Playing Hide-and-Seek: Detecting the Manipulation of Android Timestamps

Heloise Pieterse
Defence, Peace, Safety and Security
Council for Scientific and Industrial Research
Pretoria, South Africa
Email: hpieterse@csir.co.za

Martin S. Olivier
Department of Computer Science
University of Pretoria
Pretoria, South Africa
Email: molivier@cs.up.ac.za

Renier P. van Heerden
Meraka Institute
Council for Scientific and Industrial Research
Pretoria, South Africa
Email: rvheerden@csir.co.za

***Abstract*—Mobile technology continues to evolve in the 21st century, providing users with improved capabilities and advance functionality. One of the leaders of this evolution is Android, a mobile operating system that continuously elevates existing features and offers new applications. Such improvements allowed Android to gain popularity worldwide. A combination of Android's advance technology and increasing popularity allow smartphones supporting this operating system to become a rich source of trace evidence. Traces found on Android smartphones form a significant part of digital investigations, especially when the user of the smartphone is involved in criminal activities. A key component of these traces is the date and time, often formed as timestamps. These timestamps allow the examiner to relate the traces found on Android smartphones to some real event that took place. Knowing when events occurred in digital investigations is of great importance to the overall success of the investigation. This paper introduces a new solution, called the Authenticity Framework for Android Timestamps (AFAT) that establishes the authenticity of timestamps found on Android smartphones. Currently the framework determines the authenticity of timestamps found in SQLite databases by following two individual methods. The first method identifies the presence of certain changes in the Android file system, which are indications of the manipulation of the SQLite databases. The second method subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The presence of specific file system changes as well as inconsistencies in the associated SQLite databases indicates that authenticity of the timestamps might be compromised. The results presented in the paper provide preliminary evidence that the suggested approach, Authenticity Framework for Android Timestamps, shows potential.**

***Index Terms*—Digital Forensics, Mobile Forensics, Smartphones, Android, Timestamps, Manipulation, SQLite Databases.**

I. Introduction

The past decade saw the rapid improvement of smartphone technology, allowing these devices to become very popular across the globe. Their current prominence is directly related to the provided capabilities and functionality, which nowadays closely resemble a personal computer. Bundled with a complete operating system, improved connectivity and communication functions, and the option of adding additional third-party applications, smartphones have become powerful devices. The leading smartphone Operating System (OS) of 2014 was Android [1], which has been evolving in a remarkable way and continues to gain widespread popularity. The current prevalence of Android led this paper to focus only on this particular smartphone OS.

The extensive and wide use of Android smartphones allows these devices to become a rich source of trace evidence [2]. All events occurring on Android smartphones generate traces that form an important component of digital investigations, especially when the user of the smartphone is involved in criminal activities. The valuable information (such as contacts, text messages, call lists, website visited or instant messages) contained in these traces can provide a well-defined snapshot of a user's actions at a specific time. Besides providing a description of the event, traces found on Android smartphones also often store the time and date component in the form of a timestamp. Timestamps are integral to digital investigations since it provides the examiner the opportunity to relate the traces found on the Android smartphones to some physical event that took place. A collection of timestamps can be combined to construct a timeline, which provides the examiner with a chronological ordering of events.

The timestamps associated with these stored traces can be a threat to the user's privacy [3] since the traces provide an overview of the user's actions. To protect their privacy, smartphone users can use certain techniques to manipulate the timestamps of the traces and change the associated events. These techniques are referred to as Anti-forensics and are primarily used "*to compromise the availability or usefulness of evidence*" [4]. These techniques are applied by smartphone users in an attempt to either hide or change event logs, which results in the alteration of the timestamps associated with those events.

Knowing when events occurred in digital investigations is of great importance to the overall success of the investigation. Due to the significance of timestamps, it necessary for examiners to be able to verify their authenticity and accuracy. Without such verification, the collected timestamps might be incorrect or inaccurate due to tampering and will lead the examiner to make unreliable conclusions. Existing research shows few papers that attempt to offer a solution regarding the verification of the authenticity and accuracy of timestamps. Verma et. al. [5] preserve date and time stamps by capturing the system generated Modification, Access, Change and/or Creation Date

978-1-4799-7755-0/15/$31.00 ©2015 IEEE

and Timestamps (MAC DTS) values and storing it in a secure location such as, a cloud server, outside of the smartphone. The cloud snapshot of the original MAC DTS values can be used to verify the authenticity of MAC DTS values of questionable files on the smartphone [5]. Govindaraj et. al. [6] designed a solution, called iSecureRing, which allows a jailbroken iPhone to be secure and forensic ready by preserving the timestamps. These timestamps are stored outside the device on a secure server or the cloud and can be used during security incidents [6]. Both solutions, however, require the installation of additional functionality on the smartphone prior to seizing the device for investigation. There is, thus, no existing solution (to the best of the authors' knowledge) that allows for the verification of timestamps collected from seized Android smartphones.

This paper performs exploratory experiments that involve the manipulation of timestamps found in SQLite databases on Android smartphones. While conducting the experiments, the changes occurring on the Android smartphone are observed. Based on those observations, specific heuristics are identified that may indicate the manipulation of timestamps. These heuristics form a new solution that allows examiners to verify the authenticity of timestamps collected from traces found on Android smartphones. The new solution, called the Authenticity Framework for Android Timestamps or AFAT, verifies the authenticity of timestamps by following a practical methodology, which involves two methods. The first method identifies the presence of certain changes in the Android file system, which are indicators of the manipulation of the SQLite databases. The second method subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The immediate challenges to address are the following: (a) effective manipulation of timestamps found in SQLite databases on Android smartphones and (b) verifying that the authenticity of these timestamps have been compromised by using the methods of AFAT. The current paper provides preliminary evidence that, in terms of the challenges identified above, the suggested approach shows potential.

The remainder of the paper is structured as follows. Section II briefly describes the architecture of Android and the internal structure of SQLite databases. Section III presents the process followed to manipulate the timestamps and provides a detailed description of AFAT. Section IV offers an in-depth discussion of AFAT and describes potential future work. The conclusions are made in Section V.

II. Background

With the continuous growth in functionality of Android smartphones, increasing number of people make use of these devices during their daily activities. For the traces collected by Android smartphones to be of use during digital investigations, a comprehensive understanding of the architecture of Android is required. An evaluation of SQLite is also required, since most of the traces found on Android smartphones are stored in SQLite databases. This section, therefore, provides a short introduction of the Android architecture and presents the internal structure of SQLite databases.

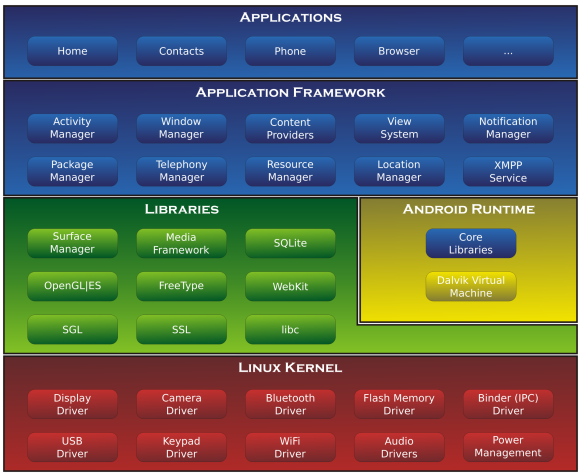


Fig. 1. Android Architecture [10]

A. Android's Architecture

Android is popular open source software architecture provided by the Open Handset Alliance [7] that is currently targeting mobile devices, such as smartphones and tablet computers. The Android software architecture (see Fig.1) is divided into five layers: Applications, Application Framework, Libraries, Android Runtime and the Linux Kernel [8]. The uppermost layer, Applications, provide access to a set of core applications. The Application Framework layer implements a software framework that reassembles functions used by existing applications. All available libraries are written in C/C++ and called through a Java interface. The Android runtime consists of a set of core libraries and a Dalvik virtual machine. The bottommost layer is the Linux kernel, which allows for interaction between the upper layers by means of device drivers [8], [9].

Until Android version 2.2 (*Froyo*), most Android smartphones used Yet Another Flash File System 2 (YAFFS2) [11]. YAFFS2 was developed in 2004 in response for larger sized NAND (Not-AND) flash devices [12]. With the release of Android version 2.3 (*Gingerbread*), the file system for Android devices switched from YAFFS2 to Fourth Extended (EXT4) file system [11]. YAFFS2 was developed with a single-threaded design, which may cause bottlenecks in devices released with a multi-core chipset. The EXT4 file system, which is one of the most used file systems in Linux, does not have this limitation and can run smoothly on multi-core devices. The disk space of the EXT4 file system is divided into logical blocks, which reduce management overhead and improves throughput [13]. The key features of the EXT4 file system promote the development of advance applications and functionalities.

The architecture of Android regularly improves to support more improved applications. It is therefore necessary to contin-

uously evaluate Android's architecture and remain up to date with the current changes.

B. SQLite Databases

SQLite is an open source software library that implements a lightweight Structured Query Language (SQL) database engine for embedded use [14], [15]. The lightweight design of SQLite does not require a separate server and thus allows for the quick processing of stored data by reading and writing directly to a disk file [16]. The main database file, *<database_name>.db* or *<database_name>.db3*, consists out of a complete SQL structure that includes tables, indices, triggers, and views [16]. To support the SQL structure, the main database file is divided into one or more pages and each page share the same size [17]. The first page of the main database file is called the header page and is composed of the database header and the schema table. The database header stores structural information and the schema table contains the table information of the database. The pages following the header page are structured as B-trees and store the actual data [18].

During transactions, SQLite stores additional information in a second file called either a rollback journal or write-ahead log (WAL) file [17]. The rollback journal is the default method of SQLite to implement an atomic commit and rollback. Beginning with SQLite version 3.7.0, the new WAL approach was introduced and allowed for improved speed and concurrent execution. The WAL approach preserves the original content in the main database file and appends changes to a separate WAL file (*<database_name>.db-wal*), which contains a header and zero or more WAL frames. Transferring the transactions from the WAL file to the main database file is call a "checkpoint". When a checkpoint occurs the updated or new pages in the WAL file are written to the main database file. The checkpoint operation leaves the WAL file untouched, allowing the WAL file to be reused rather than deleted [19]. SQLite does a checkpoint automatically when a file reaches a size of 1000 pages (about 4MB in size) [20].

SQLite databases are a popular choice for data storage in Android applications [14]. An Android application, which uses SQLite, separately includes the SQLite databases and this allows for reduced external dependencies and minimized latency [21]. A lot of events taking place on an Android smartphone generate valuable traces, for example: call history, SMS/MMS messages, e-mails (Gmail) and instant messages generated by Google Hangouts (previously Google Talk) as well as WhatsApp Messenger. A summary of the SQLite databases used to store these traces, as well as the location of these databases on an Android smartphone are provided in Table I. The examples used throughout the remainder of this paper focus on the SQLite database storing the SMS/MMS messages.

III. Detection of Manipulated Timestamps

Timestamps of traces found on Android smartphones are integral to digital investigations, especially if the owner of the smartphone participates in criminal activities. Collected timestamps allow the examiner to relate the traces to some physical event and, more importantly, establish a timeline depicting the chronological order of events. Due to the importance of timestamps in digital investigations, smartphone users, or even malicious applications, can alter timestamps to compromise the integrity of traces as evidence.

In order to manipulate timestamps found in SQLite databases, as well as, detecting the changes occurring due to the manipulation, access to the necessary files are required. All user-related data can be found in the */data* directory on an Android smartphone [22]. Access to this directory is not permitted by default and is only accessible by rooting the Android smartphone. The term rooting, which is similar to the jailbreaking of an iPhone, is often perceived as negative action [12]. Rooting an Android smartphone merely means to escalate the current rights to root access rights. Root access rights allow access to the root directory (*/*) and provide the necessary permissions to take root actions [12]. The technical process of rooting an Android smartphone is beyond the scope of this paper.

The remainder of this section describes the appropriate steps that must be followed to manipulate timestamps in SQLite databases and also introduces the Authenticity Framework for Android Timestamps. All of the experiments and analysis described below was performed on a rooted Samsung Galaxy S2, running Android version 4.1.2 (*Jelly Bean*).

A. Manipulation of Individual Timestamps in SQLite Databases

Access to the root directory, which is necessary to manipulate the timestamps in the SQLite databases, requires the enabling of the Universal Serial Bus (USB) debugging functionality [12]. Although the default setting for USB debugging is "disabled", going to Settings, selecting Developer options and touching the checkbox next to "USB debugging" will turn on this feature. Once USB debugging is enabled, interaction with the root directory can occur using the Android Debug Bridge (adb). Android Debug Bridge is a versatile command-line tool that communicates with a connected Android smartphone [23]. To allow for complete access to the root directory, adb is restarted with the command, *adb root*, to gain root permission. Full access, with the necessary permissions, has been established and it is now possible to manipulate the timestamps in SQLite databases.

Manipulation of timestamps proceeds through three individual phases: retrieve, manipulate, and return. The first phase retrieves the required SQLite database files from the Android smartphone. Since the sqlite3 command utility, which is required to interact with the SQLite databases, does not come pre-installed on Android smartphones [24], the SQLite databases must be transferred to the local computer. The command, *adb pull <remote><local>*, copies the specified file from the Android smartphone to the local computer [23]. It is necessary to repeat this command for both the main database file, as well as the associated WAL file, which cannot

TABLE I
USER DATA STORED IN SQLITE DATABASES

User Data	SQLite Database Location	Table
Call History	/data/data/com.sec.android.provider.logsprovider/databases/log.db	logs
Messages (SMS/MMS)	/data/data/com.android.providers.telephony/databases/mmssms.db	sms
E-mails (Gmail)	/data/data/com.google.android.gm/databases/mailstore.<name@gmail.com>.db	messages
Google Hangouts	/data/data/com.google.android.talk/databases/babel1.db	messages
WhatsApp Messenger	/data/data/com.whatsapp/databases/msgstore.db	messages

be edited directly. Retrieving both the main database file and the associated WAL file ensures that all the latest records are present. The list of commands is shown below:

- *adb pull /data/data/<application_package_name> /databases/<database_name>.db C:\<local_folder>*
- *adb pull /data/data/<application_package_name> /database<database_name>.db-wal C:\<local_folder>*

Manipulating the timestamps found in the copied SQLite database files is performed during the second phase. A script is created to act as a malicious application and randomly manipulate timestamps within the SQLite database. During the execution of the script the main database file is opened, allowing for a checkpoint to occur. Once the execution of the script is completed, only the main database file (*<database_name>.db*) remains and must be returned to the Android smartphone.

The final phase returns the modified SQLite database to the Android smartphone. The command, *adb push <local><remote>*, copies the specified file to the connected Android smartphone [23]. To prevent the changes from being over-written by the existing data in the *<database_name>.db-wal* file, the file must be removed. The first step is to start an interactive shell using *adb shell*, followed by *su*, which provides root permissions within the shell. The next command, *cd data/data/<application_package_name>/databases/*, change the current working directory to the directory containing the main database and associated WAL file. Using the command *rm <database_name>.db-wal* will delete the WAL file from the directory. For the changes to reflect on the Android smartphone, it is necessary to reboot the device.

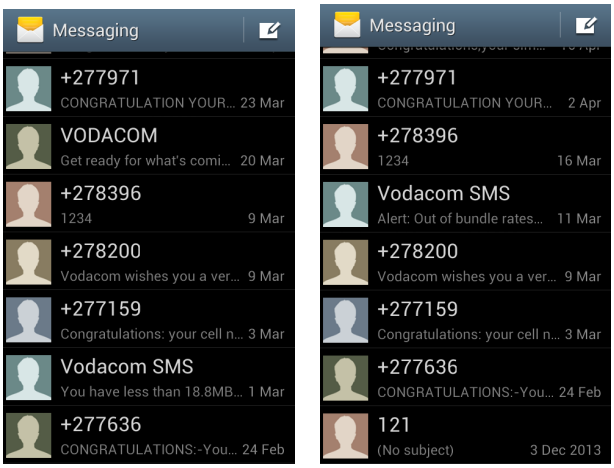
Fig.2 provides a snapshot of the existing SMS/MMS messages before and after the manipulation of the timestamps. The comparison shows significant changes to the dates of specific messages and a reordering of the messages. In order to verify that these changes are indeed due to manipulation, the methods of the Authenticity Framework for Android Timestamps must be applied.

B. Authenticity Framework for Android Timestamps

In this section the Authenticity Framework for Android Timestamps or AFAT is described. AFAT provides the examiner with a practical methodology that can be followed to verify the authenticity of timestamps. The focus of AFAT is primarily on timestamps collected from traces found on Android smartphones. The methods described in the remainder of this section verify the authenticity of timestamps found in SQLite databases. The first method identifies the presence of certain changes in the Android file system, which are indicators of the manipulation of the SQLite databases. The second step subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The presence of specific file system changes, as well as, inconsistencies in the associated SQLite databases indicates that the authenticity of the timestamps might be compromised.

1) Android File System Flags: Android File System Flags (AFS-Flags) are indicators of the potential tampering of SQLite databases on Android smartphones. Each AFS-Flag represents a change that occurs in the Android file system due to the modification or removal of a SQLite database or any other associated database files. The presence of any of these AFS-Flags is not an indication of the manipulation of timestamps but merely that the SQLite databases have been tampered with. The following four individual AFS-Flags offer guidance regarding the tampering of SQLite databases:

- **File permissions:** associated with the SQLite database files in the directory of a specific application are set to give only read/write access to the file owner and the group members. For each application, the current file owner and group members are only the individual application. Any modification or removal of a file within this directory will change the existing file permissions of the modified file from *-rw-rw—-* to *-rw-rw-rw-*. The following changes to the file permissions are therefore an indication of the possible manipulation of timestamps in the SQLite databases:
  - File permission of *<database_name>.db* file changed from *-rw-rw—-* to *-rw-rw-rw-*.
  - File permission of *<database_name>.db-wal* file changed from *-rw-rw—-* to *-rw-rw-rw-*.
- **Ownership:** of the SQLite databases is given to the specific application using the database. The file owner and group members are thus set to the user ID (UID) of the application, which is unique and specific to the application. The UID remains constant for the duration of the application on the particular Android smartphone. Modifications to any SQLite database files will result in a change of ownership and subsequently change the UID of both the file owner and group members. The following change to the ownership of the main database ifle is a possible indication of the manipulation the databases:
  - The current ownership for the file owner and group members of the *<database_name>.db* file changed from the current *UID* to *root*.
- **File Size:** of the main database file is expected to be



(a) Original timestamps (b) Manipulated timestamps

Fig. 2. SMS/MMS messages with original (a) and manipulated (b) timestamps

smaller than the size of the associated WAL file, since all new transactions are appended to the WAL file. The size of the main database file is only expected to grow after a checkpoint, when all the transactions from the WAL file is transferred to the main database file. A checkpoint, however, occurs only after the WAL file accumulated 1000 entries (leading to a file size of approximately 4MB), and thus the size of the main database remains relatively small. An automatic checkpoint occurs when the main database file is opened to allow for the manipulation of the timestamps. The WAL file must be deleted to prevent the changes made in the main database file from being overwritten by the existing content located in the WAL file. Once the Android smartphone is rebooted to reflect the changes, a new WAL file is automatically generated. This new WAL file contains limited data and thus has a file size that is smaller than the size of the main database file. A WAL file with a file size smaller than the size main database file is therefore an indication of the possible manipulation of that database.

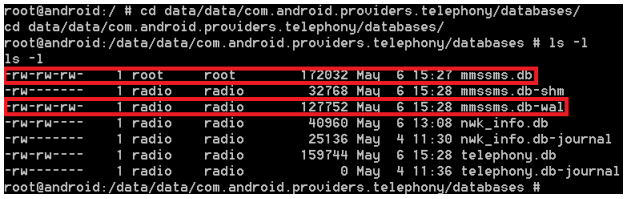
- **System Reboot:** is required for the changes made to the SQLite databases to reflect on the Android smartphone. The system reboot must occur after making the changes to the SQLite database. Therefore the timestamps of the files associated with a system reboot will follow after the timestamp that shows when the main database file was last modified. Multiple experiments revealed that the following files are indicators of a system reboot:
  - *rtc.log* file located in the */data/log/* directory.
  - *powerreset_info.txt* file located in the */data/log/* directory.
  - *SYSTEM_BOOT@[timestamp].txt* file generated in the */data/system/dropbox/* directory.
  - *event_log@[timestamp].txt* file generated in the */data/system/dropbox/* directory.

  Android log data are written to certain files in the */data/log/* directory [25]. Two files, *rtc.log* and *powerreset_info.txt*, are existing files in this directory. These files are updated with a new entry after every system reboot and every entry shows the boot time of the Android smartphone. The files, *SYSTEM_BOOT@[timestamp].txt* and *event_log@[timestamp].txt*, are located in the directory */data/system/dropbox/* [26], [27]. This folder is used by a service known as DropBoxManager (unrelated to the DropBox cloud storage service) and persistently stores chunks of data from various sources such as application crashes and kernel log records [26]. The *SYSTEM_BOOT@[timestamp].txt* file is generated consistently at boot time, with the timestamp forming part of the file name showing when the Android smartphone was booted. The other file, *event_log@[timestamp].txt*, is generated at 30 minute intervals and also indicates the time when the Android smartphone was rebooted. A system reboot occurring closely after the modification date of a SQLite database provides a possible indication of the modification of timestamps. A system reboot can, however, occur at any time after pushing the modified main database file onto the Android smartphone and is thus necessary to establish a time frame in which this particular AFS-Flag will be deemed reliable.

Fig.3 provides a comparison of the changes that occurred in the directory containing the SQLite database that stores the SMS/MMS messages. Fig.3 (b) indicates the existence of three AFS-Flags. The first AFS-Flag is the file permissions of both the *mmssms.db* and the *mmssms.db-wal*, which changed from *-rw-rw—-* to *-rw-rw-rw-*. The second AFS-Flag is the ownership for the both the file owner and group members of the main database file that changed from *radio* to *root*. The final AFS-Flag is the file size of the mmssms.db-wal, which is smaller than the size of the main database file, indicating that the mmssms.db-wal file was possibly deleted.

Fig.4 shows that the *rtc.log* and *powerreset_info.txt* files were last modified at 15:28 on May 6 and Fig.5 presents

(a) Before manipulation of the mmssms.db



(b) After manipulation of the mmssms.db

Fig. 3. Comparison of changes in the directory containing the SQLite database for the SMS/MMS application

the contents of a *SYSTEM_BOOT@1430918940293.txt*, which indicates that a reboot occurred at 15:29 on 6 May. All three files illustrate that a reboot occurred approximately at 15:28 on May 6, which follows after the last modified date of the main database file (15:27 on May 6). The existence of these files verifies that a system reboot occurred after pushing the modified main database file onto the Android smartphone.

The presence of all four AFS-Flags indicates the possible manipulation of timestamps within the SQLite database storing the SMS/MMS messages. Identification of these manipulated timestamps requires the analysis of the SQLite database for potential inconsistencies.

2) SQLite Database Inconsistencies: SQLite databases are the prominent choice for data storage in the Android OS. The association of one or more AFS-Flags with a specific SQLite database indicates the potential manipulation of the stored timestamps. Detection of the manipulated timestamps requires the further analysis of the SQLite database for any potential inconsistencies. An inconsistency in a SQLite database is described as a record that is listed incorrectly when ordered according to the following fields: primary key and a field containing dates or timestamps. The identification of inconsistencies in the tables of SQLite databases requires the evaluation of the above mentioned fields.

The tool selected to perform the evaluation is SQL. SQL is a powerful query language, allowing for the formulation of queries that can be of forensics use [28]. The evaluation of the tables available in the SQLite databases proceeds through three steps and use the following SQL statements: *CREATE TABLE*, *INSERT INTO*, and *SELECT*. To preserve the integrity of the data stored in the original table, a new temporary table is created using the *CREATE TABLE* statement. The purpose of the *CREATE TABLE* statement is to define the physical structure of the new temporary table [29]. The temporary table contains a primary key, which is an integer value that auto-increments, and all the fields that are necessary to identify the inconsistencies. The query to create the temporary table is as follows:

CREATE TABLE temp (new_id INTEGER PRIMARY KEY AUTOINCREMENT, original_id INTEGER, timestamps INTEGER);

Following the creation of the new temporary table is the population of this table with all the records currently located in the original table, which is being investigated. To perform this action, a combination of the *INSERT INTO* and *SELECT* statements are used. The *SELECT* statement selects all the records from the table currently under investigation while the *INSERT INTO* statement inserts these selected records into the temporary table. Continuing with the SMS/MMS SQLite database as an example, the SQL query required to copy the records from the *sms* table into the temporary table is as follows:

INSERT INTO temp (original_id, timstamps) SELECT _id, date FROM sms;

To locate any inconsistencies in the records collected in the temporary table, it is necessary to compare the values in the timestamps field of subsequent records. Since all the values in the timestamps field are expected to follow one another (each new record is appended at the end of the table), the difference between two subsequent values in the timestamps field must be smaller than or equal to zero. A positive difference is an indication of a timestamp that is out of order and the cause of this inconsistency is the manipulation of the timestamp. The SQL query used to detect the records that are inconsistent is as follows:

SELECT T1.original_id, T1.timestamps, (T1.timestamps - T2.timestamps) AS difference FROM temp T1, temp T2 WHERE T2.new_id = T1.new_id + 1 AND difference>0;

Applying this SQL query to the records in the temporary table leads to the identification of multiple inconsistencies in the SMS/MMS SQLite database. The inconsistent records are listed in Fig.6, showing the manipulated timestamps. The existence of these inconsistent records in the SMS/MMS SQLite database invalidates the authenticity of the database. The examiner must thus decide whether to exclude the manipulated records from the investigation or only focus the investigation around the manipulated records.

IV. Discussion and Future Work

The exploratory experiments performed while composing this paper showed that the timestamps found in SQLite databases can be manipulated by following the technique described in Section III-A. This technique is currently the most plausible technique to manipulate timestamps in SQLite

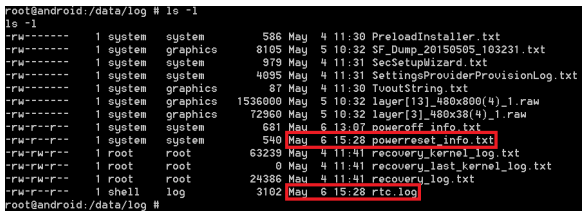


Fig. 4. The */data/log/* directory containing the *rtc.log* and *powerreset_info.txt* files

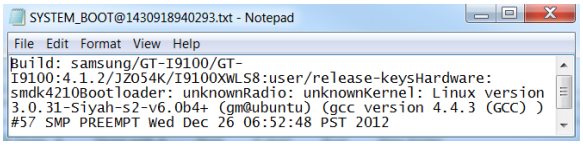


Fig. 5. The *SYSTEM_BOOT@[timestamp].txt* file generated after a reboot

RecNo	original_id	timestamps	difference
Click here to define a filter			
1	5	1426068243389	656800007
2	7	1426026953724	140410724
3	11	1426480116100	203700427
4	17	1427977953901	733420999

Fig. 6. Inconsistent records found in the *sms* table of the *mmssms.db* SQLite database

databases. Although other techniques can be designed, the inability to directly alter the data in the WAL file and the unavailability of the sqlite3 command utility on an Android smartphone will limit the capabilities and impact the efficiency of other techniques.

To establish the authenticity of timestamps found in SQlite databases and detect the potentially manipulated timestamps, this paper introduced the Authenticity Framework for Android Timestamps. The framework consists of two methods that determine the authenticity of timestamps by evaluating the file system for specific changes and identifying inconsistencies in the SQLite databases. AFAT is independent of an Android smartphone and does not require any prerequisites to be installed on the Android smartphones prior the investigation. The purpose of AFAT is to give examiners an indication of whether the timestamps collected in SQLite databases were tampered with. The results presented by AFAT allow the examiner to establish the authenticity of the timestamps. Based on the authenticity of the timestamps, the examiner can decide to either include or disregard the evidence, associated with the evaluated timestamps, in the investigation. AFAT is therefore capable to save crucial time during the investigation and allow the examiner to arrive at correct and accurate conclusions. The experiments provided throughout this paper showed that AFAT is capable of providing the examiner with the necessary support to establish the authenticity of timestamps in SQLite databases.

The successful application of AFAT depends, however, on the following two external factors. Firstly, the skills of the smartphone user or the sophistication of the malicious application performing the manipulation can influence the availability of the AFS-Flags. Smartphone users or the malicious application may be aware of the changes that occur due to the manipulation of timestamps in SQLite databases. To prevent detection, these changes can be removed or altered in an attempt to thwart the examiner performing the investigation. Secondly, the timeframe between the manipulation of the timestamps and the seizing of the smartphone can also influence the availability of certain AFS-Flags. An extended timeframe can cause specific AFS-Flags (such as File size and System reboot) to be deleted or be overwritten by the Android OS. Besides these limitations, AFAT still provides an adequate framework for determining the authenticity of timestamps found in SQLite databases on Android smartphones. AFAT is also the first solution provided that is capable of establishing the authenticity of timestamps after the Android smartphone has been seized.

The current implementation of AFAT focuses only on detecting the manipulation of timestamps found in SQLite databases. Manipulation of timestamps can, however, occur at multiple locations. The time zone settings of an Android smartphone, which can be set incorrectly or be changed (intentionally or unintentionally), can influence the accuracy of the timestamps found in SQLite databases. Besides the time zone settings, the actual time of the Android smartphone can also be manually adjusted by disabling the automatic time synchronisation feature. Manual adjustments of the time will impact the timestamps that are generated for the traces stored in the SQLite databases when certain events occur on the Android smartphone. It is therefore necessary to incorporate the evaluation of the time zone and time settings of an Android smartphone into existing solution provided by AFAT. Future work will thus focus on identification of the necessary methods required to establish the accuracy of the time zone and time settings of a seized Android smartphone. Subsubsection text here.

V. Conclusion

Evidence found, in the form of traces, on smartphones form an important asset of digital investigations. The timestamps associated with the traces allow the examiner to construct a timeline of events. Such a timeline often forms the basis for further investigation and has the ability to provide answers to certain questions. Due to the importance of timestamps, it is necessary for examiners to be able to verify their authenticity. Collected timestamps might be incorrect due to tampering and without additional verification, the timestamps will lead the examiner to make unreliable conclusions. To verify the authenticity of timestamps found in SQLite databases, this paper introduced the Authenticity Framework for Android Timestamps, which establishes the authenticity by following two methods. The first method identifies the presence of certain changes in the Android file system, which are indicators of

the manipulation of the SQLite databases. The second method subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The existence of specific Android file system changes as well as inconsistencies in existing SQLite databases indicates that the authenticity of the timestamps might be compromised. The challenges addressed in the paper were to show that (a) timestamps can be manipulated in SQLite databases and (b) identifying that the authenticity of these timestamps has been compromised. Challenge (a) was addressed by showing the process that must be followed to successfully manipulate timestamps in SQLite databases and challenge (b) was addressed by applying the methods of the Authenticity Framework for Android Timestamps. The current paper provides preliminary evidence that the suggested approach shows potential and future work will focus on expanding the framework.

REFERENCES

[1] "Gartner says smartphone sales surpassed one billion units in 2014," 2015, available online: [http://www.gartner.com/newsroom/id/2996817] (Accessed: 7 Apr. 2015).

[2] S. Radicati, "Mobile statistics report, 2014-2018," The Radicati Group, Inc., Tech. Rep., 2014.

[3] P. Albano, A. Castiglione, G. Cattaneo, and A. De Santis, "A novel anti-forensics technique for the Android OS," in *International Conference on Broadband and Wireless Computing, Communications and Applications*, 2011, pp. 380–385.

[4] R. Harris, "Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem," *Digital Investigation*, vol. 3, pp. 44–49, 2006.

[5] R. Verma, J. Govindaraj, and G. Gupta, "Preserving dates and timestamps for incident handling in android smartphones," *Advances in Digital Forensics X*, vol. 433, pp. 209–225, 2014.

[6] J. Govindaraj, R. Verma, R. Mata, and G. Gupta, "Poster: iSecureRing: Forensic ready secure iOS apps for jailbroken iPhones," in *35th IEEE Symposium on Security and Privacy*, 2014.

[7] "Open handset alliance - android," 2015, available online: [http://www.openhandsetalliance.com/androidoverview.html] (Accessed: 9 Apr. 2015).

[8] C. Maia, L. M. Nogueira, and L. M. Pinho, "Evaluating Android OS for embedded real-time systems," in *6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2010, pp. 63–70.

[9] B. Speckmann, "The android mobile platform," Ph.D. dissertation, *Eastern Michigan University*, 2008.

[10] "Android architecture," 2015, available online: [http://developer.android.com/images/system-architecture.jpg] (Accessed: 14 Apr. 2015).

[11] C. Zimmermann, M. Spreitzenbarth, S. Schmitt, and F. C. Freiling, "Forensic analysis of YAFFS2," in *Sicherheit*, 2012, pp. 59–69.

[12] J. Lessard and G. Kessler, "Android forensics: Simplifying cell phone examinations," *Small Scale Digital Device Forensics Journal*, vol. 4, no. 1, pp. 1–12, 2010.

[13] H.-J. Kim and J.-S. Kim, "Tuning the Ext4 filesystem performance for Android-based smartphones," in *Frontiers in Computer Education*. Springer, 2012, pp. 745–752.

[14] F. Freiling, M. Spreitzenbarth, and S. Schmitt, "Forensic analysis of smartphones: The Android Data Extractor Lite (ADEL)," in *Proceedings of the Conference on Digital Forensics, Security and Law*, 2011, pp. 151–160.

[15] S. Jeon, J. Bang, K. Byun, and S. Lee, "A recovery method of deleted record for SQLite database," *Personal and Ubiquitous Computing*, vol. 16, no. 6, pp. 707–715, 2012.

[16] "About SQLite," 2015, available online: [https://www.sqlite.org/about.html] (Accessed: 14 Apr. 2015).

[17] "The SQLite database file format," 2015, available online: [https://www.sqlite.org/fileformat.html] (Accessed: 14 Apr. 2015).

[18] P. Patodi, "Database recovery mechanism for Android devices," *Diss. Indian Institute of Technology, Bombay*, 2012.

[19] A. Caithness, "The forensic implications of sqlite's write ahead log," 2012, available online: [http://www.cclgroupltd.com/the-forensic-implications-of-sqlites-write-ahead-log/] (Accessed: 14 Apr. 2015).

[20] "Write-ahead logging," 2015, available online: [https://www.sqlite.org/wal.html] (Accessed: 14 Apr. 2015).

[21] A. S. Sharma, M. S. Malhi, M. Singh, and R. Singh, "CSLA an application using Android capstone project," in *MOOC, Innovation and Technology in Education (MITE)*. IEEE, 2014, pp. 382–385.

[22] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli, "Smartphone malware evolution revisited: Android next target?" in *4th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2009, pp. 1–7.

[23] "Android Debug Bridge," 2015, available online: [http://developer.android.com/tools/help/adb.html] (Accessed: 13 Apr. 2015).

[24] "How to use ADB in Android studio to view an SQLite DB," 2014, available online: [http://www.codeitive.com/0QijPqVeXU/how-to-use-adb-in-android-studio-to-view-an-sqlite-db.html] (Accessed: 13 Apr. 2015).

[25] R. Johnson and A. Stavrou, "Resurrecting the read_logs permission on samsung devices," in Black Hat Asia 2015, 2015.

[26] "Dropboxmanager," 2015, available online: [http://developer.android.com/reference/android/os/DropBoxManager.html] (Accessed: 16 Apr. 2015).

[27] M. Kaart and S. Laraghy, "Android forensics: Interpretation of timestamps," *Digital Investigation*, vol. 11, no. 3, pp. 234–248, 2014.

[28] H. Pieterse and M. Olivier, "Smartphones as distributed witnesses for digital forensics," *Advances in Digital Forensics X*, vol. 433, pp. 237–251, 2014.

[29] E. Ugboma, *Learn Database Programming using SQL of MS Access 2007*. BookSurge Publishing, 2009.