Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.6, Dec. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.6.1347

SQLite 데이터베이스 파일에 대한 데이터 은닉 및 탐지 기법 연구*

이 재 형,[1†] 조 재 형,[1] 홍 기 원,[1] 김 종 성[1,2‡]
[1]국민대학교 금융정보보안학과, [2]국민대학교 정보보안암호수학과

The Research on Data Concealing and Detection of SQLite Database*

Jae-hyoung Lee,[1†] Jaehyung Cho,[1] Kiwon Hong,[1] Jongsung Kim[1,2‡]
[1]Dept. of Financial Information Security, Kookmin University,
[2]Dept. of Information Security, Cryptology and mathematics, Kookmin University

요 약

SQLite는 트랜잭션을 제공하는 파일 기반의 DBMS(Database Management System)이며 경량 플랫폼에 적절하기 때문에 요즘에는 스마트폰에 주로 적재된다. 따라서 스마트폰의 사용량이 증가함에 따라 SQLite와 관련된 범죄가 발생할 가능성이 있다. 본 논문에서는 SQLite 데이터베이스 파일에 대한 새로운 은닉 기법과 이에 대한 탐지 방법을 제안한다. 은닉 실험 결과, 데이터베이스 파일 헤더의 영역 중 70바이트에 고의적으로 데이터를 은닉하는 행위가 가능하였다. 또한 데이터베이스 파일의 페이지 영역을 추가하여 기존의 페이지를 은닉할 수 있었다. 그러나 SQLite 구조를 기반으로 헤더의 70바이트를 파싱하거나 레코드 및 인덱스의 개수를 이용하여 제안하는 은닉을 탐지하는 방법이 가능하였다. 이후, SQLite 은닉된 데이터에 대한 탐지 알고리즘을 제시하였다.

ABSTRACT

SQLite database is a file-based DBMS(Database Management System) that provides transactions, and it is loaded on smartphone because it is appropriate for lightweight platform. AS the usage of smartphone increased, SQLite-related crimes can occur. In this paper, we proposed a new concealing method for SQLite db file and a detection method against it. As a result of concealing experiments, it is possible to intentionally conceal 70bytes in the DB file header and conceal original data by inserting artificial pages. But it can be detected by parsing 70bytes based on SQLite structure or using the number of record and index. After that, we proposed detection algorithm for concealed data.

Keywords: Database, Digital Forensics, SQLite, Concealing, Detection

I. 서 론

데이터베이스란 여러 사람에 의해 공유되어 사용될 목적으로 관리되는 데이터의 집합을 의미한다. 데이터베이스의 초기 모델은 응용프로그램이 데이터베이스를 조작하는 형태가 아닌 데이터베이스를 관리해주는 소프트웨어인 DBMS(database management system)형태로 제공되었다가 키(key)와 값(value)들의 간단한 관계를 테이블화 시킨 RDBMS(relational database management system)형태로 발전하였고, 이후 전 세계적으로 정보화 사회에 없어서는 안 될 중요한 요소 중 하나가

Received(08. 02. 2017), Modified(1st: 09. 29. 2017, 2nd:10. 30. 2017), Accepted(11. 05. 2017)
* 본 논문은 2017년도 정부(과학기술통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 작성되었습니다. (No.2017-0-00344, 최신 모바일 기기에 대한 암호해독 및 포렌식 분석)
† 주저자, pekasuce@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

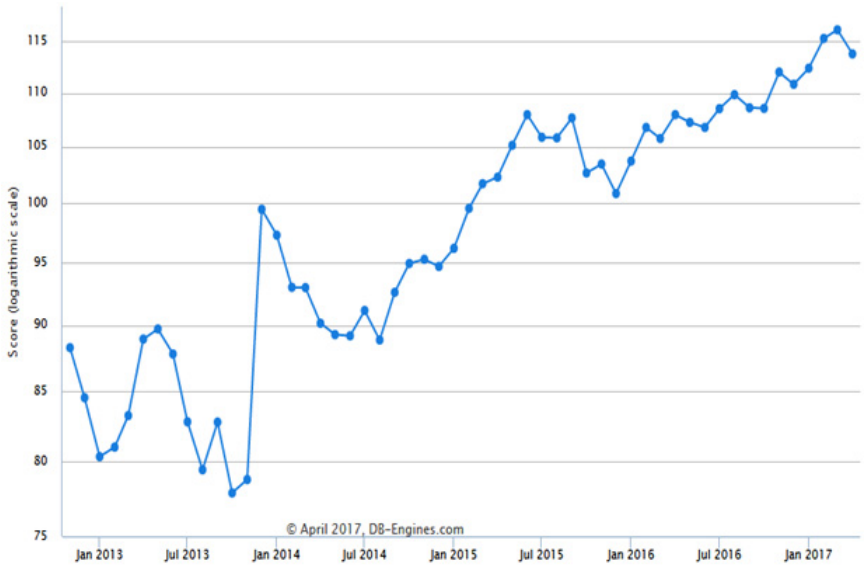

Fig. 1. Trend of SQLite popularity

되었다.

Oracle과 MySQL 같은 서버 형태의 데이터베이스가 데이터베이스 시장의 주를 이루고 있지만 사물인터넷(IoT, internet of things) 시대가 도래 하면서 임베디드 환경에 적합하며 속도도 준수한 SQLite 데이터베이스가 각광을 받기 시작했다. 대표적인 사례로 모바일이 있다. 모바일이 출현하고 데이터가 급증하게 되었기 때문에 모바일 내부 데이터를 효율적으로 저장하고 관리해주기에 적합한 SQLite가 선정되어 이용되었다.

Fig. 1은 데이터베이스의 수요 순위를 제공하는 DB-Engines에서 SQLite의 사용 추이를 나타낸 그래프이다. 웹 사이트에서 데이터베이스 언급 수, 시스템에 대한 일반적인 관심도, 해당 DB가 사용된 네트워크의 수, 소셜 네트워크와의 연관성을 종합하여 순위를 결정한다. SQLite의 사용량은 2013년 이래 점차적으로 증가해왔고, 이는 SQLite의 중요도 또한 증가한다는 사실과 연결된다. 그러나 SQLite의 중요도가 높아지는 만큼 이를 이용하는 범죄 행위들이 발생할 가능성이 있는데, 본 논문에서 제시하는 SQLite에 대한 데이터 은닉이 그 중 하나이다. 예를 들어, 기업의 기밀이 유출된 사건에서 범죄자들은 서로 스마트폰을 사용하여 데이터를 주고받았다고 가정해본다면, 범죄자가 PC를 통해 스마트폰에 탑재된 SQLite 데이터베이스 파일 내부에 중요 데이터를 고의적으로 은닉했을 때 은닉된 데이터를 찾기 어렵다.

따라서 본 논문에서는 SQLite 데이터베이스 파일에 대한 데이터 은닉 기법을 제시하였으며, 해당 기법을 빠른 시간 내에 탐지하는 은닉 탐지에 대한 알고리즘과 도구를 제시한다. 2장 관련연구를 시작으로 데이터베이스 은닉에 관련된 연구를 짚어보며 본 연구의 필요성에 대해 언급한다. 3장에서는 SQLite 데이터베이스의 데이터 저장 메커니즘을 이해하기 위해 전체적인 구조를 먼저 설명한 다음, 헤더와 페이지 내부의 구조를 설명한다. 4장에서는 SQLite 파일에 대한 은닉 기법을 제시한 후, 그에 대한 탐지와 이를 기반으로 알고리즘 및 도구를 5장에서 제안한다. 이후 결론 및 향후 연구를 마지막으로 본 논문을 마친다.

II. 관련 연구

기존 연구로 이근기 외 2명은 2008 한국방송공학회 학술발표대회 논문에 기존의 데이터베이스가 은닉되어 클라이언트 측에서 데이터를 확인할 수 없을 경우 해당 은닉 시스템을 탐지하는 방안을 제시하였다[1]. 해당 논문에서는 데이터베이스 은닉과 관련된 실제 수사 현장에서 수사관과 조직의 협조 여부에 따라 시나리오를 각각 구분하여 대응하는 방법을 제시

한다. 먼저, 수사관과 조직의 협조가 양호한 경우, 형식적으로만 협조하고 정확히 알려주지 않는 경우, 예고치 않은 수사를 실시하여 비협조 하는 경우, 고의적으로 비협조 하는 경우로 나누어 대응 상황을 고려해볼 수 있다. 수사에 비협조적인 두 가지 경우에서는 한 사례로 데이터베이스 시스템에는 접근이 불가능하게 되지만 웹 서버와 같은 기타 서버 시스템에는 문제없이 접근할 수 있는 경우가 있다. 또 다른 사례로는 특정 클라이언트만 접근을 허용하여 해당 데이터베이스를 은닉하는 시스템이 존재할 수 있다. 만약 해당 데이터베이스에 대한 접근 자체를 막게 된다면 무언가 중요한 데이터가 존재할 가능성이 조명될 수 있기 때문에, 접근 자체를 막기 보다는 소수 클라이언트에게만 접근을 허용하는 기법을 선호한다.

데이터베이스를 은닉하는 시스템을 탐지하는 방안으로는 데이터베이스 사용 흔적 수집 기법과 ADO(ActiveX Data Object)를 이용한 데이터베이스 서버 탐지 기법을 제안하였다. 데이터베이스 사용에 대한 흔적은 데이터베이스 파일, 레지스트리, 로그분석을 통하여 정보를 수집할 수 있으며, 각각의 방법에 대하여 사용되는 몇 가지 툴을 제안하여 효율적인 모니터링 기법을 제안하였다.

서울대학교 진종민은 이학석사 학위논문으로 데이터베이스의 은닉행위 식별을 위한 시스템 아티팩트 조사 기법을 제안했다[2]. 데이터베이스 은닉 관련 사건에 대해 조사하는 경우 압수수색 현장에서 전체 데이터 파일의 압수를 위한 근거가 필요하기 때문에, 데이터베이스와 연관된 아티팩트들에 대해 조사해야 한다. 따라서 해당 논문에서는 피압수자의 은닉행위를 규명하기 위해 하드디스크 교체, 데이터베이스 교체, 레코드 삭제의 세 가지 유형으로 구분하여 정의하였다. 하드디스크를 교체한 경우에는 하드디스크의 포맷일시, 윈도우 운영체제의 레지스트리에서 운영체제 설치 일시 등의 관련정보를 획득하여 데이터베이스 은닉 행위를 규명하며, 새로운 데이터베이스를 생성하여 기존 데이터를 은닉하는 경우에는 데이터베이스의 생성 일시, On/Off 일시, 백업 수행 일시 등을 조사하여 데이터베이스의 교체와 백업을 통해 데이터베이스를 은닉했던 흔적을 찾는다. 데이터베이스에서 범죄 혐의와 관련된 데이터를 삭제하는 행위로 레코드 삭제가 자주 발생할 가능성이 있다. 일반적으로 쿼리문을 통해 원하는 레코드를 삭제하기 때문에 쿼리 로그와 트랜잭션 로그의 분석을 통해 해당 증거를 획득해야 한다.

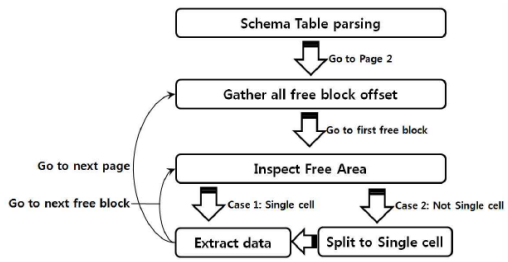

Fig. 2. SQLite recovery process of deleted data

전상준 외 4명은 SQLite 데이터베이스의 비할당 영역에 잔존하는 삭제된 레코드에 대하여 복구 기법을 제안하였다[3]. SQLite 파일에서 삭제된 영역은 세 가지로 구분될 수 있다. 먼저, 삭제된 후 다른 데이터가 덮여 쓰여 데이터의 일부가 남아있는 경우, 삭제된 영역에 하나의 레코드가 독립적인 상태로 존재하는 경우, 삭제된 영역에 레코드가 2개 이상 존재하는 경우로 구분할 수 있다.

Fig. 2는 SQLite 내부의 삭제된 데이터를 복구하는 과정을 나타낸 그림이다. 삭제된 후 다른 데이터가 덮어 쓰여 데이터의 일부가 남아있는 경우에는 Case1에 해당하고, 삭제된 영역에 하나의 레코드가 독립적인 상태로 존재하는 경우에는 Case 2에 해당한다. 그러나 삭제된 영역에 레코드가 2개 이상 존재하는 경우에는 각 영역에 관한 길이를 추측할 수 없기 때문에 복원이 불가능하다.

기존 데이터베이스 은닉과 관련된 연구는 데이터베이스 시스템을 대상으로 하는 데이터 은닉과 이에 대한 아티팩트를 분석하여 대응하는 내용을 다루고 있으며 데이터를 삭제했을 경우 복원하는 방법을 제안했다. 그러나 SQLite 데이터베이스 파일의 메타데이터를 직접 변경하여 데이터를 은닉하는 행위에 대해서는 아티팩트 기반의 탐지 기법인 기존의 연구로는 탐지가 불가능하다. 따라서 본 논문에서는 SQLite 데이터베이스 파일의 바이너리 데이터를 조작하여 정보를 은닉하는 방법과 이를 탐지하는 방법, 그리고 탐지 알고리즘과 도구를 제시한다.

III. SQLite 구조

SQLite는 모바일 기기의 애플리케이션 데이터를 저장하기 위하여 사용된다. 규모가 큰 시스템에서 사용되는 MySQL이나 Oracle과 같은 데이터베이스에 비해 작업량이 적으며, 데이터 저장 시 단일 파일

만을 사용하기 때문에 데이터베이스 파일의 구조가 상대적으로 단조롭다.

SQLite 데이터베이스 파일의 구조는 Fig. 3과 같은 형태로 나타낼 수 있다[4,5]. SQLite 데이터베이스 파일은 페이지라는 저장단위로 구성되어 있으며, 각 페이지는 일정한 크기를 가지고 있다. 가장 처음 위치하는 페이지는 헤더 페이지이고, 이후 페이지들은 B+ tree 구조로 구성된다. SQLite는 변형된 B-tree 구조인 B+ tree 구조를 사용하고 있는데 B+ tree 구조는 index 노드들을 leaf 노드를 찾아가는 경로를 제공하기 위해서만 사용한다는 특징이 있다. 헤더 페이지는 SQLite 전용 데이터베이스 파일임을 구분할 수 있는 시그니처와 페이지 크기, 데이터베이스 파일 크기 등 SQLite 데이터베이스 파일에 관련된 메타데이터가 존재하는 영역이다. 헤더 페이지 내부의 하위 부분에는 데이터베이스의 스키마에 대한 정보를 포함하고 있는 스키마 테이블이 존재한다. 테이블 이름, 필드의 타입, 이름, 개수, 변수 타입에 대한 정보가 들어있으며 각 테이블과 연관된 데이터 정보가 존재하기 때문에 GUI 뷰어에서 보여주는 테이블의 데이터와 매칭되는지 파악하는 중요한 지표가 된다. 특히 B+ tree구조에서 노드들의 시작이 되는 루트 페이지 번호를 SQL 쿼리문 바로 앞에 포함하고 있다.

본 장에서는 논문의 대상이 되는 SQLite 데이터베이스 파일 구조에 대하여 설명한다. 먼저 3.1절에서 SQLite의 헤더를 설명하고 3.2절에서 SQLite 데이터베이스 파일을 구성하는 단위인 페이지의 구조를 설명한다. 3.3절을 마지막으로 SQLite의 저장 메커니즘을 설명한다.

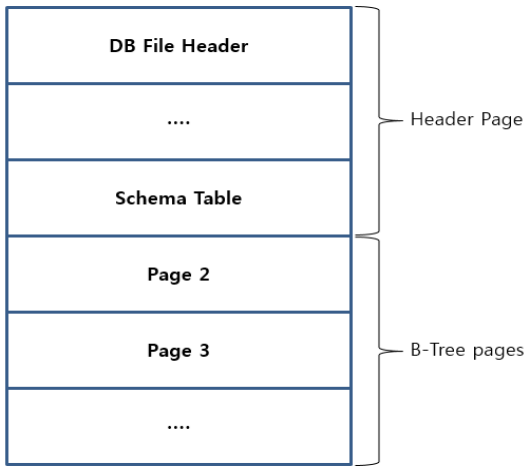

Fig. 3. SQLite database file structure

3.1 SQLite 헤더

Table. 1은 SQLite 데이터베이스 파일 내부 헤더 영역을 나타낸 표이다. 총 100바이트로 구성되어

Table. 1. SQLite database file header structure

Offset	Size	Description	Offset	Size	Description
0x0	16	Header Signature	0x28	4	Schema Cookie
0x10	2	Page Size	0x2c	4	Schema foramt number (1, 2, 3, 4)
0x12	1	Write Ver	0x30	4	Default page cache size
0x13	1	Read Ver	0x34	4	The largest root b-tree page number when in auto-vacuum or incremental-vacuum modes
0x14	1	0	0x38	4	Database text encoding
0x15	1	64	0x3c	4	User Version
0x16	1	32	0x40	4	True for incremental-vacuum mode
0x17	1	32	0x44	4	Application ID
0x18	4	File Change Counter	0x48	20	Reserved for expansion (0)
0x1c	4	Size of database file in pages	0x5c	4	Version-valid-for number
0x20	4	Page number of first freelist page	0x60	4	SQLite version number
0x24	4	Total number of freelist pages	-	-	-

있으며 오프셋 0x00에서 시그니처를 시작으로 주요 메타데이터 정보를 포함하고 있다[6].

SQLite 파일의 경우에는 정해져 있는 확장자가 없기 때문에 SQLite 전용 데이터베이스 파일인지의 여부를 알기 위해서는 헤더부분의 시그니처인 'SQLite format 3' 문구를 확인하는 작업이 필요하다. SQLite는 고정된 크기를 사용하는 페이지 구조이기 때문에 분석하는데 간편하다는 장점이 있고 페이지의 크기는 오프셋 0x10에 2바이트 크기의 빅 엔디안 형식으로 나타나 있다. 오프셋 0x1C에는 데이터베이스 파일의 크기가 4KB 단위로 명시되어 있다. SQLite의 구조가 페이지 단위로 구성되어 있기 때문에 데이터베이스 파일의 값은 페이지 용량의 배수로 표현이 가능하다. 오프셋 0x20과 0x24에는 첫 번째 freelist 페이지의 페이지 번호와 전체 freelist 페이지의 개수를 각각 4바이트씩 나타내고 있다. freelist 페이지는 사용되지 않는 페이지가 freelist에 저장이 되었다가 추후 데이터를 위한 부가적인 페이지가 필요해질 경우 재사용되는 페이지이다. 그 이외에도 Schema cookie, page cache 크기, Database text encoding 버전 등 데이터베이스 파일을 구성하는데 필요한 데이터가 수록되어 있다.

3.2 SQLite 페이지

SQLite를 구성하고 있는 페이지는 각 내부에 데이터를 포함하고 있다. Table. 2는 페이지 헤더의 정보를 나타내고 있다[6]. 페이지에서는 셀이라는 작은 저장단위에 데이터를 저장하고 있으며 맨 처음 위치하는 영역은 해당 페이지의 헤더 부분으로써 페이지에 대한 메타데이터를 포함하고 있다. 페이지의

Table. 2. interior page header structure

Offset	Size	Description
0x0	1	Page type
0x1	2	First free block offset
0x3	2	Number of cells
0x5	2	Cell content area offset
0x7	1	Fragment byte number
0x8	Cellnum * 2	Cell pointer

Table. 3. B-tree page type

Type	Value
interior index b-tree page	0x02
interior table b-tree page	0x05
leaf index b-tree page	0x0a
leaf table b-tree page	0x0d

첫 바이트에는 타입 값이 위치한다. Table. 3에 제시된 것처럼 페이지 타입은 1바이트 값이며 interior index/table 페이지, leaf index/table 페이지를 구분하는 기준이 된다. index 페이지란 페이지 내부에 실제 데이터가 아닌 데이터와 연관된 key 값들을 저장해 놓는 페이지이며, table 페이지는 실제 데이터를 포함하는 페이지이다. 오프셋 0x01에는 첫 번째 free block에 대한 오프셋이 포함되어 있어서 데이터에 연관되지 않은 영역이 시작되는 부분을 알 수 있다. 오프셋 0x03에서는 해당 페이지의 셀의 개수를 나타내고 있으며 오프셋 0x05에 처음으로 셀 데이터가 시작되는 위치를 2바이트 빅 엔디안 형태로 나타낸다. Fig. 4는 첫 번째 free block과 처음 시작되는 셀 데이터의 위치를 나타낸 그림이다. 오프셋 0x08부터 각 셀의 위치를 가리키는 포인터가 각각 2바이트 씩 연속적으로 저장되어 있다. 이 영역은 셀의 개수에 따라 길이가 유동적으로 변할 수 있는 영역이기 때문에 (셀의 개수 * 2) 가변적인 크기를 갖는 영역이다. 페이지의 데이터 영역에 위치하는 레코드는 가장 처음 들어온 레코드가 페이지 맨 밑의 영역부터 쌓이게 되는 구조로 이루어져 있으며, 셀 포인터와 셀 데이터 사이는 빈 공간으로 이루어져 있다.

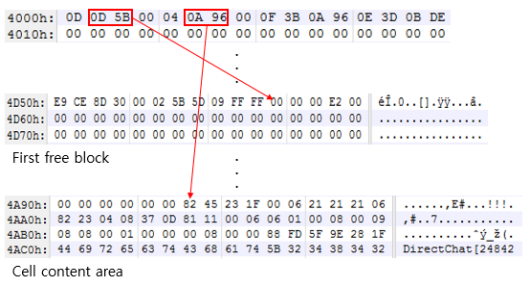

Fig. 4. First free block & Cell content area

3.3 SQLite 저장 메커니즘

Fig. 5는 B+ tree구조를 가진 SQLite 데이터베이스 파일의 데이터 참조 과정을 나타낸 그림이다[4,5]. 헤더 페이지의 스키마 테이블 영역에 존재하는 루트 노드를 기점으로 하위 노드를 참조하여 데이터를 찾아가는 일반적인 트리구조 형태를 지니고 있다. 가장 하위에 위치하는 노드는 leaf 노드이며, leaf 노드에는 실제 데이터가 저장이 된다. 루트 노드와 leaf 노드 사이에 존재하는 페이지들을 interior 페이지라고 하며 내부에는 하위 노드를 가리키는 4바이트 빅 엔디안 형태의 포인터와 2바이트 빅 엔디안 형태의 키 값이 인접하여 저장된다.

테이블 내에 쿼리를 통해 삽입된 여러 레코드들의 데이터양이 단일 페이지 용량을 넘지 않는다면 레코드 데이터는 하나의 leaf 페이지에 저장이 되지만, 만약 레코드 데이터가 단일 페이지 용량을 초과할 경우 다수의 연속된 leaf 페이지로 나누어 데이터를 저장한 뒤, interior 페이지 내부에 포인터를 저장하고 나누어진 leaf 페이지들을 참조한다. 각 페이지들은 고정된 크기로 존재하기 때문에 시작 오프셋이 단일 페이지 용량의 배수로 나타난다. 따라서 포인터를 참조하여 하위노드에 접근할 때는 (포인터 번호 * 단일 페이지 용량)에 해당하는 오프셋으로 접근해야 한다.

IV. 제안하는 데이터 은닉 기법

본 장에서는 데이터에 대한 은닉기법을 제안한다. 데이터 은닉을 위해 특정 메타데이터 영역(바이너리)에 변화를 주었을 때 발생하는 내부 데이터의 변화를 조사했다. 내부 데이터를 은닉하기 위해서는 관리자 등급의 권한이 필요한데, 실제로 다양한 공격으로 인해 어렵지 않게 관리자 권한을 탈취 당할 가능성이 존재하기 때문에 본 장에서 제안하는 데이터 은닉을 위한 전제조건이 모두 갖추어져 있다고 가정한다. 해당 연구를 위하여 데이터베이스 파일 내부 레코드를 가시적으로 볼 수 있는 GUI 기반의 도구로 SQLite Expert Personal 3, SysTools SQLite Database Recovery, DB Browser for SQLite를 사용하여 연구를 진행하였고, 바이너리 데이터 은닉을 위한 도구로 010 Editor를 이용하였으며, 실험대상파일은 안드로이드 모바일 기기에 저장되어있는 SQLite 데이터베이스 파일을 사용하였다.

SQLite 데이터베이스 파일의 메타데이터로는 헤더 페이지의 상위 100바이트 존재하는 데이터베이스 파일 헤더 영역, 테이블의 루트 노드가 존재하는 스키마 테이블 영역, 각 페이지마다 맨 처음 위치하고 있는 영역인 페이지 헤더 영역이 존재한다. SQLite 메타데이터는 SQLite의 전체 구성 정보를 담고 있

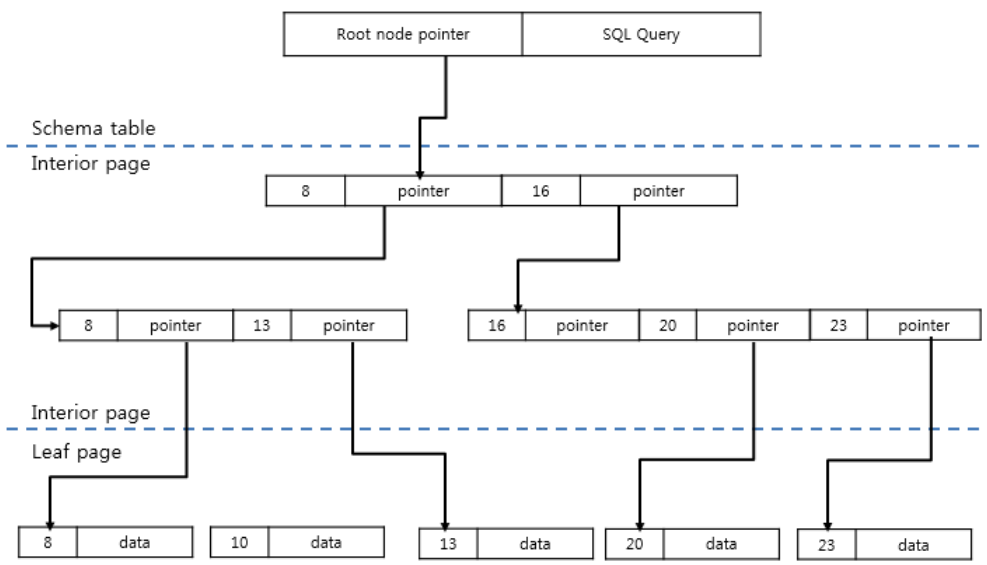

Fig. 5. B+tree structure of SQLite database file

는 데이터이며, 실제 데이터를 소유하고 있는 레코드의 저장 메커니즘이 존재한다. 따라서 메타데이터 영역에서 구성정보에 변화를 가한다면 내부 레코드를 은닉할 수 있는 가능성이 있다.

우선 SQLite의 구조를 바탕으로 모든 메타데이터 영역의 바이너리 데이터에 변화를 주어 조사를 진행하였다. 특정 영역을 변경한 경우에는 경고 메시지가 팝업 되면서 애플리케이션에서 데이터베이스 파일로 인식하지 못하게 되는 경우가 발생했다. 실제 수사에서 데이터베이스 파일을 열었을 때 정상적인 파일로 인식되지 않는다면 파일 자체에 임의의 변화를 가했다는 사실과 연결되어 인위적인 변조 행위가 드러날 것이므로, 범죄자는 경고 메시지가 발생하는 영역을 타겟팅하지 않을 것이다. 따라서 데이터베이스 파일 메타데이터의 각 영역을 인위적으로 변경하였을 때 올바른 데이터베이스 파일로 인식이 되는지에 대한 여부는 은닉을 감지하는 지표가 될 수 있다. 또한 이러한 사실을 바탕으로 애플리케이션 탐지를 우회하는 영역에 대해서는 고의적인 데이터의 저장 공간으로 사용이 가능하다는 것을 알 수 있다.

4.1 SQLite 헤더 은닉

Table. 4는 데이터베이스 파일 헤더의 메타데이터 영역에 접근하여 인위적인 데이터를 숨겼을 때 영역별로 은닉이 가능한지에 대한 여부를 구분하여 나타낸 그림이다. 시그니처, 페이지 크기, 데이터베이스 파일 크기 등 - 표시된 부분에 데이터를 숨길 경우, 기존 애플리케이션에 에러가 탐지되어 인위적인 데이터를 받아들이지 못하고 파일이 인식되지 않는 현상이 나타났으며, 에러가 탐지되지 않고 데이터를 은닉하는 것이 가능한 영역에 대해서는 O로 표시하였다. 파일로 인식되지 않는다면 해당 파일이 변조가 되었다는 것으로 간주될 수 있기 때문에, 은닉하는 공간으로 사용할 수 없다. Table. 4에서 애플리케이션 탐지를 우회할 수 있는 부분을 모두 합한 영역의 크기는 총 70바이트이므로 범죄자는 70바이트만큼 데이터를 숨길 공간을 획득하게 된다. 해당 공간에 고의적으로 URL이나 스마트폰 번호, 짧은 메시지 등을 담아서 전송에 사용한다면 70바이트 내에서 통신 교환이 가능하기 때문에 해당 공간은 얼마든지 악

Table. 4. Concealing possibility of database header area

Offset	Description	Concealing	Offset	Description	Concealing
0x0 ~ f	Header Signature	-	0x28 ~ 2b	Schema Cookie	○
0x10 ~ 11	Page Size	-	0x2c ~ 2f	Schema foramt number (1, 2, 3, 4)	○
0x12	Write Ver	○	0x30 ~ 33	Default page cache size	○
0x13	Read Ver	○	0x34 ~ 37	The largest root b-tree page number	○
0x14	0	-	0x38 ~ 3b	Database text encoding	-
0x15	64	-	0x3c ~ 3f	User Version	○
0x16	32	-	0x40 ~ 43	True for incremental-vacuum mode	○
0x17	32	-	0x44 ~ 47	Application ID	○
0x18 ~ 1b	File Change Counter	○	0x48 ~ 5b	Reserved for expansion (0)	○
0x1c ~ 1f	Size of database file in pages	-	0x5c ~ 5f	Version-valid-for number	○
0x20 ~ 23	Page number of first freelist page	○	0x60 ~ 63	SQLite version number	○
0x24 ~ 27	Total number of freelist pages	○	-	-	-

의적인 용도로 공유되는 공간이 될 수 있다.

4.2 페이지 은닉

3장에서 언급했던 스키마 테이블의 SQL 쿼리문 바로 앞에는 루트 페이지 번호를 나타내는 인덱스 값이 존재한다. 해당 값을 변경하여 다른 페이지를 참조하게 하면, 변경된 값과 상응하는 페이지와 매칭되는 테이블 정보가 나타난다. 인위적으로 페이지를 생성하였을 경우에 생성된 페이지의 크기만큼 데이터베이스 파일의 크기가 증가되므로 헤더영역의 데이터베이스 크기 영역을 추가한 페이지의 용량만큼 더해주면 뷰어에 정상적으로 인식 된다. Fig. 6은 인위적인 페이지를 생성하여 데이터를 참조시킨 그림이다. 'WHATSUP', 'HELLO', '0000000PASCAL', 'MYNAMEISTOMY' 총 4개 임의의 데이터를 저장하여 'chat_rooms' 테이블의 루트 노드 인덱스를 인위적인 페이지 번호인 0x18로 참조하게 하였을 때, Fig. 6과 같이 정상적인 파일로 인식되어 기존 페이지가 은닉되고 해당 데이터가 출력되는 현상이 나타났다. 이러한 방식으로 범죄자는 원하는 데이터를 삽입한 인위적인 페이지로 기존 페이지에 대한 은닉을 시도할 가능성이 있다.

4.3 레코드 은닉

페이지마다 실제 레코드를 담고 있는 셀의 개수,

RecNo	_id	id	type	members
Click here to define a filter				
1	34	150562382427146	DirectChat	[100000000]
2	35	150621812303903	DirectChat	[24842834]
3	34	151623940183225	DirectChat	[0000000]
4	37	152599773556817	DirectChat	[180729892]

RecNo	_id	id	type	members
Click here to define a filter				
1	34	150562382427146	DirectChat	[100000000]
2	35	150621812303903	DirectChat	[24842834]
3	34	151623940183225	DirectChat	[0000000]

Fig. 7. Concealed interior record of SQLite database file

셀의 위치를 가리키는 포인터, 처음 레코드가 위치하는 영역의 오프셋 주소 등이 존재하는 페이지 헤더영역이 존재한다. 오프셋 0x03에 위치하고 있는 셀의 개수는 해당 페이지 내부에 들어있는 셀의 개수를 의미한다. 해당 값을 변경하였을 경우, 실제 데이터는 그대로 존재하지만 가시적으로 볼 수 있는 레코드의 개수가 변경된다. Fig. 7은 페이지의 헤더 부분에서 셀의 개수 영역을 변경시키기 전(위)과 변경하고 난 후(아래)의 그림이다. 레코드의 총 개수가 4개였던 기존의 테이블에서 셀의 개수 데이터를 3개로 변경하였을 때 가장 나중에 삽입된 레코드부터 뷰어에서 은닉되는 현상이 발생했다. 또한, 셀 포인터 영역에서 각각의 포인터의 순서를 변경했을 때 레코드의 순서 또한 변경이 가능했을 뿐더러, 원하는 레코드의 포인터를 삭제했을 때 해당 레코드를 은닉하는 행위가 가능했다.

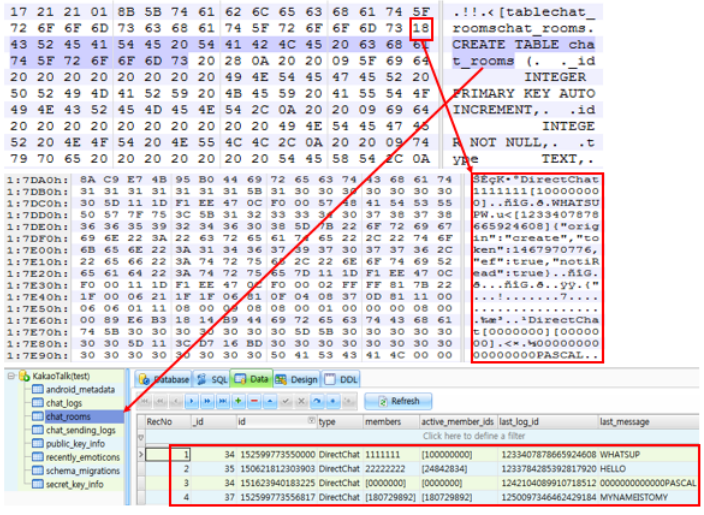

Fig. 6. Concealing data by adding artificial pages

V. 탐지 기법

수사현장에서 획득한 데이터베이스 파일에 4장에서 제시한 방법으로 은닉이 감행되었을 경우 해당 기법에 대응하기 위한 탐지 방안이 필요하다. 본 장에서는 제안된 은닉을 탐지하는 방안과 그에 따른 알고리즘을 제시한다. Table. 4의 'Concealing' 영역이 아닌 부분에 범죄자가 은닉을 시도하였다면, 애플리케이션에서 은닉행위가 탐지되어 고의적으로 접근하였다는 것을 의심할 수 있기 때문에 본 장에서는 애플리케이션의 탐지를 우회하는 파일에 대해서만 탐지를 진행하였다.

SQLite에서 은닉된 데이터를 탐지하는 경우에는 우선적으로 데이터베이스 파일의 헤더 부분에서 페이지의 크기를 알아내는 것이 중요하다. 그 후, 페이지의 단위 크기만큼 데이터를 읽어 들이고 탐색 하면서 내부 데이터에 대한 은닉 여부 조사를 진행한다.

5.1 SQLite 헤더 은닉 탐지

4장에서 데이터베이스 헤더영역에 70바이트만큼의 데이터를 은닉할 수 있음을 보였다. 은닉된 영역에 대해서는 애플리케이션의 에러 탐지를 우회할 수 있기 때문에 100바이트 전체 헤더영역을 파싱한 뒤, 70바이트 내부를 디스플레이 하여 인위적으로 숨겨진 데이터의 여부를 확인한다.

5.2 페이지 은닉 탐지

다음으로 스키마 테이블의 SQL 쿼리문 앞 루트 노드를 가리키는 포인터를 참조하여 각 테이블의 데이터를 탐색한다. Fig. 8은 SQL 쿼리문 영역 앞의 루트 노드 인덱스와 그에 상응하는 테이블을 나타낸 그림이다. 루트 노드 인덱스는 'CREATE TABLE' 문구 바로 앞의 1바이트 정수 형태로 저장되어 있다. 해당 테이블의 고유 인덱스를 바꿔 다른 테이블을 참조하게 하는 것만으로도 데이터를 은닉하는 작업이 가능했다. 하지만 인덱스가 중복 사용되었을 가능성이 있기 때문에 중복 여부를 확인해야 한다. 중복되는 페이지가 없다면 페이지를 인위적으로 생성하여 데이터 은닉을 시도하는 경우를 고려할 수 있는데, 이러한 경우에는 모든 데이터를 페이지 크기의 단위로 읽어 들인 다음, 페이지와 인덱스를 각각 매칭시키면서 일대일 대응이 되는지 확인하는 작업이 필요하다.

5.3 레코드 은닉 탐지

인위적인 페이지에 대한 판별 이후, 인덱스를 참조하여 테이블의 내부 데이터 영역에 접근한 뒤 셀 데이터의 은닉 여부를 판단해야 한다. 우선 페이지 타입을 파싱하여 0x0D의 값을 가지는 페이지인 leaf 페이지와 0x05의 경우인 interior 페이지에 대해 레코드의 은닉 여부를 확인한다. interior 페이지의 경우에는 페이지 내부에 포인터를 포함하고

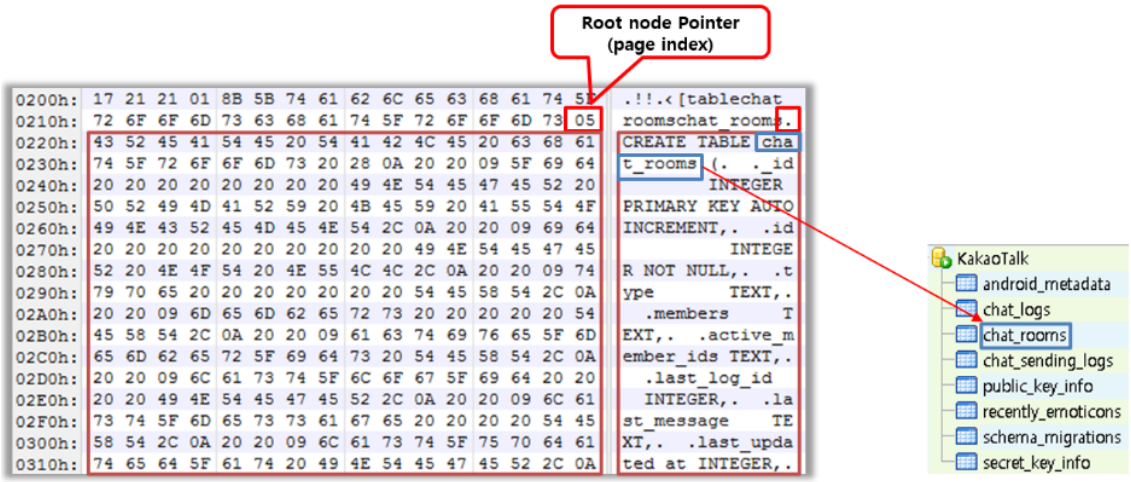

Fig. 8. Page index of root node

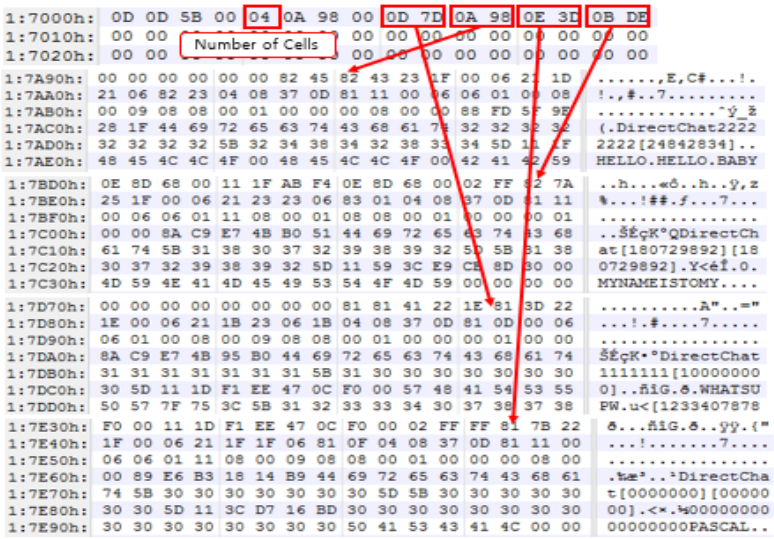

Fig. 9. Number of Cells & Cell pointers in page

있기 때문에 해당 포인터를 참조하여 leaf 페이지로 접근한 뒤 셀 데이터의 은닉 탐지를 시작한다.

leaf 페이지에서 은닉에 활용되는 메타데이터 영역은 오프셋 0x03에 해당하는 셀의 개수와 오프셋 0x08에 위치하는 셀 포인터이다. Fig. 9는 페이지 내의 셀 포인터와 셀의 개수를 나타낸 그림이다. 셀 포인터는 0x08부터 연속적으로 존재하고 있는데 Fig. 9에 제시된 포인터는 0xD7D, 0xA98, 0xE3D, 0xBDE 총 4개이며 각각 해당 위치를 참조하고 있다. 셀의 개수와 셀 포인터의 개수가 서로 다르다면 고의적으로 은닉을 시도했다는 사실을 알 수 있다. 그러나 셀의 개수와 셀 포인터의 개수가 같다고 할지라도 두 부분에 대한 은닉을 동시에 시도했을 가능성도 배제할 수 없기 때문에 반드시 내부 셀 데이터가 셀의 개수만큼 존재하는지 확인하는 작업이 필요하며, 각 셀 포인터를 통해 레코드 데이터 영역

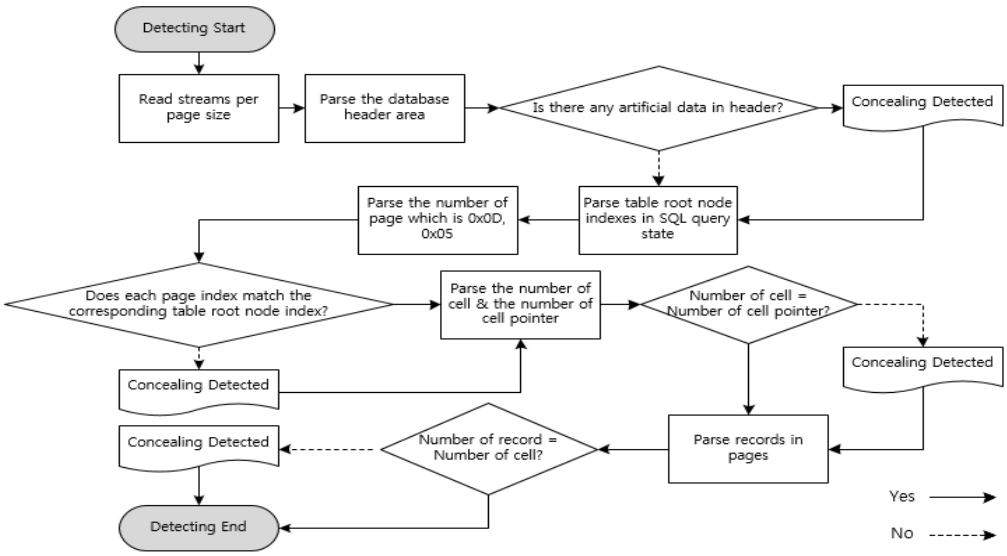

Fig. 10. Detecting algorithm for concealed database file

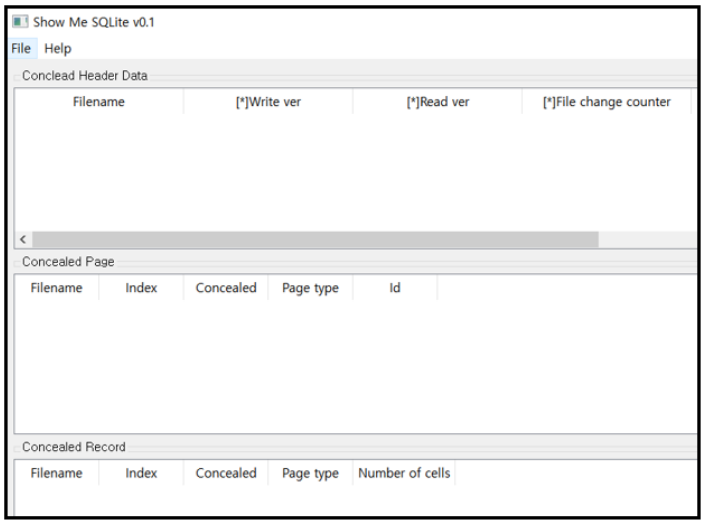

Fig. 11. First screen of Show Me SQLite

을 파싱하여 레코드의 은닉여부를 파악해야한다. 해당 탐지까지 진행하였을 때 아무런 흔적도 발견되지 않았다면 은닉이 되지 않은 원본 파일이라고 판단한 후 탐지를 종료한다.

5.4 탐지 알고리즘

Fig. 10은 본 절에서 제시하는 탐지 기법을 기반으로 한 알고리즘을 제시한 그림이다. 우선 'Detecting start'를 시작으로 하여 페이지 단위로 DB 파일을 순차적으로 읽어 들인 후 헤더 페이지의 70바이트에 대한 은닉 여부를 판단한다. 이후 은닉된 페이지가 있는지 조사하고 각 페이지 내부 레코드에 대한 은닉 여부를 분석한 뒤 탐지를 종료한다. 은닉의 흔적이 발견되었다면 'Concealing Detected' 라는 문구를 출력하며, 아무런 이상 없이 알고리즘을 통과하였다면 'Detecting End'라는 메시지와 함께 탐지를 종료하게 된다. 비교문에서는 'yes'를 실선으로, 'No'를 점선으로 표기하여 알고리즘을 제안하였다.

5.5 탐지 도구

본 절에서는 5.4절에서 언급한 알고리즘을 기반으로 제작한 도구를 설명한다. 도구의 이름은 Show Me SQLite로 명명하였다. Fig. 11은 Show Me SQLite의 초기 화면이다. 좌측 상단의 'File' 메뉴는 분석할 파일을 탐색하여 로드하는 기능을 가지고 있으며, 조작 및 내용을 쉽게 이해하기 위해 가이드를 제시하는 'Help' 메뉴가 포함되었다.

우선 파일이 적재되면, Fig. 10에서 제시한 flow를 순서대로 진행하며 파일을 분석한다. 헤더 부분에서는 은닉이 가능했던 70바이트의 공간에 대해 각 내용을 출력하며 인위적으로 데이터를 삽입한 흔적이 보였다면 해당 내용을 큰 빨간 글씨로 출력한다. 페이지 탐지 부분에서는 은닉이 탐지된 페이지에 대하여 해당 파일 이름, 페이지 인덱스, 은닉 여부, 해당 페이지 타입, 레코드 번호, 레코드 내용을 차례로 출력하고 있다. 마지막 레코드 탐지 부분에서는 셀 개수와 셀 포인터의 개수가 다른 페이지를 대상으로 탐지하여 은닉된 레코드를 탐지하고 실제 레코드를 파싱하여 출력하는 기능을 갖는다.

Fig. 12는 인위적인 데이터를 삽입한 'message (Page and record concealed).db' 파일을 본 절에서 제시한 도구로 분석한 화면이다. 헤더의 'Write ver', 'File change counter' 영역에 인위적으로 삽입한 데이터가 빨간색 큰 글씨로 표시되는 것을 확인할 수 있으며, 페이지와 레코드 탐지 부분에서도 은닉된 데이터를 출력해주는 것을 확인할 수 있다.

VI. 결론 및 향후 연구

SQLite는 모바일을 비롯한 임베디드 기기나 경량 기기에 자주 사용되어 사용빈도가 날로 높아지고

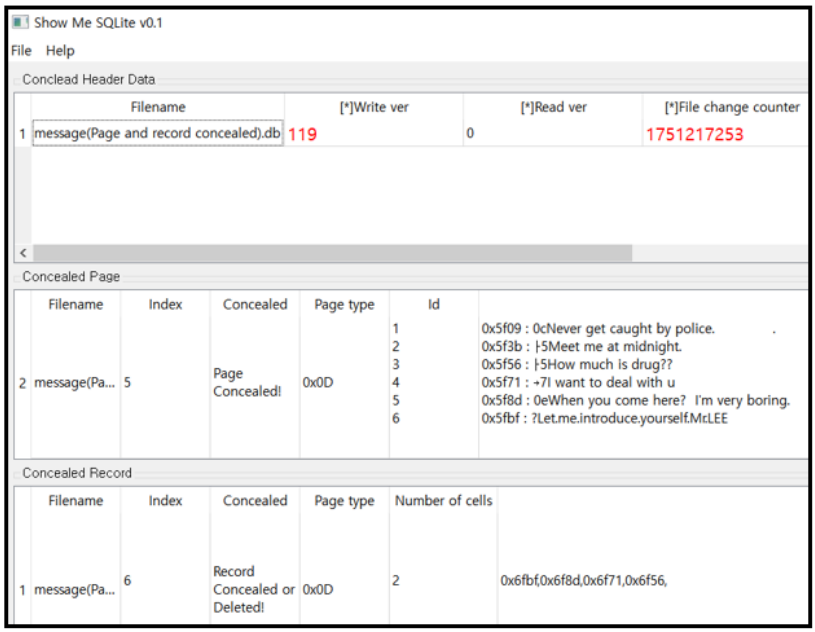

Fig. 12. Operation screen of Show Me SQLite

있다. 본 논문에서는 SQLite 데이터베이스 파일 내부에 데이터를 은닉하는 행위가 가능했다. 수사현장에서 압수 수색 영장을 받고 데이터베이스를 압수하여 분석할 때 데이터가 뷰어에서는 보이지 않게 은닉이 되었다면, 은닉의 진위를 판단하는 것이 쉽지 않다. 데이터베이스 자체에서 은닉의 여부를 판단할 수 없다면 직접 내부 데이터를 이미징 하여 파일에 대한 데이터를 일일이 분석해야하기 때문에 제시된 탐지 알고리즘은 의미가 있다. 또한 현재까지 SQLite 데이터베이스 파일에 대한 은닉 탐지에 관한 연구가 되어있지 않기 때문에 본 논문에서 제안하는 알고리즘을 SQLite와 관련된 사건에 활용할 수 있다.

본 논문에서 제안하는 도구는 SQLite 데이터베이스 파일이 법정에서 효력을 갖게 하는 매개체 역할을 한다. 은닉의 여부를 나타내는 기능, 은닉이 탐지된 부분의 영역을 표시해주는 기능, 그리고 SQLite 데이터베이스의 메타데이터 영역을 파싱하여 해당 SQLite에 대한 정보도 같이 보여주는 기능을 가지고 있기 때문에 SQLite 관련 포렌식에 많은 도움이 될 수 있다. 이후 국내 데이터베이스 포렌식 시장에서 많이 사용되고 있는 서버형태의 데이터베이스인 Oracle이나 MySQL 등 주요 데이터베이스에 대한 다양한 데이터 은닉 기법을 연구한 뒤, 그에 대응하는 탐지 기법을 개발하는 연구가 필요하다.

References

[1] Keungi Lee, Seokhee Lee, and Sangjin Lee, "A Study on Detection of Covert Database System", Korean Institute of Broadcast and Media Engineers, pp. 197-200, Feb. 2008.

[2] Jongmin Jin, "A Study on System Artifacts Investigation Technique for Identifying Database Concealment", Ph. M. Thesis, Department of Mathmatical Information Science, Seoul National University, Feb. 2017.

[3] Sangjun Jeon, Keunduck Byun, Jewan Bang, Guengi Lee, Sangjin Lee, "The Method of Recovery for Deleted Record in the Unallocated Space of SQLite Database", Masters Degree. Thesis, Journal of the Korea Institute of Information Security & Cryptology, Vol. 21, No. 3, pp. 143-154, Jun. 2011.

[4] Joonhee Lee, Mincheol Shin, Yongil Jang, and Sanghyun Park, "A Novel

Recovery Scheme for SQLite Based on Logical Logging", Journal of KIIT, Vol. 12, No. 11, pp. 181-192, Nov. 2014.
[5] SangJun Jeon, Jewan Bang, KeunDuck Byun, GuenGi Lee, and SangJin Lee, "The Method of Recovery for Deleted Record in the Unallocated Space of SQLite Database", Journal of Korea Institute of Information Security and Cryptology, Vol. 21, No. 3, pp. 143-154, Jun. 2011.
[6] Gyu-Won Lee, Seung-Jei Yang, Hyun-Uk Hwang, Kibom Kim, Taejoo Chang, and Ki-Wook Sohn, "A Recovery Scheme for the Deleted Overflow Data in SQLite Database", Journal of KIIT, Vol. 10, No. 11, pp. 143-153, Nov. 2012.

〈저자소개〉

이 재 형 (Jae-hyoung Lee) 학생회원
2016년 2월: 국민대학교 수학과 졸업
2016년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 디지털 포렌식, 정보보호

조 재 형 (Jaehyung Cho) 학생회원
2015년 8월: 국민대학교 수학과 졸업
2015년 9월~2017년 8월: 국민대학교 금융정보보안학과 석사
2017년 9월~현재: 국민대학교 금융정보보안학과 박사과정
〈관심분야〉 정보보호, 디지털 포렌식, 암호 알고리즘

홍 기 원 (Kiwon Hong) 학생회원
2016년 2월: 국민대학교 수학과 졸업
2016년 3월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 디지털 포렌식, 정보보호

김 종 성 (Jongsung Kim) 종신회원
2000년 8월/2002년 8월: 고려대학교 수학 학사/이학석사
2006년 11월: K.U.Leuven, ESAT/SCD-COSIC 정보보호 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2007년 3월~2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월~2017년 2월: 국민대학교 수학과 부교수
2014년 3월~현재: 국민대학교 일반대학원 금융정보보안학과 부교수
2017년 3월~현재: 국민대학교 정보보안암호수학과 부교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식