Digital Investigation 24 (2018) S121–S130

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2018 Europe — Proceedings of the Fifth Annual DFRWS Europe

A standardized corpus for SQLite database forensics

Sebastian Nemetz, Sven Schmitt*, Felix Freiling

Computer Science Department, Friedrich-Alexander-University Erlangen-Nuremberg (FAU), Germany

Keywords:
Digital forensics
Database forensics
Forensic corpora
SQLite3

A B S T R A C T

An increasing number of programs like browsers or smartphone apps are using SQLite3 databases to store application data. In many cases, such data is of high value during a forensic investigation. Therefore, various tools have been developed that claim to support rigorous forensic analysis of SQLite database files, claims that are not supported by appropriate evidence. We present a standardized corpus of SQLite files that can be used to evaluate and benchmark analysis methods and tools. The corpus contains databases which use special features of the SQLite file format or contain potential pitfalls to detect errors in forensic programs. We apply our corpus to a set of six available tools and evaluate their strengths and weaknesses. In particular, we show that none of these tools can reliably handle all corner cases of the SQLite3 format.

Introduction

SQLite has evolved into one of the most widely used database management systems (DBMS) in the world (SQLite Online Documentation, 2016a). Originally targeting the domain of embedded devices, its standalone DBMS implementation with comprehensive support for SQL made it a very popular storage engine for messaging applications and browsers most prominently (but not only) on mobile devices. The widespread acceptance of SQLite has lead to the situation where digital investigations regularly require the forensic analysis of data stored in SQLite databases.

Motivation

As in many other areas of forensic computing, the first tools for the forensic analysis of SQLite files were created by practicioners in response to urgent needs. Commercial tool vendors contributed their own share of responses to imminent market needs. Little, however, is known about the level of reliability and scrutiny with which these tools are able to analyze general SQLite files. A detailed knowledge of the strengths and weaknesses of tools is, however, required within a forensic analysis.

On the importance of forensic corpora

With respect to forensic tool development and tool testing, Garfinkel et al. (2009) argued that digital forensic science should strive for reproducibility of tool evaluations by adopting performance requirements and standards for forensic software. Basic concepts to realize this claim are the development and use of standardized corpora, which have been influential and beneficial in many other areas of computer security such as intrusion detection systems (Lippmann et al., 2000).

Without standardized corpora, contributions often rely on self-collected, personal and rather small sets of data, sometimes only created for a very particular purpose. Such data sets are typically not available to the public. Other researchers can neither reproduce, verify, nor build upon the test cases and results presented. There is no way to objectively compare the strengths and weaknesses of different tools with each other, without a test data set that is standardized and publicly available. In short, research efforts and results that are performed on individual data sets are not comparable and therefore less useful. Such efforts have limited impact and are basically lost for future research.

A more scientific approach is to make data sets available to the public. This way, they can be used by different people for different purposes over time. Exemplary purposes are testing and evaluating existing tools or even developing new ones. With a publicly available test dataset, new approaches and algorithms for forensic analysis can directly be compared to existing tools. Possible improvements, such as less ressource consumption, improved performance or better analysis results, can be measured and made available to everyone. The acceptance of a tool by its users has no longer to be based on the reputation of its creator(s), but can be derived from publicly available and reproducible test results. The users can have increased confidence in the tool and review its quality. This way, public review of software (tools) can make them

* Corresponding author.
E-mail addresses: sebastian.nemetz@posteo.de (S. Nemetz), sven.schmitt@cs.fau.de (S. Schmitt), felix.freiling@cs.fau.de (F. Freiling).

https://doi.org/10.1016/j.diin.2018.01.015

more robust, more reliable and more trustworthy. This is what we should request and claim for tools, especially when used in forensic investigations. We give an overview about existing corpora in the following section.

Related work

Many standardized corpora in digital forensics exist today (Yannikos et al., 2014; Digital Corpora, 2009) and are publicly available for download. Starting with a study of disk sanitization practices, Garfinkel (Garfinkel and Abhi, 2003) published a sequence of papers relating to the importance and benefits of standardized corpora in forensic computing (Garfinkel, 2006, 2007; Garfinkel et al., 2009; Zarate et al., 2014). In 1998, Garfinkel started to buy and collect hard drives from the second hand markets and thereby established the *Real Data Corpus* that, today, comprises more than 35 TB of data. This corpus aims at providing general hard disk images and a high variety of file types to test digital repository architectures (Woods et al., 2011), general disk forensics tools (Guo et al., 2009; Garfinkel, 2012a) and automation processes (Garfinkel, 2009, 2012b). The Real Data Corpus was, however, not designed to specifically test SQLite analysis tools. A comprehensive overview of available datasets with relevance to digital forensics is presented by Grajeda et al. (2017)

Contributions

In this paper, we introduce the – to our knowledge – first forensic corpus specific to the SQLite DBMS. With *SQLite* we actually refer to SQLite Version 3 (SQLite Online Documentation, 2004), which was released in 2004 and is still the most recent and widely spread version. More specifically, we make the following contributions:

- We dissect the SQLite3 file format and describe characteristics that are important with regard to forensic analysis and analysis tool robustness. While some may appear exceptional, all cases do actually fully comply with the definition of the official file format (SQLite Online Documentation, 2016b).
- Based on the file format analysis, we develop a forensic corpus of SQLite database files focussing on the inner structures of the database file format. The corpus consists of 77 databases grouped into 5 categories according to their peculiarities. Along with the database files, the corpus also comprises a technical documentation about the creation of every single database and its contents (ground truth).
- We apply the newly created corpus to a set of forensic software able to process SQLite3 database files. We thereby evaluate strengths and weaknesses of existing tools and show that none of the available tools can handle all corner cases reliably.

We make this corpus accessible for further research by making it available online at the following URL: https://faui1-files.cs.fau.de/public/sqlite-forensic-corpus/.

Outline of this paper

The paper is structured as follows: First, we give an overview about general characteristics and categorization of corpora (Section: Background on forensic corpora). Subsequently, we introduce the SQLite Forensic Corpus (Section: Introducing the SQLite Forensic Corpus) and present its structure as well as accompanying metadata. Details about the individual contents and peculiarities for each of the 77 database files comprised in the corpus are then provided (Section: Databases in the SQLite Forensic Corpus). These define different scenarios against which any SQLite processing tool can be tested. We evaluate several SQLite specific tools against the newly introduced corpus and discuss the test results (Section: Evaluation). Finally, we summarize the paper and conclude (Section: Conclusion).

Background on forensic corpora

We briefly recall general categories and taxonomies of digital forensic corpora within which we embed our work.

Categorization of corpora

Due to their varying nature, corpora in digital forensics fall into different categories, such as disk images, memory images, network packets and files. Corpora of disk images are built with a focus on file system forensic analysis and carving. Memory images are targeting forensic analysis of data structures from operating systems in main memory (RAM). Collections of network packets make the structures and characteristics of communication protocols available, whereas corpora containing files focus on the structures of different application file formats. In this paper, we take a closer look to corpora containing files. Obviously, a corpus specific to SQLite represents a corpus containing database files that conform to the SQLite file format.

Garfinkel describes seven different criteria, that are important when creating forensic corpora (Garfinkel, 2007). They shall help to measure the benefit and usefulness of a corpus. Accordingly, a corpus shall feature the following characteristics.

1. *Representative*: of data encountered during the course of criminal investigations, civil litigation, and intelligence operations.
2. *Complex*: with intertwined information from many sources. Data objects should have a wide range of sizes. The data should be in many human languages.
3. *Heterogeneous*: generated using a range of computer systems and usage patterns.
4. *Annotated*: so that new algorithms can be readily validated against existing ones.
5. *Available*: to researchers operating in an unclassified environment.
6. *Distributed*: in open file formats. Tools should be supplied with the corpora to allow for easy manipulation of the data.
7. *Maintained*: computer systems are constantly changing and evolving. A corpora needs to be routinely augmented with new information or else its interest soon becomes largely historical.

Taxonomy on corpora sensitivity

Additionally, Garfinkel et al. (2009) defined a taxonomy regarding the sensitivity of corpora as follows.

- *Test Data*: Such data sets are specifically created for the purpose of testing and thus do not contain sensitive data. They can be distributed over the Internet without restrictions.
- *Sampled Data*: Data sets that were extracted out of a larger data source, e.g., the Internet. However it might be difficult to ensure that none of the sampled files has legal restrictions regarding redistribution.
- *Realistic Data*: The contents of these data sets can be encountered in real life situations. They can, for example, be collected after installation of software or after having simulated some activities. Distribution is possible from the perspective of privacy, but might be hindered by copyright restrictions.

- *Real and Restricted Data*: Such data sets contain real data that have been created through real use of soft- and hardware. Distribution over the internet is not allowed from both a privacy and a copyright point of view.
- *Real but Unrestricted*: Data sets containing real data that have been publicly released, e.g., in court cases, or are publicly available, like pictures on *Flickr* or profiles on *Facebook*.

In the next section, we introduce and characterize the SQLite Forensic Corpus according to the given categories as well as the sensitivity taxonomy.

Introducing the SQLite forensic corpus

Due to the importance of the SQLite databases in digital forensics, we believe that it is worth dedicating a separate, stand-alone corpus to this area and thereby enabling to focus on technical details of the underlying file format. We now present the result of our research, namely the *SQLite Forensic Corpus* (Version 1.0). In this and the following section, we describe its characteristics and contents in more detail.

Corpus categorization

The SQLite Forensic Corpus falls into the category of corpora containing (logical) files. It focuses on the structures of the SQLite3 file format (SQLite Online Documentation, 2016b). Therefore, the corpus contains specifically crafted database files that were created through use of the SQLite DBMS. In the following, we characterize the corpus with respect to the seven criteria previously discussed (Section: Categorization of corpora). These may give an indication for its usefulness.

1. *Representative*: A single file format defines all structures internal to SQLite databases. Ultimately, they differ in the amount of elements (mainly tables), in the amount of columns and rows as well as in the actual contents. The corpus therefore comprises databases that vary in their settings, internal structures and contents.
2. *Complex*: Every database in the corups, along with all of its contents, is stored in a single file. The size of the databases varies from 2048 bytes through 286720 bytes. We did not include larger databases in order to minimize the corpus size for the ease of distribution. However, every database in the corpus was specifically crafted. Alltogether they contain a variety of different structures and contents that do appear in real databases. For example, all encodings supported by SQLite are also represented in the corpus.
3. *Heterogeneous*: The format and internal structure of a database is the same, regardless of the system it is generated on. That is why databases can easily be copied between different systems, independent of the operating system and hardware architecture (endianness) in use. The files in the corpus have all been generated using one and the same system. But because of the given interchangeability, we do think that possible differences between databases, that are generated on heterogeneous systems, can be disregarded.
4. *Annotated*: We make an extended effort to document the contents of the corpus. The accompanying metadata about the contained database files not only documents the ground truth, but allows for regenerating every single database file and/or adapting its contents. We include all relevant SQL statements that were used to initially build the corpus.
5. *Available*: The corpus does not contain files that are restricted in any way. Thus, we donate it into the public domain and make it available to anybody worldwide.
6. *Distributed*: Obviously, the files in the corpus comply to the SQLite file format, which is an open and, beyond that, very well documented format. We provide accompanying metadata about the corpus in the form of text-based files containing SQL and XML contents. Both formats are very well-known and a multitude of tools exist to work with, process and manipulate these formats.
7. *Maintained*: The corpus is subject to versioning. A unique version number is assigned to the state of the corpus upon each release, starting with Version 1.0. Additionally, we do provide all SQL statements necessary to reproduce the entire corpus. Both facts facilitate ongoing maintenance of the corpus with respect to future development of SQLite. There is ongoing work on different extensions for the SQLite Forensic Corpus that we plan to release in future versions.

Corpus sensitivity

Regarding the sensitivity of the corpus, it exclusively contains test data. The databases have been synthetically crafted to define different test scenarios. These are based either on different contents in the databases or on different values of internal structures, or both. A more detailed insight into the contents of the files is given later on (Section: Databases in the SQLite Forensic Corpus).

A corpus comprising test data has several advantages that can be leveraged for tool testing and development. One advantage is the existing knowledge about the ground truth, which is documented within the metadata accompanying the corpus. This allows for measuring the performance and quality of tested tools and for comparing test results of different tools with each other. Another advantage is that the distribution of the corpus is not restricted in any way. It can be freely accessed and shared by people all over the world.

In preparation of the SQLite Forensic Corpus, we collected a set of sampled data with the help of scripts automating the download of different SQLite sources over the Internet. As of today, this set contains over 1.600 SQLite database files that were publicly available since 2013. Unfortunately we cannot publically share this data set from a privacy and copyright point of view. However, although we distribute synthetically created test data, the corpus was enhanced with peculiarities that we previously derived from the sampled data set. So, we explicitely include potential pitfalls as well as unusual structures and values encountered in real data. Thereby, we raise the sensitivity of the SQLite Forensic Corpus.

Structure of the SQLite forensic corpus

The SQLite Forensic Corpus comprises a total of 77 databases. Every single database file was specifically crafted and has at least one peculiarity regarding its contents and/or internal structure. Alltogether, processing the databases in the corpus means testing a tool against a multitude of different scenarios that are to be checked. For the ease of understanding and describing the peculiarities present in the corpus, we group the databases into 14 folders assigned to 5 categories. The structure and categorization is depicted in Table 1, while a detailed description of each category is given in the following Section (Section: Databases in the SQLite Forensic Corpus).

Every folder holds one or more database files that are part of a single category. Separate files with metadata about every database are additionally provided (see Section: Metadata accompanying the SQLite Forensic Corpus). A file named `sha256sums.txt` contains SHA256 hashsums of all files in the corpus in order to allow for verifying the integrity of all databases and their metadata. The folders themselves are numbered with a hexadecimal notation from `01` to `0E`. Databases within each folder are also numbered, in

this case with a decimal notation. The reason for the hexadecimal numbering of folders is the possibility to easily distinguish future extensions of the corpus from the basic database files initially released in the realm of this work. The names of folders belonging to the latter all start with a value of zero (`01-0E`). Future extensions of the corpus are encouraged to chose a different value for the leading character in the folder names.

It is worth noting, that SQLite provides a setting called `secure_delete`. This setting determines whether database contents are overwritten upon deletion or not. If the setting is inactive, forensic traces may remain in a database, even after the deletion of contents (SQLite Online Documentation, 2018).

Metadata accompanying the SQLite forensic corpus

An important issue for the quality of a forensic corpus is the availability of documentation about the provided data. For reasons of traceability and reproducibility, we provide two files storing metadata along with every single database file.

One of the two additional files provides the SQL statements necessary to create the associated database. They are stored in the order of execution when the corpus was initially created. The files are named according to the schema `[folder]-[number].sql`. The SQL file enables anybody working with the corpus to see which data can possibly be found in or recovered from a specific database file and by what commands it was created.

The other additional file stores information about the database contents in an XML format. We thereby provide an interface for other programs in order to automate work with the SQLite Forensic Corpus. The files are named according to the schema `[folder]-[number].xml`. An internationally standardized way to describe an information resource, such as a corpus, is given by the Simple Dublin Core Metadata Element Set (DCMES) (Dublin Core Metadata Initiative). It proposes the use of 15 different elements to provide general information about a resource. The elements are title, subject, description, creator, contributor, publisher, date, type, format, language, identifier, source, relation, coverage and rights. Although the use of single elements is optional, we set most of them for every database file in the corpus. An example of how these XML files are structured is given in Listing 1.

The root element of each XML file is represented by the `database` tag. At the level below, some metainformation about the entire database is given within the `meta` tag, while the 15 elements proposed by the DCMES are enclosed by the `description` tag. The remaining part of the XML file consists of information describing every database element (`elements` tag). A database element is exactly one of the following: table, index, trigger or view. Each element description comprises three different sections: `meta`, `sql` and `entries`.

The first section (`meta`) holds general information about the element, such as type and name. The second section (`sql`) describes the schema of the element, which is derived from the corresponding SQL CREATE statement, including eventual SQL constraints. This section is composed of a definition for every column, enclosed by the `columnDefinition` tag. The third section (`entries` tag) describes the entries (rows) of the element by its columns.

Table 1
Categories and folders of the corpus.

Category	Folder	Subject
Keywords & identifiers	01	Weird table names
	02	Encapsulated column defs
	03	SQL keywords & constraints
Encodings	04	UTF Character sets
Database elements	05	Trigger, views and indices
	06	Virtual & temporary tables
Tree & page structures	07	Fragmented contents
	08	Reserved bytes per page
	09	Pointer-map page
Deleted & overwritten contents	0A	Deleted tables
	0B	Overwritten tables
	0C	Deleted records
	0D	Overwritten records
	0E	Deleted overflow pages

Listing 1. Example of a descriptive XML file.

```
<?xml version='1.0' encoding='UTF-8'?>
<database>
  <meta>
    <tables>1</tables>
    <indices>0</indices>
    <views>0</views>
    <triggers>0</triggers>
    <entries>2</entries>
  </meta>
  <description>
    <filename>01-01.db</filename>
    <dc:title>SQLite Forensic Corpus, Version 1.0
        </dc:title>
    <dc:subject>Weird table names</dc:subject>
    [...]
  </description>
  <element>
    <meta>
      <type>table</type>
      <name>users</name>
      <reference>users</reference>
      <deleted>False</deleted>
      <columns>3</columns>
      <rowsTotal>2</rowsTotal>
      <rowsAlive>1</rowsAlive>
      <rowsDeleted>1</rowsDeleted>
    </meta>
    <sql>
      <columnDefinition>
        <meta>
          <name>id</name>
          <attribute>INT UNSIGNED NOT NULL</
              attribute>
        </meta>
        <columnTypeName>INTEGER</columnTypeName>
      </columnDefinition>
      <columnDefinition>
        [...]
      </columnDefinition>
      [...]
      <tableConstraint>
        <statement>PRIMARY KEY(id)</statement>
        <flags>
          <isPrimaryKey>True</isPrimaryKey>
        </flags>
      </tableConstraint>
    </sql>
    <entries>
      <row>
        <column>
          <name>id</name>
          <content>20001</content>
        </column>
        [...]
      </row>
      <row deleted="1">
        [...]
      </row>
    </entries>
  </element>
</database>
```

Databases in the SQLite forensic corpus

The databases comprised in the corpus have all been specifically crafted. Every file is designed to set up an easily understandable example for a particular scenario. For a better understanding, we roughly introduce important structures of a SQLite database that all are candidates for different scenarios and peculiarities of the corpus.

The overall structure of a database, e.g., the amount and type of elements stored, is defined by the *database schema*. The schema is given through the set of SQL *statements* describing every single element. The general way to store an entry, or a row, in a SQLite database can be compared with storing a file in a file system. The storage area is separated into consecutive blocks of fixed size. In file systems we refer to these blocks as clusters, whereas in SQLite databases the basic blocks are called *pages*. To store any content, ultimately represented by a certain amount of bytes, as many blocks (pages) as needed are allocated. Typically pointers/offsets are used to reference blocks and to assign certain blocks to a given entity, such as a database `record` (entry/row). Metadata is stored for every single entity in order to be able to correctly retrieve the associated contents. This especially comprises information about type and length of stored contents.

When building the corpus, several peculiarities have been created by inserting unusual values for different structures internal to a database. All files still fully comply with the underlying file format and the databases are thus in a valid and consistent state. For convenience and practicality with respect to future applications of the corpus, the databases contain a single table or only few elements for the majority of scenarios. We deliberately kept them small in terms of complexity and size. The purpose of a single file is to feature a desired scenario.

For forensic tool testing, it is naturally of interest to rely on forensic traces, especially for deleted data. We therefore consider different settings of SQLite's feature called `secure_delete`: On the one hand, we work with the setting being activated when inserting contents into a database. This prevents the creation of duplicate and invalidated artifacts, e.g., caused by page reorganizations during the execution of insert statements. On the other hand, we work with the setting beeing deactivated when deleting contents, so that traces of deleted elements and records may remain in a database. We therefore deactivate the setting before executing the first SQL DROP or DELETE statement. The setting then remains deactivated for all queries that may subsequently be executed on the same database.

The default page size is 4096 bytes and the default database encoding is UTF-8. Both default settings are true for all databases in the corpus, if not stated otherwise in this section.

In the following, we describe all of the 77 database files in the corpus, grouped by category as depicted in Table 1 (Section: Structure of the SQLite Forensic Corpus).

Keywords & identifiers

Databases in this category feature weird table names, encapsulated column definitions and specific SQL keywords and constraints. Testing against this category can indicate whether a tool correctly handles SQL statements contained in a database.

Each table requires a unique name. We made up several table names containing unusual characters which are also used for other purposes in SQL statements, such as single or double quotes for encapsulation. The weird table names featured by the corpus are shown in Table 2.

Special characters, e.g., used for encapsulation, are also allowed in column definitions. These determine the names and types of a column. SQLite3 uses single quotes, double quotes and brackets as encapsulation characters. The descriptions below show the different column definitions actually applied:

```
2-01 ["name" NOT NULL,] TEXT
2-02 "] name TEXT, 'abc' TEXT [," TEXT
2-03 'name' [INTEGER, 'abc' TEXT,]
2-04 'name' TEXT"abc,"
2-05 'name' '[TEXT, ", abc" );'
2-06 'number' "INTEGER(11,0)"
2-07 "name"TEXT,surname'TEXT'
     (whitespace eliminated)
```

This category also includes databases with particularities regarding SQL keywords and constraints. For example, keywords may be part of identifiers and table constraints should – but do not need to – be comma separated. Details on the optimization used in scenario 3-01 and on the corner case applied in scenario 3-02 are documented in the SQLite Online Documentation, 2017a, 2017b. The following peculiarities are featured by the corpus:

- `3-01` table optimization `WITHOUT ROWID` applied
- `3-02` corner case `INTEGER PRIMARY KEY DESC`
- `3-03` constraint name `UNIQUE_NAME` includes keyword
- `3-04` column definition includes `UNIQUE` constraint
- `3-05` no comma used for separation of table constraints (behaviour allowed by SQLite against the official documentation)

Encoding and character sets

Databases in this category feature different text encodings. There are three encodings supported by the SQLite file format: UTF-8, UTF-16le (little endian) and UTF-16be (big endian). The encodings are exclusively applied to contents of the storage class `TEXT` and do not affect other structures. These databases also include non-latin characters. Testing against this category can indicate whether a tool correctly handles database contents with different encodings. The following peculiarities are featured by the corpus:

- `4-01` database encoding is UTF-16le
- `4-02` database encoding is UTF-16be
- `4-03` database encoding is UTF-8, Chinese characters in table name
- `4-04` database encoding is UTF-8, Latin and Chinese characters in table name
- `4-05` database encoding is UTF-16le, Unicode characters in table name and contents
- `4-06` database encoding is UTF-16be Unicode characters in table name and contents

Database elements

Databases in this category feature different types of elements. Whereas most scenarios in the corpus are based on database tables (as these are most important), these databases additionally comprise virtual and temporary tables, indices as well as trigger and views. One database with a temporary table had to be carved from RAM in order to be persisted. Testing against this category can indicate whether a tool correctly handles database elements other than regular tables. Details about the scenarios including virtual tables (6-01 through 6-03) are given in the SQLite Online Documentation, 2017c, 2017d. The following peculiarities are featured by the corpus:

5-01 validating trigger **BEFORE INSERT**, 1 entry
5-02 updating trigger **AFTER UPDATE**, 4 entries
5-03 database offering 2 views on a table
5-04 database offering 1 index on two columns of a table
6-01 virtual table using R*Tree module, 10 entries
6-02 virtual table using R*Tree module, 200 entries
6-03 virtual table using extension FTS4, 10 entries
6-04 temporary table (**CREATE TEMP TABLE**), (carved from RAM, since temp tables are not written to disk, in fact there is no difference to a regular table, except for the page size of 1024 bytes)

Tree & page structures

Databases in this category feature different scenarios regarding the internal tree and page layout of databases. Whenever SQLite needs to store contents of a certain length that do not fit on a single page, the record is split and parts of it are stored on one or more overflow pages. According to the SQLite documentation, a single byte at offset 20 in the database header indicates the existance of an unused "reserved" space at the end of each page (usually 0) (SQLite Online Documentation, 2016b). We make use of this feature and hide data within the reserved bytes at the end of every page. Files may also contain pages that do not belong to a database element, such as pages on a freelist or pointer-map pages. This is why we include a database with a pointer-map page. Testing against this category can indicate whether a tool correctly handles tree and page structures as well as fragmented contents. The following peculiarities are featured by the corpus:

7-01 overflow records due to long entry in one column
7-02 overflow records due to large payload (many columns)
7-03 free space on page after first part of split record
7-04 overflow in *sqlite_master* table (huge column names)
8-01 reserved space (strings, 16B) at the end of every page
9-01 database containing a pointer-map page (page 2)

Deleted & overwritten contents

Databases in this category feature artifacts of deleted data. The scenarios range from deleted and (partially) overwritten tables over deleted and (partially) overwritten records to deleted overflow pages/records. All arise from different sequences of the SQL statements CREATE (TABLE), INSERT (INTO), DELETE (FROM) and DROP (TABLE). Whenever desired, some new contents are inserted after having deleted others in order to (fully or partly) overwrite traces of previously deleted contents. For newly inserted entries some IDs uniquely identifying the entry may have been matched with the IDs of previously deleted entries. Testing against this category can indicate whether a tool is able to correctly recover deleted contents, as long as they are not overwritten. The following peculiarities are featured by the corpus:

A-01 create 1, insert 20, drop
A-02 create 1, insert 20, delete 20, drop
A-03 create 2, insert 10/each, drop each
A-04 create 2, insert 10/each, delete 10/1, drop each
A-05 create 2, insert 10/each, delete 10/each, drop each
B-01 create 1, insert 10, drop, create 1, insert 5
B-02 create 3, insert 10/each, drop 1, create 1, insert 5
C-01 create 1 (int cols), insert 20, delete 7
C-02 create 2 (int cols), insert 20/each, delete 5/each
C-03 create 1 (text cols), insert 20, delete 7
C-04 create 2 (text cols), insert 20/each, delete 5/each
C-05 create 2 (int/text), insert 20/each, delete 5/each
C-06 create 1 (float cols), insert 20, delete 7
C-07 create 2 (float cols), insert 20/each, delete 5/each
C-08 create 2 (float/text), insert 20/each, delete 5/each
C-09 create 1 (float/text), insert 10, delete 10
C-10 create 2 (float/text), insert 10/each, delete 10/each
D-01 create, insert 10, delete 5, insert 3
D-02 create, insert 10, delete 5, insert 5
D-03 create, insert 10, delete 5, insert 10: match 1
D-04 create, insert 10, delete 5, insert 3: match all
D-05 create, insert 10, delete 5, insert 5: match all
D-06 create, insert 10, delete 10, insert 5: match all
D-07 create 2, insert 10/each, delete 5/1, insert 5/2
D-08 create 2, insert 10/each (alt), delete 5/1, insert 5/2
E-01 records overflow like in 07-01, delete 7
E-02 records overflow like in 07-02, delete 5

Table 2
Table names of the databases in folder 01.

Database	Table name
01–01	" "
01–02	A"b"c
01–03	[
01–04	]
01–05	empty
01–06	''
01–07	A'b'c
01–08	–
01–09	#
01–10	space
01–11	%
01–12	%s%s
01–13	(
01–14	)
01–15	,
01–16	\t
01–17	\n
01–18	\v\f\r

Evaluation

After having created the corpus, we now leverage it to examine how tools behave in different scenarios analyzing the entirety of the 77 databases.

With respect to forensic analysis, there are in general two kinds of software: Tools that do not recover deleted artifacts and tools that do. On the one hand, the first set of tools simply extracts data logically present in the database. When evaluating such tools that work like a database browser or viewer, we can measure the amount of contents (database elements/entries) which are interpreted and presented in a correct way. On the other hand, the second set of tools claims to additionally recover artifacts of previously deleted contents, if present. Such tools typically carve contents from unallocated areas within a file (or file system).

Most of the databases in the SQLite Forensic Corpus offer logical contents to be extracted, especially in the first four categories (folders 01-09). However, the last category (folders 0A-0E) explicitely features deleted artifacts to be recovered (see Section: Databases in the SQLite Forensic Corpus).

Evaluated tools

We selected six different tools and evaluated them against the newly created corpus. Some of the tools are freely available, some of them are commercial. Some are to be considered database viewers, whilst others are promoted to support forensic analysis of SQLite database files, including the recovery of deleted contents. We did not perform an evaluation of the SQLite3 engine itself, since all databases in the corpus conform to the file format specification and are thus smoothly processed, e. g. by the SQLite3 command line shell. In the following, we shortly introduce the tested tools.

The software *Undark* is an open source program promoted as "a SQLite deleted and corrupted data recovery tool" (Daniels, 2017). The output of Undark comprises entries logically present in the database as well as artifacts recovered from previously deleted records. In the context of this work, we used the latest version available, which is v0.6.

The tool *SQLite Deleted Records Parser* is an open-source program written in python and promoted as script "to recover deleted entries in an SQLite database" (DeGrazia, 2017). It works like a carver and focuses on areas in which deleted records are supposed to reside. Thus, deleted records may be recovered, while no existing records are extracted/displayed. In the context of this work, we used the latest version available, which is v1.3.

SQLiteDoctor is a piece of proprietary software sold by the company *SQLabs* developed to repair and restore corrupted databases (SQLabs, 2017). In the context of this work, we used the demo of the latest version available, which is v1.3.0. In the demo version, only five records of each table are written into an output database, but this is quite enough for most scenarios of the corpus.

The proprietary software named *Stellar Phoenix Repair for SQLite* is a commercial tool sold by the company *Stellar Data Recovery* (Stellar Data Recovery, 2017). It is promoted to recover corrupted databases and to "easily recover[s] all deleted records from the database". In the context of this work, we used the demo of the latest version available, which is v1.0.0.0. With the demo version, recovered database contents can be displayed, but not saved. However, the latter is not necessary for our evaluation purposes.

The tool *SQLite Database Recovery* is a piece of proprietary software marketed by the company *SysTools* (SysTools Software, 2017). It is promoted as a tool to repair and export corrupt SQLite files. But it is not able to restore deleted records. In the context of this work, we used the demo of the latest version available, which is v1.2. With the demo version, database contents are displayed, but cannot be stored.

The proprietary software *Forensic Browser for SQLite* is developed by the company *Sanderson Forensics* (Sanderson Forensics, 2017). Forensic Browser is promoted as forensic tool to display all present data and restore deleted records. For the evaluation we used the fully licensed Version 3.1.6a.

Results grouped by category

A short summary of the most interesting results is given in the following for each category of the SQLite Forensic Corpus. Besides the textual description we give a more detailed overview of the results for every single tool and database file in separate tables. The first four categories (regular data) are depicted in Table 3, while the detailed results of the last category (deleted data) are summarize in Table 4. For lack of space, a short explanation of the signs depicted in these tables is only given at the end of Table 4.

Table 3
Analysis results for regular contents.

File: *.db	Undark	SQLite-Parser	SQLite-Doctor	Phoenix Repair	DB Recovery	Forensic Browser
01-01	–	–	✓	✓	✓	✗
01-02	–	–	✓	✓	✓	✗
01-03	–	–	✓	✓	✓	✗
01-04	–	–	✓	✓	✗	✗
01-05	–	–	✗	✗	✓	✗
01-06	–	–	✓	✓	✓	✗
01-07	–	–	✓	✓	✓	✗
01-08	–	–	✓	✓	✓	✗
01-09	–	–	✓	✓	✓	✗
01-10	–	–	✓	✓	✓	✗
01-11	–	–	✓	✓	✓	✗
01-12	–	–	✓	✓	✓	✗
01-13	–	–	✓	✗	✗	✗
01-14	–	–	✓	✓	✓	✗
01-15	–	–	✓	✓	✓	✗
01-16	–	–	✓	✓	✓	✗
01-17	–	–	✓	✓	✓	✗
01-18	–	–	✗	✗	✗	✗
02-01	–	–	✗	✗	✗	✗
02-02	–	–	✗	✗	✗	✗
02-03	–	–	✓	✗	✗	✗
02-04	–	–	✓	✗	✓	✗
02-05	–	–	✓	✗	✗	✗
02-06	–	–	✓	✗	✓	✓
02-07	–	–	✓	✗	✗	✗
03-01	✗	✓*	✗	✗	✗	✗
03-02	✓*	–	✓	✓	✓	✓
03-03	✓*	–	✓*	✓*	✓	✓
03-04	✓*	–	✓	✓*	✓	✓
03-05	✓*	–	✓	✓*	✓	✓
04-01	✓*	–	✓*	✗	✓	✓
04-02	✓*	–	✓*	✗	✓	✗
04-03	✗	–	✓	✗	✓	✓
04-04	✗	–	✓	✗	✓	✓
04-05	✗	–	✓*	✗	✓	✓
04-06	✗	–	✓*	✗	✓	✗
05-01	–	–	✓	✓	✓	✓
05-02	–	–	✓	✓	✓	✓
05-03	–	–	✓	✓	✓	✓
05-04	–	–	✓	–	✗	✓
06-01	✗	–	✓*	✗	✗	✗
06-02	✗	–	✓*	✗	✗	✗
06-03	✓*	–	✓*	✓*	✗	✓*
06-04	✗	–	✓	✓	✓	✓
07-01	✓*	–	✓*	✗	✓	✓
07-02	✓*	–	✓*	✓	✓	✓
07-03	✗	–	✓	✗	✓	✗
07-04	✓*	–	✓	✗	✓	✓
08-01	✗	✓	✗	✗	✗	✗
09-01	✓*	–	✓*	✓*	✗	✓

Table 4
Analysis results for recoverable contents.

File: *.db	Undark	SQLite-Parser	SQLite-Doctor	Phoenix Repair	DB Recovery	Forensic Browser
0A-01	✓*	✗	–	✗	–	✗
0A-02	9/20*	✓*	–	✗	–	✗
0A-03	✓*	✗	–	✗	–	✗
0A-04	15/20*	10/20*	–	✗	–	✗
0A-05	11/20*	✓*	–	✗	–	✗
0B-01	✗ ✓*	✗ –	– ✓	✗ ✓	– ✓	✗ ✗
0B-02	✗ ✓*	✗ –	– ✓*	✗ ✓	– ✓	✗ ✗
0C-01	✗ ✗	✗ –	– ✓*	✗ ✓*	– ✓	✓ ✓
0C-02	✗ ✗	✗ –	– ✓*	✗ ✓*	– ✓	✓* ✓
0C-03	✗ ✗	✓ –	– ✓*	✗ ✓	– ✓	2/7 ✓
0C-04	✗ 2/30*	✓* –	– ✓*	✗ ✓	– ✓	1/10* ✓
0C-05	✗ 1/30	✓* –	– ✓*	✗ ✓*	– ✓	✓* ✓
0C-06	✗ ✗	✗ –	– ✓*	✗ ✓	– ✓	✗ ✗
0C-07	✗ ✗	✗ –	– ✓*	✗ ✓	– ✓	✗ ✗
0C-08	✗ ✗	✓* –	– ✓*	✗ ✓	– ✓	✗ ✗
0C-09	5/10* –	✓* –	– ✓	✗ ✓	– ✓	✗ ✓
0C-10	11/20* –	✓* –	– ✓	✗ ✓	– ✓	✗ ✓
0D-01	✗ ✗	2/5* –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-02	✗ ✗	1/5* –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-03	✗ ✓*	✗ –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-04	✗ 1/8*	2/5* –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-05	✗ ✓*	✗ –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-06	1/10* ✓*	5/10* –	– ✓	✗ ✓	– ✓	✗ ✗
0D-07	✗ 16/20*	✓* –	– ✓*	✗ ✓	– ✓	✗ ✗
0D-08	✗ 16/20*	✓* –	– ✓*	✗ ✓	– ✓	✗ ✗
0E-01	3/7 12/13*	2/7 –	– ✓*	✗ ✗	– ✓	3/7 ✓
0E-02	✗ ✓*	✗ –	– ✓*	✗ ✓*	– ✓*	✗ ✓*

✓ → all elements correctly processed
✓* → some elements correctly, some wrongly processed (errors)
✗ → no element correctly processed
– → not supported by the tool
upper rows → results concerning recoverable contents
lower rows → results concerning regular contents

Keywords & identifiers

For the first folder (file names starting with `01`), the table name only containing white space characters (file `01-18`) is the one that most programs do not display correctly. All window-based programs print a wrong table name, most show gender symbols (`malevenus`). An opening parenthesis causes an error in three, a closing bracket in two programs. Three programs cannot handle an empty table name, while the Forensic Browser has issues with all databases in this category.

Many programs yield errors when analyzing databases containing special encapsulation characters in keywords and identifiers (second folder, file names starting with `02`). SQLiteDoctor only fails on strange column names, but has no issues with encapsulated type definitions. Database Recovery gets 2 scenarios out of 7 right, Forensic Browser only one 1. All other databases are handled incorrectly by these two tools. Phoenix Repair entirely failes for encapsulated column definitions.

Regarding specifically crafted SQL keywords and constraints (third folder, file names starting with `03`), we can conclude that none of the tools correctly handles a database without rowid. However, most of the other scenarios do not cause severe problems.

Encodings

This concerns the fourth folder (file names starting with `04`). The Forensic Browser fails at databases encoded in UTF-16be. Phoenix Repair cannot handle any UTF16 encoded databases. It also fails at non-latin characters. Undark is able to print ASCII-characters only, while SQLiteDoctor always converts to a UTF8 encoded database.

Database elements

When analyzing trigger, views and indices (fifth folder, file names starting with `05`), Database Recovery misses to display an index. All other scenarios are correctly processed by the applicable tools.

However, none of the programs can handle the databases using the R* tree module (files 06-01 and 06-02), except from SQLiteDoctor. The latter copies contents from the old database to a new one, resulting in a valid database, but a massive amount of error messages is thrown during the analysis. The content of the database using the FTS extension (file `06-03`) can be displayed by every program. The structure of the FTS causes the contents to be principally readable by the tested tools, although none of them supports FTS. That is why the assignments of contents to tables are often not correct.

Tree & page structures

Most programs can handle records that spill to overflow pages well (seventh folder, file names starting with 07). We detected only minor errors for some records. Phoenix Repair struggles most, as it is not able to display the entire record in 3 out of 4 scenarios.

Regarding reserved space at the end of each page (file `08-01`), the only program allowing its user to find "hidden" messages is SQLite Parser. Both hidden texts can be found in the output of the program. Crucial is the result yielded by SQLiteDoctor: The output database resulting from the analysis seems to comply to a "standard template", with a default encoding and without any reserved space. This means that SQLiteDoctor destroys some traces in this case.

The database file containing a pointer-map page (file `09-01`) is correctly handled by all tools, except from Database Recovery. The latter displayed different amounts of entries (100-750) during multiple runs and there is no obvious explanation for this behaviour.

Deleted & overwritten contents

The crafted scenarios regarding deleted tables are very different (file names starting with `0A`, see Table 4). For example, in some cases all records have been deleted before dropping a table, while in other cases tables have been dropped without deleting the records. The evaluation shows different results for these cases: Without deleting records, Undark restores every record. Having them deleted, Undark restores about half of the records. SQLite Parser behaves the other way round, being able to recover text fields from dropped tables with deleted records, but none of the "not-deleted" records. Neither Phoenix Repair nor Forensic Browser can restore a single record. In case of overwritten tables (file names starting with `0B`), deeper analysis of the databases showed that it is not possible to restore deleted records, because the relevant storage areas of a page seem to be cleared with nullbytes upon reuse – even if the `secure_delete` setting is deactivated. This is why none of the recovery programs can restore a single record.

SQLiteDoctor, Phoenix Repair (except special chars) and Database Recovery are able to print the logically existing records, even if deleted records are present in the database (file names starting with `0C`). Undark, however, shows unexpected behavior, because neither existing nor deleted records are displayed in some cases, if deleted records are present in the database. SQLite Parser successfully restores every text field that has been deleted. Forensic Browser can sometimes restore every record (e. g. `0C-01`) and sometimes only a few records (e. g. `0C-03`). Restored records may be assigned to the wrong table by this tool. If recovery mode is enabled, neither existing nor deleted records are displayed, when they contain floating point numbers. These are only displayed by Forensic Browser when in non-recovery mode. So, none of the tools is able to recover floating point numbers, while only Forensic Browser correctly restores integer values and only SQLite Parser allows for recovering all text fields. When recovering overwritten records (file names starting with `0D`), one of the major differences is the inability of Forensic Browser to display overwritten records. SQLite Parser displays at least some texts contained within the overwritten records.

Recovering records from deleted overflow pages seems not to be trivial as well. From the first database (file `0E-01`) only some records are correctly recovered by the three tools offering undeletion (Undark, SQLite Parser, Forensic Browser). From the second database (file `0E-02`) none of the programs can restore a record completely, some only in parts.

Conclusion

In this work, we introduce the – to our knowledge – first standardized corpus targeting SQLite databases: the *SQLite Forensic Corpus*. It can be leveraged for tool testing and validation as well as for the development and comparison of new algorithms and tools. It comprises 77 database files that are in a consistent state and fully comply with the SQLite3 file format. 50 database files focus on crafted peculiarities of SQL statements and of structures internal to the file format, while 27 databases feature explicitely deleted artifacts that may be recovered.

A set of six different programs targeting the (forensic) analysis of SQLite databases was tested against the newly created corpus. Half of the tools are to be considered database viewers, while the other half is promoted to be able to recover data previously deleted from a database. The results clearly show that there is no perfect tool for the analysis of SQLite, since all struggle with different peculiarities. Especially the undeletion of entries containing numeric values, be it integers or floating point numbers, pose severe problems to the recovery tools. Our results can therefore be used to help forensic examiners select the tool with least deficiencies for a particular analysis task.

We draw the following conclusions from the analysis results of our work. These should be understood as basic requirements that a forensic tool needs to satisfy and can be considered as a guideline in the future developments of any analysis software for SQLite:

1. Traces from underlying evidence shall not be destroyed when converted or transferred to the output of a forensic analysis.
2. Erroneous analysis in parts of the evidence shall not cause the elimination or (silent) omission of other parts like those that have not yet been analyzed.
3. Analysis of existing, logically present data in evidence, if supported, shall not be degraded by activating or using the functionality for data recovery.

Finally, we make the SQLite Forensic Corpus presented in this paper available under the following URL: https://faui1-files.cs.fau.de/public/sqlite-forensic-corpus/.

References

Daniels, Paul L., 2017. Undark-a SQLite Deleted and Corrupted Data Recovery Tool. Online; Last accessed: http://www.pldaniels.com/undark. (Accessed 6 October 2017).

DeGrazia, Mari, 2017. SQLite-deleted-records-parser-Script to Recover Deleted Entries in an SQLite Database. Online; Last accessed: https://github.com/mdegrazia/SQLite-Deleted-Records-Parser. (Accessed 6 October 2017).

Digital Corpora, 2009. Producing the Digital Body. Online; Last accessed: http://digitalcorpora.org. (Accessed 18 December 2016).

Dublin Core Metadata Initiative, "Dublin Core Metadata Element Set", Version 1.1, Reference Description.

Garfinkel, S.L., 2006. Forensic Feature Extraction and Cross-drive Analysis. Digit. Invest. 3, 71–81. https://doi.org/10.1016/j.diin.2006.06.007. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

Garfinkel, S.L., 2007. Forensic Corpora: a Challenge for Forensic Research, pp. 1–10. URL. http://www.academia.edu/download/30602968/10.1.1.102.3537.pdf.

Garfinkel, S.L., 2009. Automating disk forensic processing with sleuthkit, xml and python. In: SADFE. IEEE Computer Society, pp. 73–84. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?reload=true&punumber=5341543.

Garfinkel, S.L., 2012. Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus. Digit. Invest. 9 (Supplement), S80–S89. https://doi.org/10.1016/j.diin.2012.05.002. The Proceedings of the Twelfth Annual DFRWS Conference.

Garfinkel, S.L., 2012. Digital forensics xml and the dfxml toolset. Digit. Invest. 8 (3–4), 161–174.

Garfinkel, S.L., Abhi, S., 2003. Remembrance of data passed: a study of disk sanitization practices. IEEE Secur Priv 1 (1), 17–27. https://doi.org/10.1109/MSECP.2003.1176992.

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6, S2–S11. https://doi.org/10.1016/j.diin.2009.06.016. The Proceedings of the Ninth Annual DFRWS Conference.

Grajeda, C., Breitinger, F., Baggili, I., 2017. Availability of datasets for digital forensics – and what is missing. Digit. Invest. 22, S94–S105. https://doi.org/10.1016/j.diin.2017.06.004.

Guo, Y., Slay, J., Beckett, J., 2009. Validation and verification of computer forensic software tools - searching function. Digit. Invest. 6, S12–S22. https://doi.org/10.1016/j.diin.2009.06.015.

Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K., 2000. The 1999 darpa off-line intrusion detection evaluation. Comput. Network. 34 (4), 579–595.

Sanderson Forensics, 2017. Forensic Browser for SQLite. Online; Last accessed: http://sandersonforensics.com/forum/content.php?198-Forensic-Browser-for-SQLite. (Accessed 6 October 2017).

SQLabs, L.L.C., 2017. SQLiteDoctor. Online; Last accessed: http://www.sqlabs.com/sqlitedoctor.php. (Accessed 6 October 2017).

SQLite Online Documentation, 2004. SQLite Version 3 Overview. Online; Last accessed: http://www.sqlite.org/version3.html. (Accessed 29 September 2017).

SQLite Online Documentation, 2016. Most Widely Deployed SQL Database Engine. Online; Last accessed: http://www.sqlite.org/mostdeployed.html. (Accessed 18 December 2016).

SQLite Online Documentation, 2016. File Format for SQLite Databases. Online; Last accessed: http://www.sqlite.org/fileformat2.html. (Accessed 9 August 2016).

SQLite Online Documentation, 2017. Clustered Indexes and the without ROWID Optimization. Online; Last accessed: http://www.sqlite.org/withoutrowid.html. (Accessed 6 October 2017).

SQLite Online Documentation, 2017. ROWIDs and the INTEGER PRIMARY KEY. Online; Last accessed: http://www.sqlite.org/lang_createtable.html#rowid. (Accessed 6 October 2017).

SQLite Online Documentation, 2017. The SQLite R*Tree Module. Online; Last accessed: http://www.sqlite.org/rtree.html. (Accessed 6 October 2017).

SQLite Online Documentation, 2017. SQLite FTS3 and FTS4 Extensions. Online; Last accessed: http://www.sqlite.org/fts3.html. (Accessed 6 October 2017).

SQLite Online Documentation, 2018. Pragma Statements Supported by SQLite. Online; Last accessed: http://www.sqlite.org/pragma.html#pragma_secure_delete. (Accessed 14 January 2018).

Stellar Data Recovery, 2017. Stellar Phoenix Repair for SQLite-repair Damaged or Corrupt SQLite Database Files. Online; Last accessed: https://www.stellarinfo.com/sqlite-repair.php. (Accessed 6 October 2017).

SysTools Software, 2017. SQLite Database Recovery. Online; Last accessed: https://www.systoolsgroup.com/sqlite-database-recovery.html. (Accessed 6 October 2017).

Woods, K., Lee, C.A., Garfinkel, S.L., 2011. Extending Digital Repository Architectures to Support Disk Image Preservation and Access. In: Newton, G., Wright, M.J., Cassel, L.N. (Eds.), Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries. ACM, New York, pp. 57–66. http://doi.acm.org/10.1145/1998076.1998088.

Yannikos, Y., Graner, L., Steinebach, M., Winter, C., 2014. Data Corpora for Digital Forensics Education and Research. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics X. Springer, Berlin Heidelberg, pp. 309–325. https://doi.org/10.1007/978-3-662-44952-3_21. URL https://doi.org/10.1007/978-3-662-44952-3_21.

Zarate, C., Garfinkel, S.L., Heffernan, A., Gorak, K., Horras, S., 2014. A Survey of XOR as a Digital Obfuscation Technique in a Corpus of Real Data. Tech. rep. The NPS Institutional Archive, Monterey, California, calhoun. All Technical Reports Collection, NPS-CS-13–1005. http://hdl.handle.net/10945/38680.