A Reconstructing Android User Behavior Approach Based on YAFFS2 and SQLite

Ming Xu*
College of Computer, Hangzhou Dianzi University, Hangzhou, China
Email: mxu@hdu.edu.cn

Jun Yao, Yizhi Ren, Jian Xu, Haiping Zhang, Ning Zheng, Shiyue Ling
College of Computer, Hangzhou Dianzi University, Hangzhou, China
Email: { renyz, jian.xu, zhanghp, nzheng }@hdu.edu.cn

***Abstract*—Nowadays, a variety of Android user behavior information is automatically stored in SQLite to indicate when and what user behavior took places. In this paper, an approach to reconstruct Android user behavior from YAFFS2 based on SQLite is proposed. Based on the storage mechanism of YAFFS2 file system and the file structures of SQLite, all of the SQLite records can be recovered from the Android image, regardless of whether the SQLite file has been deleted or not in YAFFS2, and the user behaviors are parsed from those recovered SQLite records; then an Android user behavior timeline is constructed for visualizing based on the time stamp stored in it's SQLite records. The evaluated experiment results show that the proposed method can reconstruct user behavior correctly, and can obtain more user behaviors than Encase to help investigators to complete digital forensic.**

***Index Terms*—Digital forensics, Android, YAFFS2, SQLite, User behavior**

I. Introduction

With the growth in functionality and market share of Android smart phone, increasing numbers of people use them for day to day activities. In criminal cases, Android smart phones often contain relevant communication information about the user, incriminating images, videos or documents transferred from PC. In this paper, we define the most common daily events those take place in Android as user behavior, such as sending or receiving an SMS or phone call, adding a contact, etc. And a large amount of user behavior is stored in the SQLite database. In the paper, we only consider the user behavior that stored in SQLite. Different Android phones have different file systems based on their versions and manufacturers. There are still many devices that are running a lower version than Gingerbread (Android 2.3) in use, the official file system of this kind of Android smart-phone is YAFFS2. Therefore, it is significant to handle such devices that are using YAFFS2. A lot of forensic methods are developed to recover and analyze data from Android smart phone. But those existing forensic investigation methods performed poorly in collecting data selectively and automatically for user behavior, mainly reflecting in costing long time at the collection of data and further prolonging the period of analysis of data. Investigators need to select and analyze data to reconstruct useful user behavior from large number of data. It seriously affects the investigator's work. Besides, since the capacity of storage media gets higher and user behavior become more diversified, such a phenomenon become gradually worsen.

This paper proposes a method to reconstruct Android user behavior based on YAFF2 and SQLite, and construct a user behavior timeline. Unlike the existing investigation methods, the proposed method can reconstruct user behavior correctly, based on events which consist of the behaviors users had done at a specific time. Besides, we developed a visualized tool to display user behavior, making the forensic investigation more convenient.

This paper is structured as follows: Section2 introduces some related work; the forensic-friendly specifics of SQLite and YAFFS2 are presented briefly in Section3; In Section4, we propose a method of reconstructing user behavior from YAFFS2 and constructing the user behavior timeline; Section5 implements an Android smart-phone forensics system and Section6 describes the evaluated experiment. The paper ends up with the discussion of the future works and conclusions (Section7).

II. Related Work

Although there are some previous researches were recovering and analyzing user data from Android smart-phone, they are still insufficient in recovering or visualizing user data. This section will discuss the existing researches about recovering and visualizing user data from Android.

A study conducted by Ming Xu [1] attempted to recover files, reconstruct file system and recover their previous history versions trace from YAFFS2 based on metadata. For recovering all files based on metadata, even damaged files can be recovered. However, some information stored in the damaged file cannot to be obtained, and investigators need to select useful file from large number of file.

Another method to recover history records for SQLite database was suggested by Beibei Wu [2]. While the objective of deleted SQLite file recovery is on the line of Ming Xu's work. In this paper, an algorithm based on rollback journal and metadata to recover deleted SQLite

doi:10.4304/jcp.9.10.2294-2302

file was proposed. However, the information stored in the damaged SQLite database file still cannot to be obtained.

A tool called 'ADEL' (Android Data Extractor Lite) was designed by Michael Spreitzenbarth [3]. It can automatically dump predefined SQLite database files from Android devices and extract the contents stored in the dumped database. However, the tool can only obtain the data contained in Android devices and it cannot recover the deleted records of SQLite database files.

Another tool called 'DFRC USER BEHAVIOUR ANALYZER' was developed by Namheun Son [4]. It collects events from computer and smart-phone to analyze users' behavior patterns. This tool does not provide a method to recover data, it is used to analyze the data that already recovered and it only supports iOS of iPhone.

Several commercial forensic tools have been developed for mobile phones, including Oxygen Forensic Suite [5], EnCase [6] and MOBILedit! [7]. Additionally, hardware solutions for forensic analysis such as XRY [8] are available. The tool implemented by us in this paper doesn't provide all the functions of those existing tools. This paper aims to reconstruct user behavior correctly from YAFFS2 and visualize its timeline for the convenience of investigators to complete digital forensic.

This paper depicts the specifics of storage mechanism of YAFFS2 and internal structure of SQLite on NAND flash, and proposes a method to reconstruct user behaviors from YAFFS2. Meanwhile a timeline tool, which can reconstruct user behavior correctly and completely, is designed to visualize user behavior data for investigators.

III. Specifics Of YAFFS2 And SQLite

A. YAFFS2

For smart phones, hard disks are too large in size, too fragile and too high in power consumption to utilize in reality. In contrast, flash memory provides fast read access time and better kinetic shock resistance than hard disk. There are two fundamentally different types of flash memory: NOR (Not-OR) and NAND (Not-AND), where NOR is bit-addressable, and NAND is block addressable [9]. NOR memory can be used like RAM, with every single bits being addressed and applications can run directly from it. NOR is low density, offers slow writes and fast reads. NAND cannot address individual bit, and must access blocks of memory through a controller (Myers, 2008) [10]. Nevertheless, NAND is low cost, high density and offers fast writes and slow reads [11]. Android (version lower than 2.3) mobile phone's internal memory is a NAND flash chip using YAFFS2 file system to manage data.

NAND flash memory contains three logical structures: flash erasable zone, flash block and flash page. The flash erasable zone is the unit of managing bad block, because the NAND flash almost contains bad block. A flash erasable zone contains one or more flash blocks, and a flash block is comprised of 32, 64 or 128 flash pages. The flash page is the mini-addressable unit in NAND.

A flash page contains two parts: usable area and spare or Out Of Band (OOB) area. The usable area stores the user data while the OOB area stores NAND drive data. In a flash page, the usable area and OOB area has a different size based on their manufacturers. For most Android devices, the usable area contains 2048 bytes while the OOB/spare area contains 64 bytes, where various tags and metadata are stored in blocks.

a. Block and chunk allocation of YAFFS2

YAFFS2 use the chunk that has the same size with the usable area of a NAND flash page as the mini-allocation unit to store data. Usually the NAND drive using only a portion of the OOB area of a NAND flash page, the rest portion of the OOB area stores YAFFS2's meta-data.

YAFFS2 allocates the block and chunk in sequence: anytime there is only one the allocating block that is termed *the allocation block*. After finding the block, it will allocate chunks for the file sequentially from *the allocation block*. When *the allocating block* runs out of chunks, if there are other block not in use, the block are used to become *the allocation block* sequentially. Otherwise, YAFFS2 will recycle the dirtiest block according to garbage collection and make the recycled block to become *the allocation block*.

Because NAND flash only sustain a limited number of writes and erases, In order to increase service time of NAND flash, YAFFS2 will not rewrite directly but write data to a new chunk when data stored in a chunk modified.

b. File storage mechanism of YAFFS2

In YAFFS2, all types of data (directory, regular data file, hard link, soft link, etc.) are treated as objects. Each object is stored in the form that is composed of an Object header chunk and a plurality of Data chunks. Data chunk contains data of a file, while Object header chunk is used to store the meta-data of an object (the owner of ID, group ID, object size, object type, object name, etc.).

YAFFS2 stores a file on the NAND flash memory like this: when a new file needs to be stored, it will allocate a Object header chunk with a *ObjectType* and a specific *objectID* which are stored in OOB area of the Object header chunk. Each file has different *objectID*. The Object header chunk contains the file's meta-data such as file name, time stamps (access time/modified time/create time), length of the file, etc. Data chunks following with The Object header chunk are allocated to store file's data. Each Data chunk can be organized by *chunked* and *ChunkType* which are stored in OOB area of the Data chunk.

When a file is modified, YAFFS2 will not rewrite directly but write data to a new chunk. This non-overwritten strategy is named "out-of-place-write". YAFFS2 will allocate a new Object header chunk with new information to indicate this operation and a plurality of Data chunk to store the update data. So an Object header chunk with new information can be used to indicate an operation. Even though a file is deleted, YAFFS2 will never rewrite the Data chunk directly.

YAFFS2 stored the meta-data in OOB area of Object header chunk and Data chunk, such as *objectID*, *ObjectType*, *chunkID*, *ChunkType*, *sequence number* and *bytenum*, etc. *ObjectID* indicates which object a chunk belongs to; *ObjectType* indicates that an object is a file or a directory (file: 0x10/directory: 0x30); *chunkID* tells where the chunk belongs within an object; *ChunkType* is used to illustrate that this chunk is a data chunk or object header chunk (Object header: 0x80/Data: 0x00); *sequence number* increases by 1 when a block is allocated and every chunk in that block shares the same number, so all the allocated blocks can be organized in chronological order by this special tag; *bytenum* shows the number of bytes of valid data in a data chunk.

B. Internal Structure of SQLite

The SQLite database file tends to become fragmented because of "out-of-place-write" strategy of YAFFS2. Thus, the most efficient method to obtain the SQLite records for reconstructing user behaviors is to search by the records themselves and ignore all the database structure. To do this, knowledge of their internal structure is required.

Essentially, SQLite database is stored in segments, called *pages* [12]. A SQLite database is composed of multiple B-trees, and every B-tree takes a full page at least. One B-tree for each *table* and *index*, structured as B-trees for *index*, and B+trees for *table*. Each *table* or *index* in a SQLite database has a root page that defines the location of its first page. The root pages for all *indexes* and *tables* are stored in the *sqlite_master* table. The *sqlite_master* table is a system table that contains information about all the *table*, *view*, *index*, and *trigger* in the database. And the *sqlite_master* table has a fixed structure: *sqlite_master* (*type* TEXT, *name* TEXT, *tbl_name* TEXT, *rootpage* INTEGER, *sql* TEXT), the *type* field is one of *table*, *view*, *index*, and *trigger*; the *name* and *tbl_name* field is the table name for a table; the *rootpage* field is the root page of the table; the *sql* field stores the SQL text related to how to create the table. Furthermore, the root and internal pages of B+trees only contain only navigation information and the table data (database records) which are stored in leaf pages, as shown in Figure.1 (Owens, 2010, page 305). We can obtain the SQLite records according to the SQL text that stored in the *sqlite_master* table ignoring B+tree structure.

The SQLite record is stored in binary form using a specialized record format that describes all the fields in the record, it has the following sequential structure: the *record* size, the *id value*, the *header* and the *data segment* (D1 to DN), as shown in Figure. 2. The *record size* only includes the size of *header* and *data segment*. The *header* comprises the *header size* (*hsize*) and an array of field types and sizes (F1 to FN), which describes each field stored in the *data segment*. The *hsize* and the array, as well as the id, are represented as a variable-sized 64-bit integer value. For this kind of value, SQLite uses a compression method based on Huffman coding [13].

It is important to note that the *record size*, the *id*, and the *header size* are stored using Huffman coding. However, fields' sizes and data do not use the compression method, the array of types and sizes in the header uses one value to identify the field type and the size. Table I lists the entry specifies data type and the size of its corresponding field value.

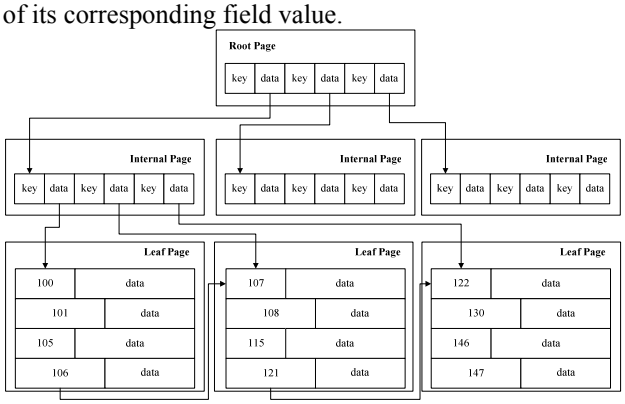

Figure 1. B+ tree structure of SQLite databases (Owens, 2010, page 305)

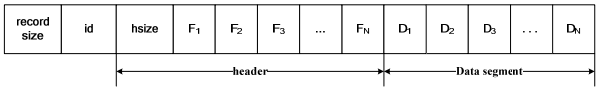

Figure 2. Record structure

TABLE I.
FIELD TYPE VALUES

Type Value	Meaning	Length of Data
0	NULL	0
N in 1..4	Signed integer	N
5	Signed integer	6
6	Signed integer	8
7	IEEE float	8
8-11	Reserved for future use	N/A
N>12 and even	BLOB	(N-12)/2
N>13 and odd	TEXT	(N-13)/2

C. User Behavior in Android

There are lots of events taking place in Android smart-phone, but 5 kinds of user behaviors more important: Call history, SMS/MMS, Location info, Web Use History [14] and SNS (Social Networking Services) [15]. Currently, the application software usually uses SQLite to save these user behaviors information in Android. The saved file directory and stored table about 5 kinds of user behaviors are described in Table II.

It is possible to obtain the information that when and with whom users had a phone call by analyzing call history. Contact information and call history information are stored in *contacts2.db* on the user data partition. In the *contacts2.db*, the table *calls* contain detailed information of call history, such as call time, call duration and the number of incoming and outgoing calls. And the detailed information of contacts is stored in the table *data*. For SMS/MMS, the information of when, about what and with whom users talked, can be obtained as well. Unlike Call History, it is possible to know the contents of their conversation, so it can be used as very important data. The information of SMS/MMS is stored in *mmssms.db* on the user data partition. The table *sms* store all data about SMS and the table *pdu* store the detailed information of MMS. When an application within map information such as Google map and Browser is used, the application will

record data indicating the present location, which can be used to detect the user's location. The location information is stored in multiple SQLite database files on the user data partition. The *cachedPositions.db* contains the location information which is stored in the browser, and the *search_history.db* contains the location information which was ever searched in the Google map. Browser is a good source to find out sites and search words that users have visited and used, and the Android's default browser stores the browser data in *browser.db* on the user data partition. The table *bookmark* contains both bookmarks and browser histories in versions 2.x of the Android platform, and the table *searches* contain the default browser's search histories. People use SNS services like Skype, Facebook, Twitter and Sina-Weibo on their smart-phone, and those services include meaningful data, such as sending text messages and free-for-charge calls. Apparently, it is inevitable to analyze SNS data. These applications also store important user data stored in SQLite database. We have not thoroughly inspected these applications' mechanism of storing data.

IV. The Proposed Method

In this section, we propose a method to obtain useful information in SQLite database file from YAFFS2 and reconstruct user behaviors timeline. The Figure.3 is the framework of the proposed algorithm.

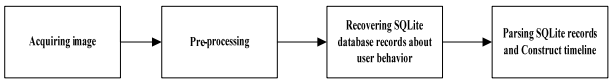

Figure 3. The process framework of the proposed method.

A. Acquiring Image

Before starting to analyze internal file system of android phone, it's needed to obtain the image file of data partition firstly. There are two main ways to get an image from Android devices—physical and logical. Physical method obtain image that carried out by JTAG [16] and logical method carried out by using "DD" or "NANDdump" instruction after the android devices have acquired rooting right.

In this paper, we only consider the logical method to obtain the image file of data partition. To carry out logical imaging, android devices must be rooted beforehand. "SuperOneClick" [17] was used in this work. Once the devices have acquired rooting right, the most forensically-sound method of acquiring data from the device is the following:

- Put the device in developer mode and connect the device with a computer;
- Add "NANDdump" command using Android Debug Bridge (ADB);
- View the MTD partitions using command "adb shell cat";
- Acquiring the image using "NANDdump".

In this work, "NANDdump" instruction is used to acquire the complete data partition image of android phone. The 1.18 version of BusyBox [18] contain the command of "NANDdump". In order to support "NANDdump" command for the device, the ADB tool [19] can be used to install BusyBox on the system partition, and this process only affects the system partition [20].

As a result, the integrity of data partition, which is the research object in most forensic task, can be retained bit-by-bit. A bit-by-bit image allows an investigator to conduct analysis on the entirety of the data, such as deleted data, which is a great role in any investigation.

B. Pre-processing

The pre-processing is such a process that obtains each chunk's meta-data that stored in OOB area in the allocation order on chip. From the perspective of the storage mechanism of YAFFS2, each block's *sequence number* of a flash memory are allocated in ascending order, and each chunk have the same *sequence number* in a block. Therefore, we scan the image in the reverse order of *sequence number* in our approach. Each chunk's *objectID*, *objectType*, *chunkID*, and *chunkType* can be obtained in our approach according to the OOB tags obtaining algorithm as pseudo-code show in Algorithm 1.

The pre-processing work is completed as follows. Firstly, the *sequence number* for each block and the corresponding physical block offset is stored in an array of structures *bsn*[], during scanning each logical block in sequence (lines 1-3). Secondly, all the blocks stored in *bsn*[] are sorted by *sequence number* from the largest to the smallest (line 4). Finally, 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, each chunk's *objectID*, *objectType*, *chunkID* and *chunkType* are stored into an array of structures *pbk*[] separately (lines 5-9). Each chunks physical address can be calculated as below:

TABLE II.
USER DATA STORED IN SQLITE

User's data	Saved file directory	SQLite table
Call history	*data/data/com.android.providers.contacts/databases/contacts2.db*	*calls*
Contact data	*data/data/com.android.providers.contacts/databases/contacts2.db*	*data*
SMS MMS	*data/data/com.android.providers.telephony/databases/mmssms.db*	*sms* *pdu*
Browser data	*data/data/com.android.browser/databases/browser.db*	*Bookmark searches*
Location info	*data/com.google.android.apps.maps/databases/da_destination_history* *data/com.google.android.apps.maps/databases/search_history.db*	*destination_history* *history*
SNS Application	*/data/data/com.android.<Application Name>/databases/*	

$$blockOffset = bsn[i / NUM] \quad (1)$$

$$chunkOffset = NUM - (i\%NUM) \quad (2)$$

$$chunkAddress = blockOffset + chunkOffset \quad (3)$$

Where i means the location of the chunk which is ready to be stored into *pbk*[], *NUM* is the number of chunks in one block, *blockOffset* is the chunk i's block number, *chunkOffset* is the chunk i's relative offset number in the block, and *chunkAddress* is the physical address of chunk i.

Since the whole chip is scanned reversely in chronological order, we don't need to consider the spatial sequence.

ALGORITHM I
OOB TAGS OBTAININGALGORITHM

Algorithm 1: OOB Tags Obtaining Algorithm

Input: the image
Output: the array of structures *pbk*[]
01 **for** each block in image **do** /*scan the entire image in order*/
02 read *sequence number* and *physical number* stored in the array of structures *bsn*[];
03 **end for**
04 sort by *sequence number* from the largest to the smallest for the array of structures *bsn*[];
05 **for** each block in *bsn*[] **do**
06 **for** each chunk in block from the last one to the first **do**
07 calculate chunk's physical address by formula (1~3);
08 read *objectID*, *objectType*, *chunkID*, and *chunkType* stored in the array of structures *pbk*[];
09 **endfor**
10 **endfor**
11 **return** *pbk*[]

C. Recovering SQLite Database Records and Reconstructing User Behavior

Most of user behavior data in Android mobile phone are stored in SQLite database file. In order to reconstruct user behavior, it is necessary to recover SQLite database records about user behavior, and then analyze the low-level SQL events corresponding to each user behavior. Two simplified algorithm in pseudo-code are shown below. We recognize SQLite database file's page in the reverse order of *sequence number* scanning on chip according to recovering SQLite database records algorithm as pseudo-code show in Algorithm 2, and we parse records from each page of SQLite according to parsing SQLite database records algorithm as pseudo-code show in Algorithm 3.

ALGORITHM II
RECOVERING SQLITE DATABASE RECORD ALGORITHM

Algorithm 2: Recovering SQLite Database Records Algorithm

Input: the image;
the array of structures *pbk*[] that stored each chunk's *objectID*, *objectType*, *chunkID* and *chunkType*;
SqliteName = {s_i | s_i is the *ith* sqlite database file's name that you want to obtain records from it, such as *contacts2.db*, *mmssms.db* and so on};
Output: *Records* = {r_i | r_i is the *ith* record we obtained in the sqlite file}
01 $i = 0$;
02 **while** $i <$ *pbk*.length-1 **do**
/*scan the entire image reversely in chronological order*/;
03 **if** (*pbk*[i].*chunkType*=0x80) and (*pbk*[i].*objectType*=0x10) **then**
/*recognize an file's object header chunk*/
04 **if** (the filename$\in$ *SqliteName*) **then**
/*Read the filename in object header chunk and determined the sqlite database file*/
05 Parse the SQLite file's *sqlite_master* table;
/*obtain the *SQL_text* that describes how to create the table and store it*/
06 Extract information such as *time stamps*, *objectID* and store them;
07 **for** each chunk *k* from *pbk*[*i*+1] to *pbk*[*pbk*.length -1] **do**
08 **if** (*k.objectID* = *pbk*[*i*].*objectID*) **then**
09 **if** (*k.chunkType*=0x80) and (*k.objectType* = 0x10)**then**
10 **break**;
11 calculate chunk *k*'s physical address by formula (4~6);
12 *tempRecords*{$\mathbf{t_i}|\mathbf{t_i}$ is the *ith* record obtained from chunk *k*}= Algorithm III (k, *SQL_text*);
/*parse record from chunk based on Algorithm 3*/
13 **if** $k\in$ *ParsedChunk* = {p_i | p_i is the *ith chunkID* of the *ith* chunk has been parsed}; **then**
14 **if** *tempRecords*$\not\subset$*Records* **then**
15 add *time stamps* obtained from the previous object header chunk to the record $t_i \notin$ *Records*;
16 **end if**
17 **end if**
18 *Records* $\leftarrow$*Records* $\cup$ *tempRecords*;
19 *ParsedChunk* $\leftarrow$ *ParsedChunk* $\cup$ *k.chunkID*;
20 **end if**
21 **end for**
22 **end if**
23 **end if**
24 *i*++;
25 **end while**
26 **return** *Records*

ALGORITHM III
PARSING SQLITE DATABASE RECORDS ALGORITHM

Algorithm 3: Parsing SQLite Database Records Algorithm

Input: the chunk *k* of the image and the *SQL_text* obtained from *sqlite_master* table
Output: *tempRecords* = {$t_i|t_i$ is the *ith* record obtained from chunk *k*}
01 read the first 1024 bytes from chunk store in page[];
/*SQlite page size is 1024,chunk's size is 2048*/
02 **if** (*page*[0]=0x0D) **then** /* recognize the B+ tree page*/;
03 *i*=0 and calculate the *total_records* of the file by formula (4);
04 **while** *i* < *total_records* **do** /*traverses every record in the page */;
05 calculate the *record_offset* of the *i* record by formula (5);
06 **for** each field,according to SQL_text **do**;
07 read its content and content's size store them;
08 **end for**;
09 **if** content's size of each field match with Table I **then**;
10 *tempRecords* $\leftarrow$ *tempRecords* $\cup$ record *i*;
11 **end if**
12 *i*++;
13 **end while**
14 **end if**
15 read the another 1024 byte from chunk store in array of structures *page*[];
16 redo step 2 to step 14;
17 **return** *tempRecords*

To parse the Android image, the pre-processing work is completed as Algorithm 2 and all the chunks are scanned reversely in spatial (lines 1-2). An object header chunk is recognized by *chunkType* (*chunkType=0x80*) in OOB (line 3). The information, such as type, *parent_objectID*, *filename*, *time stamps*, *objectID* and file length, is extracted from object header chunk. YAFFS2 uses type in object headers to distinguish a directory from a normal file.

The relationship between a directory and its files is holded by that all the *parent_objectID* of its included files are equal to the directory's *ObjectID*. In order to obtain the record that related with user behavior, these SQLite database file name need to be known in advance (line 4). These SQLite database files can be recognized by their magic number "53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00". This magic number byte sequence corresponds

to the UTF-8 string "SQLite format 3" including the null terminator character at the end. When a database file is identified, the table's SQL text that contains the record related with user behavior can be obtained according to the fixed format of *sqlite_master* table (line 5).

After the SQL text is obtained, the information about the table that contains records about user behavior can be obtained, such as each field's type and its name. As mentioned above, YAFFS2 appends an object header to data chunks when some operations (create, modify and append, etc.) have been done. This means that the chunks between the two object headers, stored the updated data, and the time stamps can be used to show when the operation happened can be obtained from the object header chunk (line 6). Not until the next object header chunk appear, it is necessary to traverse all these chunks that follow with the object header chunk to obtain all record related to operation. The follow chunk have the same *ObjectID* with the object header chunk (lines 7-10), each chunk's physical address can be calculated according to equations 1, 2 and 3 (line 11).

When the location of the chunk has been obtained, the work that parsing SQLite database records from the chunk completed as Algorithm 3. The chunk size is 2048 bytes, while the SQLite page size default is 1024 bytes. Therefore, when the physical address of chunk is determined, we store the following a SQLite page size bytes in *page* [] (line 1). The table's records only store in the SQLite database file's B+tree, and the B+tree page's first byte is *0x0D*. If page is a B+tree page (lines 2-5), records can be obtained from the page. The total record number can be calculated as below:

$$total\_records = page[3] * 256 + page[4] \quad (4)$$

In order to obtain each record from the page, the ith record offset address from page header can be calculated as below:

$$record\_offset = page[8 + 2(i-1)] * 256 + page[9 + 2(i-1)] \quad (5)$$

Then, according to the SQL text, record's entire field's content and size can be obtained (lines 6-8). If each field's size matches with its type, then we stored the record in a set *tempRecords* (lines 9-11). For example if a field's type is TEXT, its size must be an odd number greater than 13. Because a SQLite database file hardly contains two tables with the same SQL text, *tempRecords* contain records related to user behavior. When all records are obtained from the page, another 1024 bytes in the chunk will be processed in the same way as the first 1024 bytes (lines 15-16).

After obtaining records from the chunk, we need continue follow the steps of Algorithm 2. It is necessary to check out whether the chunk has been parsed (line 13). If the chunk has been parsed, then it is necessary to check out whether the chunk contains the deleted record. If the *tempRecords* contains some records that have not been obtained, it indicates that there is a delete operation took place between the previous object header chunk and the object header chunk. And because of all the chunks are scanned reversely order of sequence number, the previous object header chunk contain the time stamps of the delete operation taking place. So the time stamps need to be appended to the record as its DELETE time and store the record (lines 14-17). If the chunk hadn't been parsed, all of these records obtained from the chunk will be stored directly (lines 18-19). When all chunks in the image have been parsed, the recovery work is completed.

After all recoverable SQLite records in the database file are recovered, user behavior need to be parsed from these SQLite records by extracting some important fields in a record related to user behavior. When the call history records need to be parsed into corresponding user behavior, the detailed information about call should be extracted from the call history records, such as the call time, call duration and the number of incoming and outgoing calls. The process involves some unrelated fields. For SMS/MMS, the SMS/MMS body field, the number field, and time field and name field should be extracted from the SMS/MMS history records. To the record related to location information, the longitude, the latitude, the place name and the time fields should be extracted. The most important of the record related to browser is URL and time, both them should be obtained to translate into user behavior. For simplicity, only two most popular SNS services, Twitter and Sina-Weibo, are considered. The reposted tweet or microblog and private message contact with others are important to both, so both them should be obtained to translate into user behavior besides time field.

If the record has a *DELETE time* field, it indicates that the record is a deleted record. Namely, user once deleted the record sometime. For example, a SMS record containing the *DELETE time* field which has a value of *t* means the user deleted the SMS at time *t*. We must extract and reconstruct these user behaviors too.

D. Construct Timeline and Analyze User Behavior

After the user behavior has been reconstructed, a timeline can be constructed by timestamps contained in the corresponding SQLite recorder, like SMS time, call time, visiting website time, deleted time, etc. Every user behavior of timeline is arranged in chronological order, and different icons represent different user behavior. We mark the icon of the delete user behavior in red, because these behaviors may be deemed suspicious.

Through the analysis of the timeline, we will have a global awareness about what and when the user did. Considering a scenario where criminals delete all information, related to other suspect, such as call history, SMS. We can reconstruct the user behavior by recovering the SQLite records and building the user behavior timeline, it will play an important and positive role in the investigation process of such scenario.

V. ATCL FORENSIC TOOL

Based on the above proposed method, a tool named ATCL (Android timeline constructor Lite) is implemented. It is a user behavior timeline construction and analysis tool for versions 2.x of the Android platform. It can automatically recover SQLite records from Android image dumped by *NANDdump* instruction, and construct the timeline of user behavior. The tool converts

SQLite records data into visual timeline so that investigators can easily understand the data and efficient analysis user behavior. Figure 4 shows the workflow of ATCL. In the beginning, it is needed to select a mode on the User Interface after determining whether read an image directly or obtain the image via the Android debugging bridge (adb). Then, the tool starts collecting data from the image. To accomplish this work, we developed a specialized parser module for the image, which extracts SQLite database records from the Android image dumped by *NANDdump* and parsed into corresponding user behavior. After the user behavior is parsed, the tool starts to reinterpreting and visualizing the user behavior. The user behavior timeline is constructed automatically, and the user behavior containing deleted time will be highlight because these behaviors are deemed suspicious, such as deleted call history, deleted SMS and browser searches key word. The tool also has a statistic function about user behavior, such as call, SMS, web visit and etc. By using the statistic function, it is likely to grasp the interest tendencies of Android users in a forensic investigation.

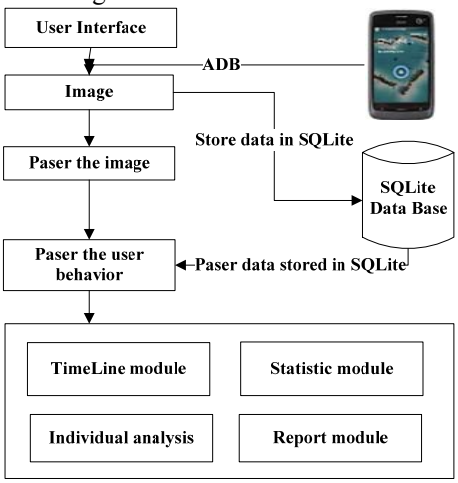

Figure 4. The ATCL workflow

VI. Experiment And Evaluation

In this section, the proposed method in Section IV will be evaluated by experiments. Two types of experiments were designed on three real Android phones and a public and available Android images dataset.

A. Experiments on Three Real Android Phones

Step1: Propagating the system

In order to evaluate the proposed approach in three real phone: ZTE U880, HUAWEI C8650 and MOPS T800. Before the evaluating, three smart phones were rooted using three different methods (modifications based on SuperOneClick) respectively due to the different handset vendors. Actually, all of three smart phones have been used more than a year.

Step2: Experiment design and implementation

In this scene, The ATCL and the latest version of Encase v7.07 [21] had been used to analyze the user behavior in a real android phone (ZTE-U880). There are two ways to analyze method for smart phones in Encase v7.07: logical acquisition and physical acquisition. All of three analysis methods (logical acquisition, physical acquisition by Encase, and the proposed method) used about 2 minutes in this test. The compared result about the user behavior acquired from the android phone was shown in Table III. We can only obtain the undeleted Call, SMS, MMS and contacts from the logical acquisition by Encase, and the data is very limited for analysis. The physical acquisition by Encase can obtain more information than the logical acquisition, and the deleted information can also be obtained. However, both methods by Encase are weak in obtaining the data about the browser history, location information and SNS. The physical acquisition by Encase treats browser history and location information as URL, which will affect the correctness of forensic investigation. Furthermore, Encase only provider visualizes function for analyzing the Android image, and doesn't provide visualizes function for user behavior's data. The ATCL can obtain more information than Encase including 155 browser

TABLE III.
USER BEHAVIOR OBTAINED FROM ZTE-U880

User behavior	Contacts	Call	SMS	Browser history	Google map info	Sina-Weibo
Encase(logical)	72	494	1393	0	0	0
Encase(physical)	72	500	1394	160	0	0
ATCL (our method)	72	500 (6 deleted)	1394 (1 deleted)	188 (33 bookmark)	11	4

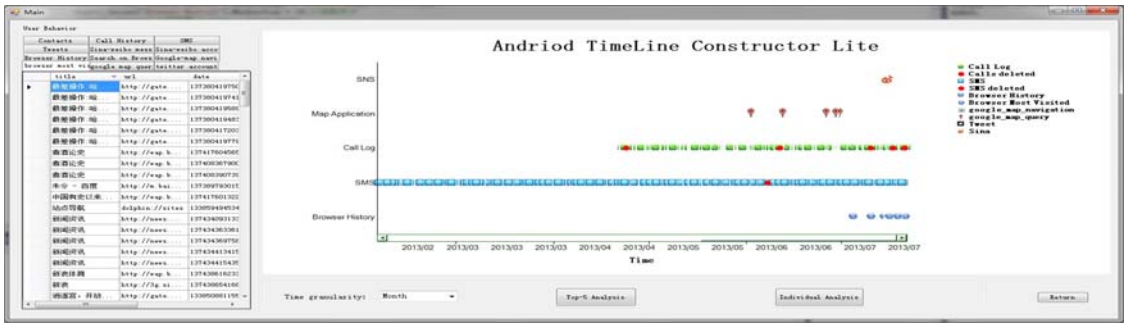

Figure 5. Timeline constructed by ATCL

history, 33 bookmarks, 11 Google map search information and 4 Sina-weibo personal messages. Additionally, it provides a function that visualizes the acquired data by a timeline, as shown in Figure 5. Some suspicious user behavior will be marked red, such as deleted call history, deleted SMS, etc. If something interests you, you can select an object on the timeline. Then the detail information of the user behavior can be showed in a new message box. ATCL can effectively reduce the difficulty of forensic investigation analysis, so that forensic investigation has become more convenient and intuitive. The experiment was also conducted on other smart phones: HUAWEI C8650 and MOPS T800, and the results are similar to that in ZTE U880 phone.

B. Experiments on Public and Available Android Images Dataset

In this part, experiments on a public dataset are used to evaluate the effectiveness of the proposed method.

The DFRWS has created two scenarios for the forensics challenge in 2011 [22]. Images for Scenario 2 were acquired through *NANDdump*. OOB area can be acquired through *NANDdump*. File mtd8.dd for Scenario 2 is the user image that contains a large number of user information. So in this experiment, the proposed method is used on the mtd8.dd image. The user behavior obtained from mtd8.dd is displayed in Table IV. The number of information related to the aforementioned user's behaviors in our result is the same to the DFRWS' final result. The user behaviors timeline generated by the ATCL for the mobile user is shown in Figure 6. We can easily know about what and when the user did from the timeline. Moreover, the recovered and acquired user behaviors information is listed on the left of the timeline, we can easily find any information recovered from the Android image.

VII. Conclusions And Future Work

In this paper, a method to reconstruct user behavior from YAFFS2 based on SQLite was proposed. Based on user behavior such as SMS/MMS, call history and Geo Location, we proposed a method and developed a tool named ATCL to reconstruct user behavior and provide a visual user behaviors timeline. The experimental results show the efficiency of the proposed method.

Although the proposed method is based on the YAFFS2 file system, and the file system had turned to EXT4 in the higher version than Gingerbread (Android 2.3), our method also provides successful experiences of others to go by for different file system and other popular cell-phone platforms. By fully analyzing the storage mechanism of file system, we can collect the user behavior conveniently.

Automated evidence collecting and analyzing plays an important role in digital forensics. In this paper, the proposed method could only be used to analyze YAFFS2 files systems. Automated user behaviors collecting and analyzing from ext4 file system is our research work in the near future.

Acknowledgment

This work is supported by the Natural Science Foundation Natural Science Foundation of China under Grant No.61070212 and 61003195, the Zhejiang Province Natural Science Foundation Natural Science Foundation of China under Grant No.Y1090114 and LY12F02006, the Zhejiang Province key industrial projects in the priority themes of China under Grant No 2010C11050, the science and technology search planned projects of Zhejiang Province (No.2012C21040), the soft science research project of Hangzhou (No. 20130834M15), Project of Department of Education of Zhejiang Province (No. Y201226281).

References

[1] Ming Xu, Xue Yang, Beibei Wu, Jun Yao, Haiping Zhang, Jian Xu and Ning Zheng, "A metadata-based method for recovering files and file traces from YAFFS2." Digital Investigation, vol. 10, pp. 62–72, June 2013.

[2] Beibei Wu, Ming Xu, Haiping Zhang, Jian Xu, Yizhi Ren and Ning Zheng, "A Recovery Approach for SQLite History Recorders from YAFFS2." in the 2013 international conference on Information and Communication Technology, pp.295–299, 2013.

[3] Michael Spreitzenbarth, Sven Schmitt and Felix Freiling, "Comparing Sources of Location data From Android

TABLE IV.
FIELD TYPE VALUES

User behavior	Calls history	SMS	Browser history	Google map info	tweets
Number	4	17	50	1	161

Figure 6. Timeline constructed by ATCL

Smartphones", in International Conference on Digital Forensics, South Africa: Pretoria, 2012, pp.143–157.
[4] Namheun Son and Sangjin Lee, "Forensic investigation method and tool based on the user behaviour analysis." Proceedings of the 9th Australian Digital Forensics Conference, pp. 125–133, 2011.
[5] Oxygen Forensic Suite, http://www.oxygen-forensic.com/.
[6] EnCase, http://www.encaseondemand.com/Home/tabid/632/Default.aspx.
[7] Compelson Labs, MOBILedit! Forensic Overview, www.mobiledit.com/mef-overview.html.
[8] Micro Systemation, XRY Physical, www.msab.com/xry/xry-physical.
[9] *Siddharth Choudhuri, Tony Givargis*, "Deterministic Service Guarantees for NAND Flash using Partial Block Cleaning." Journal of Software, vol. 4, pp. 728-737, 2009.
[10] Daniel Myers, "On the Use of NAND Flash Memory in High-Performance Relational Databases."
[11] NAND vs NOR Flash Memory: Technology Overview, http://www.toshiba.com/taec/components/Generic/Memory_Resources/NANDvsNOR.pdf
[12] SQLite. SQLite Database File Format. http://www.sqlite.org/, 2008.
[13] Wikipedia. Huffman coding. http://en.wikipedia.org/wiki/Huffman_code.
[14] Jianbo Bai, Yuzhe Hao, Guochang Miao, "Integrating Building Automation Systems based on Web Services." Journal of Software, vol. 6, pp. 2209-2216, 2011.
[15] Hui Chen, "Relationship between Motivation and Behavior of SNS User." Journal of Software, vol. 7, pp. 1265-1272, 2012.
[16] Ing. M.F. Breeuwsma, "Forensic imaging of embedded system using JTAG(boundary-scan)." Digital Investigation, vol. 2, pp. 32–42, March 2006.
[17] superoneclick, http://www.superoneclick.net/.
[18] BusyBox, http://busybox.net.
[19] ADB tool, http://developer.android.com/tools/help/adb.html.
[20] Blog about android phone rooting: Blocg of Tegrak, http://pspmaster.tistory.com.
[21] EnCase Forensic v7.07, http://www.encase.com/products/Pages/EnCase-Forensic/Search.aspx.
[22] DFRWS, DFRWS-2011-challenge. http://www.dfrws.org/2011/challenge/index.shtml, 2011.

Ming Xu is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Zhejiang University in 2004. His research interests include Digital Forensics, Network Security, Social Network and Data Mining.

Jun Yao received the B.S. degree in Information and Computing Science from the China JiLiang University in 2011. He is currently a master candidate in computer technology from the Hangzhou Dianzi University, P. R. China. His research interest includes Smart-phone Forensics and Computer Forensics.

Yizhi Ren is a Lecturer in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Dalian University of Technology in 2011. His research interests include Network Security, Social Computing and Evolutionary game.

Jian Xu is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Zhejiang University in 2004. His research interests include Location-based Services, Mobile Computing, Distributed Database and Network Security.

Haiping Zhang is an Associate Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the master degree in Computer Software and Theory from the Hangzhou Dianzi University in 2005. His research interests include Digital Forensics, Network Security, Social Network and Corporate Information Technology.

Ning Zheng is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. His research interests include Network Security, CAD, and CAM.

Shiyue Ling is a Lecturer in the college of Computer, Hangzhou Dianzi University, P. R. China. She received the master degree in Computer Technology from the Hangzhou Dianzi University in 2007. Her research interests include Digital Forensics, Network Security, Social Network and Corporate Information Technology.