A Recovery Approach for SQLite History Recorders from YAFFS2

Beibei Wu, Ming Xu, Haiping Zhang, Jian Xu, Yizhi Ren, and Ning Zheng

College of Computer, Hangzhou Dianzi University, Hangzhou 310018
Jhw_1314@126.com,{mxu,zhanghp}@hdu.edu.cn

Abstract. Nowadays, forensic on flash memories has drawn much attention. In this paper, a recovery method for SQLite database history records (I.e. updated and deleted records) form YAFFS2 is proposed. Based on the out-of-place-write strategies in NAND flash memory required by YAFFS2, the SQLite history recorders can be recovered and ordered into timeline by their timestamps. The experiment results show that the proposed method can recover the updated or deleted records correctly. Our method can help investigators to find the significant information about user actions in Android smart phones by these history recorders, although they seem to have been disappeared or deleted.

Keywords: Digital forensics, Android, YAFFS2, SQLite, Recovery.

1 Introduction

With the growth of Android smart phones, the need for digital forensics in this area has shown a significant increase. For the small size, and fast running speed, SQLite is widely used in application software that needs to save simple data in a systematic manner or adopted into embedded device software. In the Android, a large amount of user data is stored in the SQLite database, such as short messages, call logs, and contacts [1]. Considering that deletion of data is frequently practiced in order to manage storage space or to update with the latest data, acquiring deleted data information is equally as important as retrieving undamaged information from the database.

Although since the beginning of 2011 with version Gingerbread (Android 2.3), the platform switched to the EXT4 file system, there are still many devices in use running a lower version than 2.3 and using YAFFS2. Therefore, insights into the amount and quality of evidence left on YAFFS2 devices are still of major interest. Compared to databases in some other physical environments, because of the write-once limitation of NAND flash and YAFFS2 page allocation mechanism [2-3], SQLite in restoring means are different. This paper proposed a recovery method of updated or deleted record for SQLite database based YAFFS2. Consequently, with sequence number and timestamps for SQLite database file, we can construct timeline of SQLite operating log and then analyze user actions.

2 Related Work

Researches of database record recovery are already begun in 1983 by Haerder [4]. He suggested that the deleted record can be recovered using transaction file. This method can be applied to traditional database on PC when the information of deleted record is included in transaction file.

A study conducted by Pereira [5] attempted recovering deleted records in Mozilla Firefox 3 using rollback journal file. In the paper, an algorithm to recover deleted SQLite entries based on known internal record structures was proposed and an exception was used that the rollback journal is not deleted at the end of each transaction when the database is used in "exclusive locking" mode.

While the objective of deleted data recovery in SQLite is on the line of Pereira's work, a tool using recovery method that approaches actual data files instead of a journal file that would offer improved practical availability was suggests by Sangjun Jeon [6]. They analyzed the file structure of SQLite database and proposed a method to recover deleted records from the unallocated area in page. However, this method is hard to recover deleted records because remained data are partial in deleted area and the length of each field is difficult to estimate. For the android phone that implemented wear-leveling by YAFFS2, the deleted records can be recovered from previous versions of file.

3 Recovery Elements of SQLite Deleted Records

From the "out-of-place-write" strategy of YAFFS2 and atomic commit in SQLite [7], a deleted SQLite file could be recovered. Therefore, the deleted records can also be restored. In YAFFS2, obsolete chunks can only be turned into free chunks by the process of garbage collection. Whenever one or more obsolete chunks exist within a block, the corresponding data is still recoverable until the respective block gets garbage collected. And from the perspective of the storage mechanism of YAFFS2, it can be concluded that every object header corresponds to a version of file. Once a transaction occurs, there will be an object header and a new version is created. So, the deleted file can be recovered until the respective block gets garbage collected. And versions of each database can be restored as much as possible.

4 The Proposed Method

The process framework of the proposed algorithm is shown in Fig. 1.

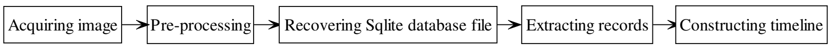

Fig. 1. The process framework of the proposed algorithm

There are two ways to acquire Android image—physical and logical method. Physical method is carried by JTAG [8] while logical method is carried out by "DD" or "NANDdump" instruction after rooting is executed. In this paper, only the logical method is considered.

The pre-processing sequentially records each chunk's objectID, objectType, chunkID, and chunkType in accordance with the allocation order on chip. All the blocks of a flash chip are sorted by *sequence number* from the largest to the smallest. And then these sorted blocks are scanned from the one with the largest *sequence number* to the one with the smallest, and within a block, its chunks are scanned from the last one to the first. During this process, *objectID*, *objectType*, *chunkID* and *chunkType* of each chunk are stored into an array csq[] separately. A simplified algorithm of recovering sqlite database file in pseudo-code is shown below.

```
Algorithm: Recovering SQLite database file
Input: the array of structures csq[] and the image
Output: all SQLite files group by the filename
1.for each chunk in csq[] do /*scan the entire image reversely
      in chronological order*/
2.  if (chunkType=0x80) and (objectType=0x10) then
        /*recognize a file's object header chunk*/
3.    Find the data chunk of chunkID=1;
4.    if (magic number="SQLite format 3") then
          /*Read the magic number in file header and de-
          termine sqlite database file*/
5.      if the folder named by database's filename does not
            exist then
6.          create this folder;
7.      Extract information such as timestamps, objectID and
        file's length and store it;
8.      Calculate the num_chunks of the file by formula (1);
9.      Find all data chunks for the file;
10.     Calculate all data chunks' physical address by formula
        (2~4);
11.     Reassemble this file named by filename and its
        objectID in the folder;
```

$$num\_chunks = \lceil length / size\_chunk \rceil \tag{1}$$

Here *num_chunks* is the number of chunks the file occupies, *length* is the file's length, and *size_chunk* is the size of a page on NAND flash chip.

$$block\_offset = bsq[k/N] \tag{2}$$

$$chunk\_offset = N - (k\%N) \tag{3}$$

$$chunk\_address = block\_offset + chunk\_offset \tag{4}$$

Here *k* means the *k*th chunk which was stored, *N* is the number of chunks in a block, *block_offset* is the chunk's physical block offset, *chunk_offset* is the chunk's relative offset in a block, and *chunk_address* is its physical address.

When scanning the whole chip, all sqlite database file are recovered indicated by object header. During this process, all files are grouped by the filename, and then distinguished by the history version number. In next step, all the records are extracted from each recovered integrated database file and stored into a CSV file, and verify that the integrity of the file that we restored.

When all recoverable history version of a same database file are recovered, a timeline for SQLite CRUD operations can be constructed by timestamp recorded in object header. In contrast of the SQLite records extracted from the two adjacent versions files, the SQL event from one change to another can be inferred. Then, through the analysis of the low-level SQL events corresponding to each user action, a user actions timeline can be constructed by analyzing the entire timeline of SQL event. Finally, we can have a global awareness about what the user did and when.

5 Experiments

In this part, a publicly dataset experiment is used to verify the effectiveness of the recovery method we proposed in the real scene.

The DFRWS has created two scenarios for the forensics challenge in 2011[9]. Images for Scenario 2 were acquired through NANDdump that OOB area can be acquired. 269MB File mtd8.dd for Scenario 2 was used in this experiment because it is the user image that contains a large number of user information. 110 sqlite files of different versions are recovered. Comparing our experimental results with the DFRC team's result, the latest version of all files in our result is equal to the DFRC team's result. In addition, the older versions of the sqlite file can be recovered using our method and the user actions can be analyzed using these files.

For example, the recovered file mmssms.db contains the user's short messages sent and received using this device. In our result, 16 versions file are recovered. Then a timeline **Fig. 2** and **Fig. 3** can be constructed according to these files. In the **Fig. 3**, you can clearly see what the user did and when. For instance, insert a record of id=17 in 04:43:04 represents the user received a short message. Update a record in 04:43:13 represents the user read it. Similarly, other database files can be analyzed too. Through a joint analysis of the results of all databases, a whole timeline of user behavior can be obtained. Experiment in this part shows that our proposed method is suitable for the real case and play an important role in forensic work.

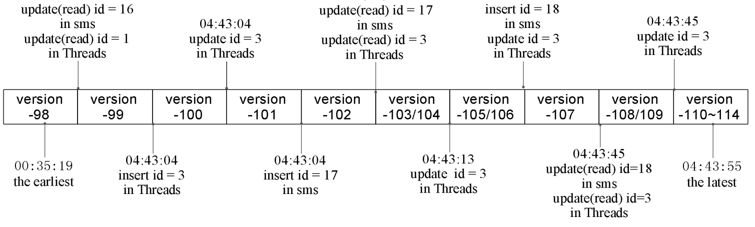

Fig. 2. Timeline of the SQL event about mmssms.db

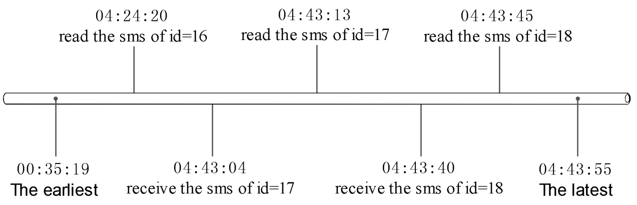

Fig. 3. Timeline of the user action about mmssms.db

6 Conclusions and Future Work

In this paper, a recovery method for SQLite record based YAFFS2 is proposed. Then construct timeline for SQLite CRUD operations by utilizing timestamp recorded in object header, and these may supply significant information about user behaviors to forensic investigations. The experimental results show the efficiency of the proposed method. This paper proves that file recovering from flash chips is practical, but as ext4 file system is widely used in android phone and Linux system, more forensic research is needed on ext4 in digital forensic perspective. Thus the technology of recovering data records from SQLite in ext4 file system will be our research direction.

Acknowledgements. This work is supported by the NSF of China under Grant No. 61070212 and 61003195, the Zhejiang Province NSF under Grant No. Y1090114 and LY12F02006, the Zhejiang Province key industrial projects in the priority themes under Grant No 2010C11050, and the science and technology search planned projects of Zhejiang Province (No. 2012C21040).

References

1. Quick, D., Alzaabi, M.: Forensic Analysis of the Android File System Yaffs2. In: 9th Australian Digital Forensics Conference (2011)
2. Manning, C.: How YAFFS works (2012), `http://www.yaffs.net/documents/how-yaffs-works`
3. Manning, C.: YAFFS Spec (2012), `http://www.yaffs.net/yaffs-2-specification`
4. Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. 15(4), 287–317 (1983)
5. Pereira, M.T.: Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records. Digital Investigation 5(3), 93–103 (2009)
6. Jeon, S., Bang, J., Byun, K., et al.: A Recovery Method of Deleted Record for SQLite Database. Pers. Ubiquit. Comput. 16(6), 707–715 (2012)
7. Atomic Commit in SQLite (2012), `http://www.sqlite.org/atomiccommit.html`
8. Breeuwsma, M.: Forensic imaging of embedded system using JTAG(boundary-scan). Digital Investigation (2006)
9. DFRWS. DFRWS-2011-challenge (2011), `http://www.dfrws.org/2011/challenge/index.shtml`