ABSTRACT

Rao, Karthik. M.Tech. Department of Cyber Security, Amrita Vishwa Vidyapeetham, 2015. Forensic Support to Android Device.

The growing advanced features of the smartphones catch the attention of criminal to perform criminal activity. Information placed in a smart phone is plays a crucial role the course of a criminal investigation and while presenting in court of law. An ordinary man with an ordinary mobile device easily gives away his/her contact details, call history and SMS, just by looking at the device. Smart phones contains even more useful information such as social network messages, E-mails, network connections, browser history etc. Such a device in hands of professional criminal is a bad news, but becomes a good news to a forensic invigilator. A knowledgeable criminal might modify or wipe the traces of activities and the mobile data. This data is harder to retrieve once the user deletes it. We propose forensic support to the Android ROM, which monitors all the user activities and stealthily stores it in the cloud. Other than just activity this thesis propose in adding a keylogger and call tapping facility.

Keywords: Android, Forensics, Android Framework, adb tool, Binder

Contents

List of Tables

List of Figures

1

Proposed System

Here, we are proposing of adding a forensic support service to the Android ROM[Mateti et al. 2015]. This service proactively identifies, prioritizes, collects and secretly stores the forensically sound data initially on a stealth file system within the device, that is opportunistically uploaded to a cloud server. This section is an overview of the architecture Figure 3.1. The next section describes a design and an initial implementation.

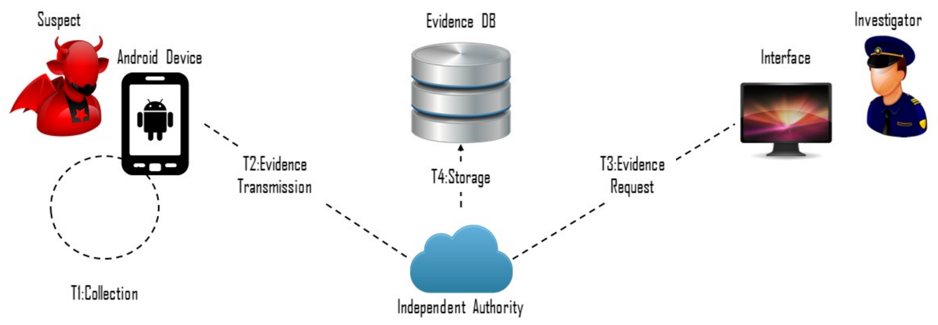

Figure 1.1: Proactive Forensic Support Architecture

The architecture contains four phases. (i) Data collection, (ii) Transmission of the Evidence, (iii) Evidence request for investigation, and (iv) Evidence storage

1.1 Architecture of Forensic module

The forensic module we proposed gets data mainly from two sources (Figure 3.2). The Primary source contains database files, API calls, and the file monitoring tool. Secondary source deals with keyloggers, call recording and Vital sensor details. This data gets stored in a separate file partition named `/forensic`, and this data resides in the device until gets the chance to upload it to the cloud server.

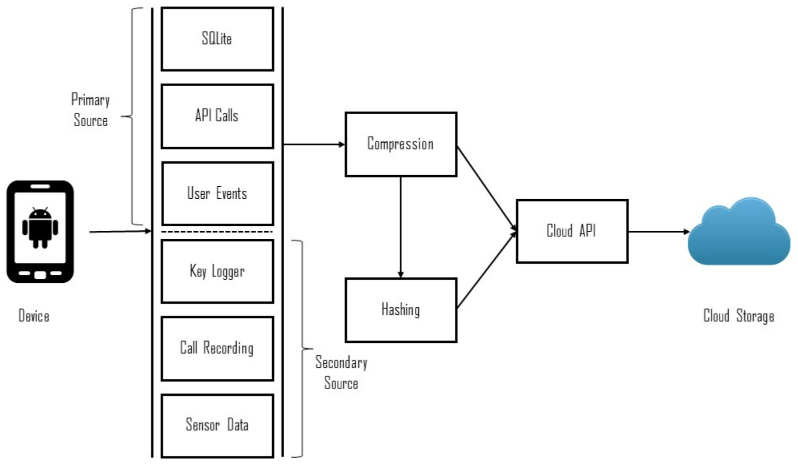

Figure 1.2: Detail archiecture of proactive forensic support

1.1.0.1 Primary Sources

Android applications store their private data using SQLite databases and these files are the main source of information. For example, phone logs are stored in the **contacts2.db** file. We collect these SQLite files in a scheduled manner so that no data is forgotten when investigating. User activities like camera, network usage, and GPS locations are also collected by using Android APIs Table 3.1. Filesystem events get captured by using a tweaked version of `inotify`[Wikipedia 2015a] tool.

User Data	**API**
Phonebook	ContactsContract.CommonDataKinds.Phone.NUMBER
Call History	android.provider.CallLog.Calls.CONTENT_URI
Camera	android.permission.CAMERA
Alarm Clock	android.provider.AlarmClock
Bluetooth	android.permission.BLUETOOTH
GPS Location	android.permission.ACCESS_FINE_LOCATION
Fingerprint	android.permission.USE_FINGERPRINT

Table 1.1: Selected Android APIs

1.1.1 Secondary Sources

These include keyloggers which collect the keystrokes on the device. Call recording/taping records the conversation between two or more people. The sensor data provides useful information like the motion, environmental, and position of the device and its surroundings.

1.1.2 Temporary Storage

The collected data is stored on a newly created device partition named `/forensic` until it gets uploaded to the cloud. The partition is root owned and this cannot be accessed by any normal applications. The data stored are opportunistically uploaded to the cloud server using a cloud API residing on the device.

1.1.3 Evidence Transmission

The data upload is done only if the device has the charge of 60% or more or it is in charging mode. The device uploads the data in the WiFi session because if the forensic module uses the mobile data, the user might/will get concerned. The battery, as well as the processor usage, might attract the user attention, so the cloud upload is only done when the device is in the idle state.

The data stored in the device accumulate a lot of space which eventually hinders the performance which causes the user to be suspicious. So these data are compressed into smaller units and then uploaded. To verify these transfers are bound to integrity, we are hashing these units. Both are explained in detail in upcoming sections.

1.1.4 Real Time Analysis

Most of the Android applications encrypt their data and databases. Forensic data can be collected only by analysing or decrypting those application data. We do not propose real-time analysis be done on the Android device because of the processing speed, consumption of the battery and slow down of the system. The data can be collected even the absence of the device' physical location. Location of the device is tracked using the GPS location. The data uploaded are tracked with the timestamps to provide provenance of the device to the investigator. Also real time upload is possible if the device if it is in the range of WiFi with sufficient battery backup.

The Evidence present in the cloud can be requested by the investigator, on a PC, to investigate. This can be easily downloaded or copied to the system for further analysis. The files will be smaller units of the compressed file and along with a hash value of it. Investigator before proceeding can match the value.

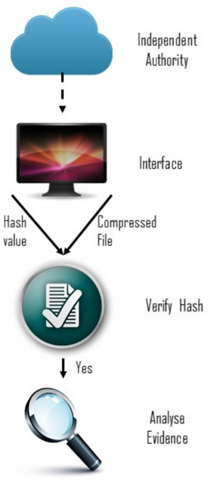

Figure 1.3: Detail archiecture of proactive forensic support

1.1.5 Data Storage Phase

The data present in the cloud should be transferred to the separate DB called as Evidence Database to safeguard evidence. Here we are not working on this piece of architecture. This has been written for completion purpose and may be for the future work.

1.2 Hiding Process

There are several process listing commands and APKs to list running processes in Linux systems. These tools use procfs filesystem to get these data. The forensic process is hidden from the user by using `hidepid` option. The values are as follows:

`hidepid=0` - The classic mode - anybody may read all world-readable `/proc/<pid>/*` files (default).

`hidepid=1` - denotes users may not access any `/proc/<pid>/` directories, but their own. Sensitive files like cmdline, sched*, status are now protected against other users.

`hidepid=2` - denotes `hidepid=1` plus all `/proc/<pid>/` will be invisible to other users. It doesn't signify whether a process with a specific `pid` value exists, but it hides process' `uid` and `gid`, which may be learned by `stat()`'ing `/proc/`<`pid`>/ otherwise[GITE 2014].

1.3 Keystroke logging

Keystroke logging is the action of recording (logging) the keys struck on a keyboard without making the person using the keyboard aware that their actions are being monitored.

1.3.1 Using Swiftkey

SwiftKey Keyboard is among the top paid app in the Play store at the moment and it's a great app. Malware-ridden Android app is taking a serious risk of a keylogger being inserted[Casey 2013] and people tracking all their passwords, Google searches, and Credit Card numbers. Basic knowledge of Java is sufficient to create an app and make it as default keyboard.

1.3.2 Using Man-in-the-Binder

The Man-In-The-Binder exploit was described in [Artenstein and Revivo 2014]. This attack is analogous to the man-in-the-middle attack, where the attacker code stands in the middle of a communication. The traditional approach is to replace a malware with built in keyboard. This can be easily detected, even by a normal user. Using a Man in the Binder attack would be a more robust and stealthy solution. To receive keyboard data, an application has to register with an Input Method Editor (IME) server. The default IME in most Android is `com.android.inputmethod.latin`. We no longer need to intercept the data going down from the application into Binder - we have to intercept the reply tossed up from Binder to the app.

It has not been decided which approach to use. Completeness of work on both is still under progress.

1.4 Call Recording

Telephone tapping is the monitoring of telephone and Internet conversations by a third party, often by covert means. The recording will be saved as .amr (`MediaRecorder.OutputFormat.RAW_AMR`) files and are located in the folder "recordedCalls" in /forensics storage.The Android SDK includes a `MediaRecorder` class, which one can leverage to build audio recording functionality. Doing so enables a lot more flexibility, such as controlling the length of time audio is recorded for.

The `MediaRecorder` class is used for both audio and video capture. After constructing a `MediaRecorder` object, to capture audio, the `setAudioEncoder` and `setAudioSource` methods must be called. Additionally, two other methods are generally called before having the `MediaRecorder` prepare to record. These are `setOutputFormat` and `setOutputFile.` `setOutputFormat` allows us to choose what file format should be used for the recording and `setOutputFile` allows us to specify the file that we will record to.

1.5 Compression

The data, in this era where TB's of data are carried in a pocket, piles up when it's not transmitted or some huge data is captured like a video or bunch of pics. This usually causes the slowdown of the device and a user can easily notice the lag. So as a simple solution we compress this data as much as possible to save disk space. Android provides an algorithm called `Deflater()`. A new Deflater instance using the default compression level will be constructed. This can be found in `java.util.zip` package.

The zip file will be created not as single but as multiple files of small size. The size may be around 5 Mb. These files are hashed to maintain the integrity and later are uploaded to the cloud along with the hash file.

1.6 Sensor Details Capture

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful in monitoring the device. With acute observation one can analyse what exactly user was up to.

The Android platform supports three broad categories of sensors:

- Motion sensors
- Environmental sensors
- Position sensors

To monitor raw sensor data you need to implement two callback methods that are exposed through the `SensorEventListener` interface: `onAccuracyChanged()` and `onSensorChanged()`.

1.7 Hashing

The collected files are compressed, divided into smaller chunks and hashed. Each compressed file and hash are sent to the cloud server to maintain the integrity. The chunks of compressed are hashed. These hash can be calculated based on the functionality available in Java.

```
java.security.MessageDigest.getInstance("MD5");
```

1.8 Real-Time Data Collection

The Data Collection module of the forensic service collects the forensic relevant data and stores them in a tiered stealth file volume. Two companion projects help us. The Network Ombudsman[George 2015] project gathers all (wlan0, eth0, Bluetooth, 2G/3G/4G/LTE, NFC, etc) network events. The Android Audit[Kamardeen 2015] project logs all non-network events. Both uploads to cloud storage.

1.9 Stealth Challenges

Since the forensic module utilizes additional device storage space, Internet connection and some amount of processor speed. There should be a covert operation to hide these activities from the user. The main challenge is building an undisclosed partition. Other is to hide the forensic process from process listing. Cloud upload is also a challenging job and this has to be done in an opportunistic way so that the user must not concern or notices it.

Stealth: The device that has forensic support enabled should not be visible or interrupt the owner while performing regular actions. Only APKs with root permissions can detect the forensic partition. A major future goal is to make this detection very difficult.

Hiding the forensic service is not easy. Linux internals transact with `proc` and `sysfs` needs to be altered. This can be done exploring `hidepid` command.

Opportunistic Uploads waits for an opportune moment to transfer the forensically sound data i.e, availability of the WiFi. There is always a risk of the device being running out of memory so to mitigate this the data is deleted from the device as soon as it is sent to the cloud.

References

AIYYAPPAN, P. 2015. Android forensic support framework. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti, `http://cecs.wright.edu/~pmateti/GradStudents/index.html`.

AKARAWITA, I. U., PERERA, A. B., AND ATUKORALE, A. 2015. Androphsy–forensic framework for android. In *International Conference on Advances in ICT for Emerging Regions (ICTer)*. Vol. 250. 258.

ARRIGO, B. A. 2014. *Encyclopedia of Criminal Justice Ethics*. SAGE Publications.

ARTENSTEIN, N. AND REVIVO, I. 2014. Man in the binder: He who controls ipc, controls the droid. *BlackHat Europe*.

BORELLO, G. 2014. Hiding linux processes for fun and profit. `https://sysdig.com/hiding-linux-processes-for-fun-and-profit/`.

CASEY, G. 2013. Inserting keylogger code in android swiftkey using apktool
. `http://www.georgiecasey.com/2013/03/06/inserting-keylogger-code-in-android-swiftkey-u`

CCLUSER. 2012. Android pattern lock scripts. `http://www.cclgroupltd.com/product/android-pattern-lock-scripts/`.

EASTTOM, C. AND MURPHY, G. 2015. *CCFP Certified Cyber Forensics Professional All-in-One Exam Guide*. McGraw-Hill Education.

GEORGE, N. 2015. Network Ombudsman for Android. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. Advisor: Prabhaker Mateti.

GITE, V. 2014. Linux: Hide processes from other users. `http://www.cyberciti.biz/faq/linux-hide-processes-from-other-users/`.

GROVER, J. 2013. Android forensics: Automated data collection and reporting from a mobile device. *Digital Investigation 10*, S12–S20.

HOOG, A. 2011. *Android forensics: investigation, analysis and mobile security for Google Android.* Elsevier.

INC., I. R. 2015. Smartphone os market share, 2015 q2. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

KALADHARAN, Y., MATETI, P., AND JEVITHA, K. 2016. An encryption technique to thwart android binder exploits. In *Intelligent Systems Technologies and Applications.* Springer, 13–21.

KAMARDEEN, J. 2015. Auditing Android system for anomalous behavior. M.S. thesis, Amrita Vishwa Vidyapeetham, Ettimadai, Tami Nadu 641112, India. Advisor: Prabhaker Mateti; `http://cecs.wright.edu/~pmateti/GradStudents/index.html`.

MATETI, P., AIYYAPPAN, P., GEORGE, N., KAMARDEEN, J., SAHADEVAN, A. K., AND SHETTI, P. 2015. Design and construction of a new highly secure Android ROM. Tech. rep., Amrita Vishwa Vidyapeetham, Ettimadai, Tamil Nadu 641112, India. 6. Advisor: Prabhaker Mateti; `http://cecs.wright.edu/~pmateti/GradStudents/index.html`.

MCGOVERN. 2012. Inotifywait for android. `https://github.com/mkttanabe/inotifywait-for-Android`.

RAJA, H. Q. 2011. Android partitions explained: boot, system, recovery, data, cache and misc. `http://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/`.

SAHU, S. 2014. An analysis of whatsapp forensics in android smartphones. *International Journal of Engineering Research 3,* 5, 349–350.

TAMMA, R. AND TINDALL, D. 2015. Learning android forensics.

WIKIPEDIA. 2012. Android's architecture diagram. `https://en.wikipedia.org/wiki/Android_(operating_system)#/media/File:Android-System-Architecture.svg`.

WIKIPEDIA. 2015a. inotify. `https://en.wikipedia.org/wiki/Inotify`.

WIKIPEDIA. 2015b. Mobile device forensics. `https://en.wikipedia.org/wiki/Mobile_device_forensics`.

YAGHMOUR, K. 2013. *Embedded Android: Porting, Extending, and Customizing.* " O'Reilly Media, Inc.".