 사례가 아닌 국내 애플리케이션의 SQLite 실사용 현황 분석 및 포렌식 관점에서 SQLite의 중요도 제시를 위해 '2023년 포브스코리아가 선정한 한국인이 많이 사용하는 모바일 애플리케이션'[6]를 사용한다. 그중 상위 30개 애플리케이션을 분석하며, 표 1은 해당 분석에 사용된 모바일 기기를 보인다.

본 연구는 2024년도 정부(법무부)의 재원으로 대검찰청의 지원을 받아 수행된 연구결과임 (디지털수사과-분석기법개발-24-01).

표 1. 실험에 사용된 기기 정보

제조사	OS	기종
Samsung	안드로이드 10 (Quince Tart)	Galaxy S9
	안드로이드 12 (Snow Cone)	Galaxy S10
	안드로이드 11 (RedVelvet Cake)	Galaxy Wide4
Apple	iOS 15.0.2	iPhone 8
	iOS 17.5.1	iPhone SE2

2.1.1 국내 상위 30개 애플리케이션 분석

표 2는 사용률 상위 30개의 애플리케이션에서 사용하는 데이터베이스를 보인다. 안드로이드 환경에서는 2개를 제외한 모든 애플리케이션에서 SQLite를 사용하는 것으로 확인되었으며, 나머지 2개 애플리케이션에서도 SQLite에 보안 적용된 SQLCipher를 사용하였다. iOS 환경에서는 iOS 미지원 으로 인해 29개의 애플리케이션을 대상으로 분석을 진행하였으며 23개 환경에서 SQLite를 사용하고 있는 것으로 확인되었다. 괄호 안 O, X의 경우 데이터의 암호화 여부를 뜻하며, 표기되지 않은 애플리케이션의 경우 사용 데이터 부족 및 보안 정책으로 인해 확인하지 못하였음을 말한다.

표 2. 상위 30개 애플리케이션 사용 데이터베이스

순위	애플리케이션	Android	iOS
1	유튜브	SQLite (X)	SQLite (X)
2	카카오톡	SQLite (O)	SQLite (X)
3	인스타그램	SQLite (O)	SQLite (X)
4	쿠팡	SQLite (X)	SQLite (X)
5	당근마켓	SQLite (X)	SQLite (X)
6	올웨이즈	SQLite	SQLite (X)
7	배달의민족	SQLite (X)	Realm (X)
8	캐시워크	SQLite (X)	Realm (X)
9	네이버지도	SQLCipher (O)	SQLite (X)
10	네이버웹툰	SQLite (X)	Realm (X)
11	파파고	SQLite SQLCipher	SQLite (X)
12	토스	SQLite	SQLite (X)
13	밴드	SQLite	SQLite (X)
14	카카오T	SQLite	SQLite (X)
15	삼성페이	SQLite	지원 X
16	KB국민은행 스타뱅킹	SQLite	SQLite (X)
17	틱톡	SQLite (O)	SQLite (X)
18	넷플릭스	SQLite (X)	SQLite (X)
19	KB Pay	SQLite	SQLite (X)
20	페이북/ISP	SQLite	SQLite (X)
21	T Map	SQLite (X)	SQLite (X)
22	네이버 카페	SQLite SQLCipher	SQLite (O)
23	모니모	SQLite	SQLite (O)
24	아프리카TV	SQLite	SQLite (X)
25	여기어때	SQLite (X)	Realm (X)
26	알리익스프레스	SQLite	SQLite (X)
27	배달 요기요	SQLite	SQLite (X)
28	야놀자	SQLite	SQLite (X)
29	11번가	SQLite (O)	Realm (X)
30	카카오뱅크	SQLite	SQLite (X)

2.1.2 데이터베이스 파일 상세 분석

국내 애플리케이션을 대상으로 한 포렌식 관점에서 SQLite의 중요도가 높다고 판단하기 위해서는, 실제 데이터베이스 파일에 어떠한 데이터가 저장되어 있는지에 대한 분석이 필요하다. 이에 본 연구진은 대상 애플리케이션으로 스트리밍 서비스 2개, 메신저 및 SNS 2개, 지도 2개를 선정하고 해당 애플리케이션에 대한 상세 로그 분석을 진행한다. 표 3은 상세 분석 결과를 보여준다.

표 3. 애플리케이션 SQLite 상세 분석

분류	애플리케이션	내용
스트리밍 서비스	유튜브	검색기록, 계정, 이메일
	넷플릭스	오프라인 데이터 최근 시청 기록
메신저 및 SNS	카카오톡	채팅, 채팅방, 북마크, 친구 목록, 차단 목록
	인스타그램	사용자 정보, 메시지
지도	네이버 지도	검색, 경로, 북마크
	T Map	최근 경로, 목적지, 검색 시간

분석 결과 스트리밍 서비스의 경우 검색 기록, 계정 정보, 프로필 이메일 오프라인 데이터 등의 데이터를 저장한다. 메신저 및 SNS는 데이터베이스에 채팅, 채팅방, 북마크, 친구 목록, 차단 목록과 같은 데이터를 저장하며, 카카오톡 데이터의 경우는 암호화가 되어있는 것으로 확인하였다. 지도 애플리케이션의 경우 대표적으로 경로, 검색 기록, 목적지 등의 데이터를 확인 할 수 있었다. 이처럼, 다양한 애플리케이션들이 SQLite를 활용하여 중요한 사용자 데이터를 저장하고 관리하고 있으며, 이러한 분석 결과는 포렌식 관점에서 SQLite가 높은 중요도가 있음을 보여준다.

2.2 SQLite 구조

그림1은 SQLite 파일(file) 구조 및 페이지(page) 구조를 보여준다. SQLite 파일 구조는 페이지를 기본 단위로 사용하며, 여러 개의 페이지로 구성되어 있다. 페이지 는 순차적으로 배열되며, 페이지 번호는 1번부터 시작된다. 페이지의 일반적인 크기는 4KB이며 사용자가 임의로 지정할 수 있다.

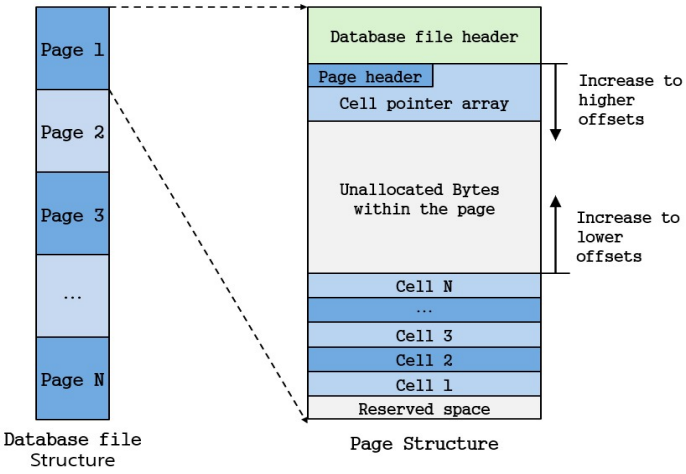

그림 1. SQLite 파일 및 페이지 구조

각 페이지는 여러 영역으로 나뉘며, 이는 데이터베이스 파일 헤더, 페이지 헤더, 셀 포인터 배열, 비 할당 영역, 셀, 그리고 예약 영역으로 구성된다.

- **데이터베이스 파일 헤더:** 첫 번째 페이지에만 존재하며, 100바이트 크기로 구성된다.
- **페이지 헤더:** 페이지의 메타데이터를 저장하고, 페이지 유형, 프리 블록 위치, 페이지 내부 셀 개수 등을 포함한다. 8~12바이트 크기로 구성된다.
- **셀 포인터 배열:** 페이지 내 각 셀의 위치를 저장한다.
- **비 할당 영역:** 마지막 셀 포인터 배열 항목과 첫 번째 셀 시작 부분 사이의 공간으로 아직 사용되지 않는 영역이다.
- **셀:** 실제 데이터를 저장하는 단위로, 페이지의 하단부터 상단으로 확장된다.
- **예약 영역:** 페이지의 가장 끝 부분에 위치하며, 성능 최적화나 데이터 무결성 유지를 목적으로 사용된다.

그림 2는 셀 내부 구조를 보여준다. 셀은 실제 데이터를 저장하는 곳으로 셀 헤더, 레코드 헤더, 레코드 데이터 영역으로 이루어져 있다.

- **셀 헤더:** 셀의 시작 부분에 위치하며, 셀의 크기와 Row ID를 담고 있다. Row ID는 각 레코드를 식별하기 위한 고유한 ID로, 일반적으로 자동 생성된다.
- **레코드 헤더:** 각 데이터 필드의 타입과 길이를 저장한다. 이는 SQLite 내부 데이터 저장 형식을 따른다.
- **데이터 영역:** 실제 데이터가 저장되는 영역으로, 각 데이터는 순차적으로 저장된다.

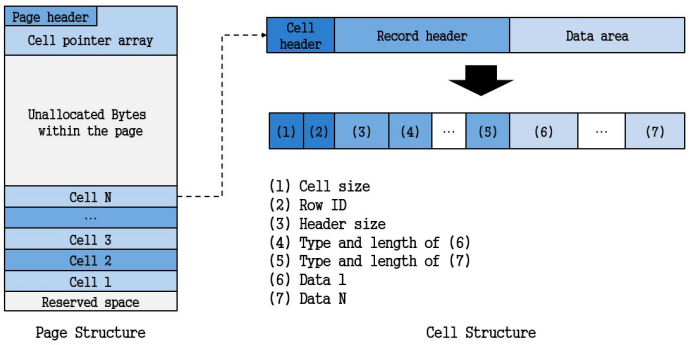

그림 2. SQLite 셀 구조

3. SQLite 삭제된 레코드 복원 기법

3.1 프리 페이지 리스트 분석

프리 페이지 리스트를 통한 삭제된 레코드 복원[7, 8, 9]은 SQLite의 대표적인 복원 기법이다. 데이터베이스 파일의 모든 프리 페이지는 리스트로 서로 연결되어 있으며, 일부는 삭제된 데이터를 포함하고 있을 수 있다. 프리 페이지 리스트를 통해 삭제된 레코드를 복원하는 주요 단계는 다음과 같다.

프리 페이지 리스트: 파일의 헤더는 SQLite 설정을 위해 정의된 구조로 되어 있다. 표 4는 SQLite 파일 첫 40바이트의 구조와 간략한 설명을 보여준다. 자세한 내용은 SQLite 공식 문서에서 확인할 수 있다[10].

표 4. SQLite 헤더 오프셋

Offset	Size	Description
0	16	헤더 문자열 - "SQLite format 3₩000"
16	2	페이지 크기
18	1	파일 형식의 쓰기 버전
19	1	파일 형식의 읽기 버전
20	1	예약된 영역 크기
21	1	최대 페이로드 개수
22	1	최소 페이로드 개수
23	1	리프 페이지의 페이로드 비율, 32 고정
24	4	파일 변경 카운터
28	4	파일의 크기, 페이지 단위
32	4	첫 번째 프리 리스트 트렁크 페이지의 번호
36	4	프리 리스트 페이지의 개수

SQLite 파일의 헤더에서 페이지 크기, 프리 리스트 트렁크 페이지 시작 번호, 프리 페이지 개수 등의 정보를 통해 프리 페이지 리스트의 시작 주소를 계산할 수 있다. 그림 2에서 보듯이 파일 헤더에서 프리 리스트 트렁크 페이지의 시작 번호를 획득하면, 아래 식을 통해 주소를 찾는 것이 가능하다.

$$Address = (offset32 - 1) \times 0x1000 \qquad (1)$$

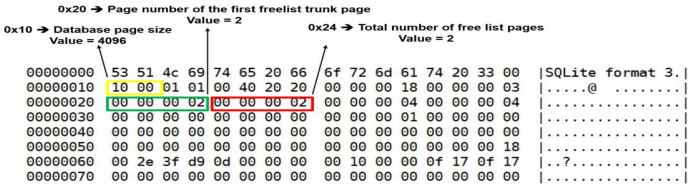

그림 3. 데이터베이스 파일 헤더 예시

페이지 헤더 분석: 각 페이지의 헤더는 리프 페이지의 경우 8바이트, 내부 페이지의 경우 12바이트이다. 헤더를 통해 페이지 형식과 해당 페이지 내부의 셀 개수를 확인할 수 있다. 이후, 셀 포인터 배열에서 셀 내용의 시작 주소를 획득할 수 있다. 그림 4는 페이지 헤더 분석의 예시를 보여준다.

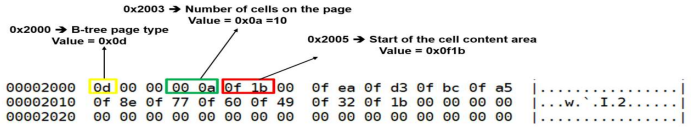

그림 4. 페이지 헤더 예시 (셀 포인터 배열 포함)

레코드 복원: 데이터의 위치를 획득하면 셀 헤더, 레코드 헤더, 그리고 데이터를 확인할 수 있다. 레코드 헤더에 있는 데이터의 타입과 길이를 이용하여 삭제된 데이터가 포함된 페이지의 모든 레코드를 복원할 수 있다. 그림 5는 셀 헤더 및 레코드 헤더를 이용한 데이터 복원의 예시를 보여준다.

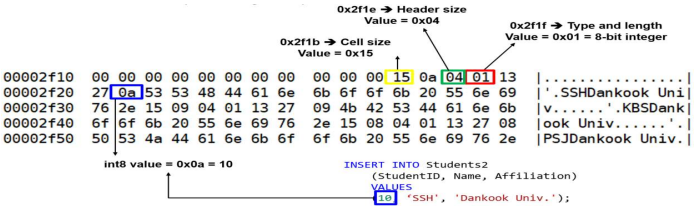

그림 5. 레코드 복원 예시

3.2 아티팩트 카빙

프리 페이지 리스트를 이용한 복원은 메타데이터와 데이터가 온전히 유지되어 있어 비교적 쉽게 복원할 수 있다. 그러나 프리 페이지 리스트가 존재하지 않거나 덮어씌워진 경우, 파일 곳곳에 슬랙 공간(slack space)을 탐색해야 한다. 아티팩트 카빙은 프리 페이지 리스트에 포함되지 않는 파일의 모든 부분을 검사하여 복원을 진행하는 기법이다. 이 기법에서 중요한 요소는 레코드 헤더와 데이터베이스 파일 헤더의 테이블 정보이다. 두 개의 헤더가 덮어씌워진 상태에서는 패턴을 통한 유추가 필수

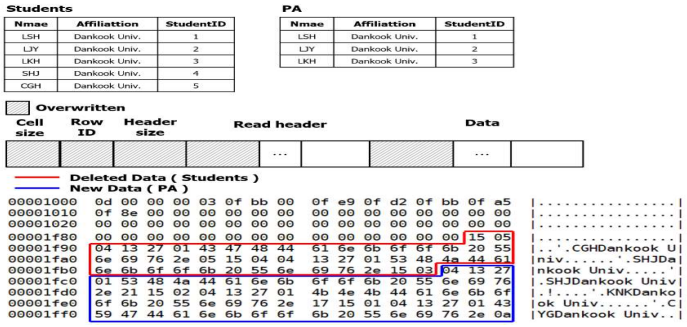

그림 6. 잘못된 복원이 가능한 시나리오

적이어서 복원이 매우 어려워진다.

그림 6은 복원 신뢰도에 문제가 생길 수 있는 시나리오를 보여준다. 이 시나리오에서는 기존 테이블을 삭제한 후, 같은 형식의 새로운 테이블을 생성하고 레코드를 삽입한다. 이 과정에서 SQLite 파일 내부 남아 있는 삭제된 데이터 일부가 덮어씌워질 수 있다.

해당 시나리오에서는 아티팩트 카빙 작업을 통해 삭제된 레코드의 복원이 진행되었다. 그러나 레코드 헤더가 덮어씌워지고, 데이터베이스 파일 헤더에 테이블 정보가 남아 있지 않아 정상적인 복원이 어려웠다. 더욱이, 같은 테이블 구조를 사용하기 때문에 잘못된 테이블로 복원될 수 있다. 결국, 복원에 대한 신뢰도에 문제가 발생하며, 사용자는 이러한 시나리오를 인지하고 주의해야 한다.

4. 실 험

4장에서는 높은 복원율을 보이는 SQLite 레코드 복원 도구인 FQLite[11]를 선택하여 SQLite 포렌식 코퍼스에서 평가를 진행한다. 또한 복원 가능 여부를 검증하고, 각 기술의 복원율을 확인하기 위해 데이터베이스 뷰어 및 Hex dump를 이용하여 직접 프리 페이지 리스트와 아티팩트 카빙을 수행한다.

실험에는 SQLite 포렌식 코퍼스가 사용되었으며, 코퍼스는 77개의 SQLite 파일로 구성되어 있다. 이는 '키워드 및 식별자', '인코딩', '데이터베이스 요소', '트리 및 페이지 구조' 그리고 '삭제 및 덮어씌워진 내용'으로 총 5개의 범주로 이루어진다. 각 범주는 세부 범주로 나뉘며 01부터 0E까지의 16진수로 이름이 지정된다. 이 중 삭제된 레코드 복원 평가를 목적으로 생성된 '삭제 및 덮어씌워진 내용'을 이용하여 평가를 진행한다. 이는 0A부터 0E까지의 항목이 포함되며 삭제된 레코드가 278개 있는 총 27의 SQLite 파일이다.

'삭제 및 덮어씌워진 내용'에서는 삭제된 데이터의 흔적이 남아 있도록 SQLite의 기능인 'secure_delete'를

표 5. SQLite 파일 생성 방법

세부 범주	이름	생성 방법
삭제된 테이블	0A-01	create 1, insert 20, drop
	0A-02	create 1, insert 20, delete 20, drop
	0A-03	create 2, insert 10/each, drop/each
	0A-04	create 2, insert 10/each, delete 10/1, drop each
	0A-05	create 2, insert 10/each, delete 10/each, drop each
삭제 및 덮어쓴 테이블	0B-01	create 1, insert 10, drop, create 1, insert 5
	0B-02	create 3, insert 10/each, drop 1, create 1, insert 5
삭제된 레코드	0C-01	create 1 (int cols), insert 20, delete 7
	0C-02	create 2 (int cols), insert 20/each, delete 5/each
	0C-03	create 1 (text cols), insert 20, delete 7
	0C-04	create 2 (text cols), insert 20/each, delete 5/each
	0C-05	create 2 (int/text), insert 20/each, delete 5/each
	0C-06	create 1 (float cols), insert 20, delete 7
	0C-07	create 2 (float cols), insert 20/each, delete 5/each
	0C-08	create 2 (float/text), insert 20/each, delete 5/each
	0C-09	create 1 (float/text), insert 10, delete 10
	0C-10	create 2 (float/text), insert 10/each, delete 10/each
덮어쓴 레코드	0D-01	create, insert 10, delete 5, insert 3
	0D-02	create, insert 10, delete 5, insert 5
	0D-03	create, insert 10, delete 5, insert 10: match 1
	0D-04	create, insert 10, delete 5, insert 3: match all
	0D-05	create, insert 10, delete 5, insert 5: match all
	0D-06	create, insert 10, delete 10, insert 5: match all
	0D-07	create 2, insert 10/each, delete 5/1, insert 5/2
	0D-08	create 2, insert 10/each (alt), delete 5/1, insert 5/2
삭제된 오버플로우 페이지	0E-01	create, insert 20 (Overflow due to the insertion of large records, one column), delete 7
	0E-02	create, insert 20 (Overflow due to the insertion of large records, many column), delete 5

비활성화하고 삭제 작업을 진행하였다. 표 5는 '삭제 및 덮어씌워진 내용'의 각 SQLite 파일이 어떻게 생성되었는지를 보여준다.

표 6은 SQLite 삭제된 레코드 복원 도구 상세 분석 결과를 보여준다. * 표시는 일부 복원 및 잘못된 복원이 포함된 경우를 뜻한다. 일부 복원의 경우 할당되지 않은 영역을 처리하지 않음으로써 발생하고, 잘못된 복원은 덮어쓰기 및 스키마 구조 인지 실패로 인해 발생한다.

프리 페이지 리스트를 이용한 복원 기법은 아티팩트 카빙에 비해 매우 낮은 복원율을 보였는데, 이는 대부분의 데이터베이스 파일 헤더에 프리 페이지 리스트에 대한 정보가 남아 있지 않아서이다. 일부 가능한 경우에도 페이지 헤더가 이미 손상된 경우가 많았기 때문이다. Undark 및 SQLite Deleted Records Parser는 상대적으로 높은 수치를 보여주었으나, 잘못된 복원에 해당하는 경우가 많았다.

아티팩트 카빙의 경우, 3.2에서 언급한 바와 같이 복원 신뢰도에 문제가 발생하였다. 0B-01에서 이러한 문제가 발견되었으며, 아티팩트 카빙을 통한 삭제된 레코드 복원은 가능했으나, 데이터베이스 파일 헤더에 삭제된 테이블에 대한 정보가 남아 있지 않아 해당 레코드가 어느 테이블에 속해 있었는지를 알 수 없었다.

그림 7-a는 데이터베이스 뷰어 및 Hex dump를 이용하여 기법별 '삭제 및 덮어씌워진 내용' 복원을 수행한 결과이다. 실험 결과 프리 페이지 리스트의 경우 전체의 11.51%로 표 6에서의 결과와 같은 모습을 보여주었다. 아티팩트 카빙은 100%의 복원율을 보여주었다. 이는 아티팩트 카빙이 프리 페이지 공간에 대한 복원 역시 가능함을 나타낸다. 그러나 B0-01에서 보인 신뢰도 문제는 동일하게 발생하였다. 해당 실험은 SQLite 포렌식 코퍼스 '삭제 및 덮어씌워진 내용'에 한정된 실험 결과이다. Secure delete가 수행되거나 덮어쓰기로 인해 패턴을 통한 유추가 강제되면 복원이 불가능 경우가 증가하게 된다. 또한 안티-포렌식을 위한 기능 중 하나인 'VACUUM'과 같은 기능이 수행되면 모든 프리 페이지 및 슬랙 공간이 회수되기 때문에 복원할 수 없다.

그림 7-b는 기법별 성능을 보인다. 해당 실험의 경우 아티팩트 카빙에서 탐지 및 복원 작업은 덮어씌워진 정도 및 데이터 구조에 따라 매우 큰 차이를 보이므로, 기법별 레코드 복원을 위한 페이지 헤더 획득 및 파싱까지의 과정으로 제한을 둔다. 또한 각 기법은 같은 개수의 페이지를 읽도록 설정하였다. 실험은 Intel(R) Core(TM) i7-12700KF, 16GB 메모리, 삼성 870 EVO 1TB 환경에서 수행되었으며, chinook.db 데이터베이스(15,607행, 약 1MB)를 이용하였다.

실험 결과 프리 페이지 리스트를 이용한 기법의 경우 아티팩트 카빙을 이용한 결과보다 약 32.34% 빠른 성능을 보였다. 이는 아티팩트 카빙의 경우 결국 페이지 헤더가 손상되었을 가능성이 있어 8바이트 혹은 12바이트에 해당하는 데이터에 대한 추가 수집이 강제되기 때문이다. 이는 페이지 헤더를 이용한 판별이 가능한 이상적인 경우이며, 추가적인 셀 정보에 대한 비교 작업이 필요할 시에는 상대적으로 긴 작업 시간이 요구된다. 결국 프리 페이지 리스트 및 페이지 유형에 관한 오프셋이 존재하는 경우 상대적으로 빠른 프리 페이지 리스트를 이용하고, 이외의 경우 아티팩트 카빙을 적용하는 것이 효율적이다.

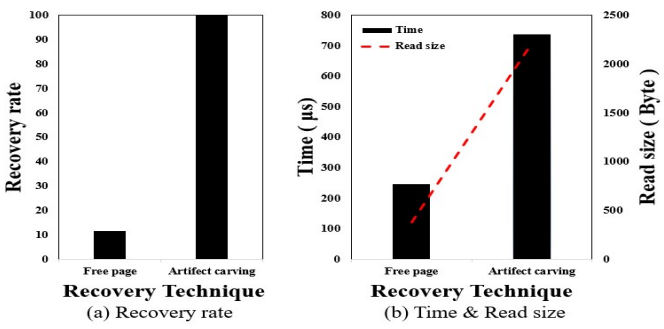

그림 7. 복원 기법별 성능 평가

표 6. SQLite 삭제된 레코드 복원 도구 상세 분석 결과

구분	SQLite 포렌식 코퍼스 저자 테스트[12]						본 논문 저자 테스트		
	Undark	SQLite Deleted Records Parser	SQLabs SQLite Doctor	Stellar Phoenix Repair forSQLite	SysTools SQLite Database Recover	Sanderson Forensic Browser for SQLite	FQLite	Free page	Artifact Carving
0A-01	20/20*	0/20	0/20	0/20	0/20	0/20	20/20	0/20	20/20
0A-02	9/20	20/20*	0/20	0/20	0/20	0/20	20/20	0/20	20/20
0A-03	20/20*	0/20	0/20	0/20	0/20	0/20	20/20	10/20	10/20
0A-04	15/20*	10/20*	0/20	0/20	0/20	0/20	20/20	10/20	10/20
0A-05	11/20*	20/20*	0/20	0/20	0/20	0/20	20/20	10/20	10/20
0B-01*	0/10	0/10	0/10	0/10	0/10	0/10	10/10	5/10	5/10
0B-02	0/10	0/10	0/10	0/10	0/10	0/10	10/10	0/10	10/10
0C-01	0/7	0/7	0/7	0/7	0/7	7/7	7/7	0/7	7/7
0C-02	0/10	0/10	0/10	0/10	0/10	10/10*	10/10	0/10	10/10
0C-03	0/7	7/7	0/7	0/7	0/7	2/7	7/7	0/7	7/7
0C-04	0/10	10/10*	0/10	0/10	0/10	1/10*	10/10	0/10	10/10
0C-05	0/10	10/10*	0/10	0/10	0/10	10/10*	10/10	0/10	10/10
0C-06	0/7	0/7	0/7	0/7	0/7	0/7	7/7	0/7	7/7
0C-07	0/7	0/10	0/7	0/7	0/7	0/7	7/7	0/7	7/7
0C-08	0/10	10/10*	0/7	0/7	0/7	0/7	7/7	0/7	7/7
0C-09	5/10*	10/10*	0/10	0/10	0/10	0/10	10/10	0/10	10/10
0C-10	11/20*	20/20*	0/20	0/20	0/20	0/20	20/20	0/20	20/20
0D-01	0/5	2/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-02	0/5	1/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-03	0/5	0/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-04	0/5	2/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-05	0/5	0/5	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-06	1/10*	5/10*	0/10	0/10	0/10	0/10	10/10	0/10	10/10
0D-07	0/5	5/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0D-08	0/5	5/5*	0/5	0/5	0/5	0/5	5/5	0/5	5/5
0E-01	3/7	2/7	0/7	0/7	0/7	3/7	7/7	2/7	5/7
0E-02	0/5	0/5	0/5	0/5	0/5	0/5	5/5	0/5	5/5

5. 결론

본 논문에서는 국내에서 사용 빈도가 높은 애플리케이션에서 SQLite의 사용 여부를 확인하기 위한 데이터베이스 추출을 진행하고, 실제 로그에 기반하여 포렌식 관점에서 SQLite의 중요도를 분석하였다. 또한, 다양한 복원 기법을 검증하고, 표준화된 SQLite 포렌식 코퍼스를 이용하여 도구 및 복원 기법의 복원율과 성능을 측정하였다. 이 과정에서 삭제된 레코드를 복원하는 데 주의가 필요한 시나리오를 제시하고, 발생할 수 있는 문제를 확인하였다. 이후 추가적인 연구를 통해 사용자 행위의 추론 가능성에 대한 분석과, 새로 입력된 데이터 크기에 따른 복원 가능 여부 및 복원율을 확인하고자 한다.

참고 문헌

[1] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, J. M. Patel, “SQLite: Past, Present, and Future”, VLDB, 2022

[2] Stackoverflow, “Most popular technologies“, 2023, https://survey.stackoverflow.co/2023/

[3] DB-engines, “DB-Engines Ranking” 2024, https://db-engines.com/en/ranking

[4] A. Fitzpatrick, “Here's How Police Get a Suspect's Facebook Information”, 2012, https://mashable.com/archive/police-facebook

[5] E. Kim, K. Park, H. Kim, J. Song, “Design and analysis of enumeration attacks on finding friends with phone numbers: a case study with KakaoTalk”, Computers & Security, 2015

[6] Z. Lee, “2023 한국인이 사랑한 모바일앱 200” https://jmagazine.joins.com/forbes/view/338788, 2023

[7] L. M. Aouad, T. M. Kechadi, R. D. Russo, “Ants road: a new tool for SQLite data recovery on android devices” ICDF2C, 2012

[8] S. Jeon, J. Bang, K. Byun, S. Lee, “A recovery method of deleted record for SQLite database”, PUC, 2012

[9] F. Ramisch, M. Rieger, “Recovery of SQLite data using expired indexes”, IMF, 2015

[10] SQLite, “Database File Format”, https://www.sqlite.org/fileformat.html

[11] D. Pawlaszczyk, C. Hummert, “Making the Invisible Visible – Techniques for Recovering Deleted SQLite Data Records”, CFATI, 2020

[12] S. Nemetz, S. Schmitt, F. Freiling, “A standardized corpus for SQLite database forensics”, DFRWS, 2018