A Study of User Data Integrity During Acquisition of Android Devices

By

**Namheun Son, Yunho Lee, Dohyun Kim,
Joshua I. James, Sangjin Lee and Kyungho Lee**

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2013 USA

Monterey, CA (Aug 4th - 7th)

Contents lists available at SciVerse ScienceDirect

Digital Investigation

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

A study of user data integrity during acquisition of Android devices

Namheun Son[a], Yunho Lee[a], Dohyun Kim[a], Joshua I. James[b], Sangjin Lee[a], Kyungho Lee[a,*]

[a] Center for Information Security Technologies (CIST), Korea University, Anam-Dong, Seongbuk-Gu, Seoul 135-713, Republic of Korea
[b] UCD Digital Forensic Investigation Research Group (DigitalFIRE), University College Dublin, Belfield, Dublin 4, Ireland

Keywords:
Digital forensic investigation
Android forensics
Data acquisition
Data integrity
Android Recovery Mode
JTAG

A B S T R A C T

At the time of this writing, Android devices are widely used, and many studies considering methods of forensic acquisition of data from Android devices have been conducted. Similarly, a diverse collection of smartphone forensic tools has also been introduced. However, studies conducted thus far do not normally guarantee data integrity required for digital forensic investigations. Therefore, this work uses a previously proposed method of Android device acquisition utilizing 'Recovery Mode'. This work evaluates Android Recovery Mode variables that potentially compromise data integrity at the time of data acquisition. Based on the conducted analysis, an Android data acquisition tool that ensures the integrity of acquired data is developed, which is demonstrated in a case study to test tool's ability to preserve data integrity.

1. Introduction

The number of Android device users in the third quarter of 2012 is reported to be 181 million (Android Market Share, 2012). This figure represents approximately 75% of smartphone users. As the number of Android device users increases, so too does the number of available applications. There are currently over 600,000 applications in the Play Store, which is the Android market where users can download Android applications.

Dottech (Android App Store, 2012) states that approximately fifty applications are installed on one Android device. As the number of installed application increases, more user data is stored in the device. Such user data is potentially relevant to an investigation when an Android device is examined during a criminal or civil investigation.

During an investigation involving Android devices, a number of smartphone forensic tools may be used. The methods of acquiring data from smartphone forensic tools are largely divided into either physical acquisition or logical acquisition methods. However, such methods of data acquisition do not normally verify the alteration to user data. If integrity of the collected user data is not protected, evidential data used in the investigation could be compromised.

This study will evaluate the Android Recovery Mode acquisition method proposed by Vidas et al. (2011) to determine whether user data integrity is maintained at the time of acquisition. This work will study the various Android Recovery Mode variables, and each variable's effect on the integrity of user data during acquisition of an Android device.

2. Related work

This section examines prior work regarding methods of data acquisition from Android devices.

Data acquisition from an Android device can be largely divided into logical acquisition and physical acquisition

* Corresponding author. Tel.: +82 10 5002 5099; fax: +82 2 928 9109.
E-mail addresses: kevinlee@korea.ac.kr, klee@secubase.com (K. Lee).

methods. The types of logical acquisition are partition imaging, copying files/folders, content provider (Hoong, 2011), and Recovery Mode (Vidas et al., 2011), the last of which is the focus of this study.

For data partition imaging that includes user data or for file/folder copying, the device must first be "rooted", that is, administrative privilege must be gained. For such data acquisition, the device must be booted normally. However, when a device is booted normally, integrity of user and unallocated data areas may not be ensured.

Similarly, using a content provider to acquire data also requires the device to be booted normally, and an application should be installed for acquisition to take place. Because of the need to install software on the device, this method could potentially be more destructive than rooting the device while on.

When Android Recovery Mode is used, administrator privileges can be granted while the device is in a state where the alteration to user data can be minimized.

During a physical acquisition, JTAG (Kim et al., 2008; Breeuwsma, 2006) is commonly used, as well as a method that uses chip-off (Jovanovic, 2012). Acquisition using JTAG is only possible when a JTAG debug port is identified on the embedded device. For this method, the battery, case and any connected cables should be removed, then the JTAG debug port and cable should be connected for communication (fundamentally soldering). This method appears to result in a full acquisition of device data; however, this method may take a long time to acquire selected data.

To the author's knowledge, other studies to determine whether or not the JTAG method guarantees data integrity has not been conducted. This work will use the JTAG method to verify the integrity of the Recovery Mode acquired data. By using both methods this work will attempt to determine whether or not integrity is guaranteed at the time of data acquisition by using JTAG through comparison with Recovery Mode-acquired data.

The chip-off method directly separates memory that is attached to the board of an embedded device, and once it is separated the embedded device, the device is very unlikely to be repaired. Also, memory could get damaged in the process of separating it.

Additionally, there is a method of acquisition using Live SD (Chen and Yang, 2011). The study of a Live SD acquisition applied the existing Live CD/USB concept to an SD card. Recovery Mode was also used in this method. In order to use this method the device should support memory expansion through an SD device, but not every Android device supports this.

The UFED tool by Cellebrite (2012) is a smart forensic tool that consists of an acquisition terminal and software analyzer. It acquires data by connecting a smartphone to the acquisition terminal. In order to acquire data, the device must be rooted. This is normally achieved by executing an exploit that has been independently developed. UFED supports file and folder acquisition or a whole partition unit. Since UFED is unable to target a specific partition for acquisition, it could take longer than acquiring only the user data partition.

The XRY tool from Micro Systemation (Micro Systemation XRY, 2012) also acquires data by connecting the device to a terminal. XRY supports acquisition of several devices at the same time using this terminal. However, for an Android device, data acquisition is possible only when rooted.

These tools have integrity check functions like file hash calculation after acquisition. But these functions can't ensure the integrity of user data partition before acquisition.

3. Background

NAND flash memory is used in order to save data in an Android device. NAND flash memory consists of three elements: block, page, and spare. Page is constructed with actual data and spare, and several pages build one block. In the spare area, it either saves metadata of the file system on NAND flash or information required when the hardware controller combines pages. Since the lifespan of Read/Write is determined for NAND flash, the controller controls writes to memory so that the entire NAND flash can be used as uniformly as possible. Such role is called 'Wear Leveling' (Han, 2000).

When data is divided into several pieces and saved equally in NAND Flash and the relevant data is accessed, it is necessary to recombine the data. At that time, the controller combines pieces of data into one continuous group of data based on metadata that is in spare.

NAND Flash uses two methods for handling data access: Memory Technology Device (MTD) and Flash Translation Layout (FTL).

MTD is a device driver that is used to directly access NAND Flash storage.

Yet Another Flash File System 2 (YAFFS2), which was used as the default in Android, is a file system for NAND Flash. In order to construct the file system, metadata of file system is recorded in the spare area of NAND Flash. Therefore, in case of YAFFS2, if the file system is not changed, the spare area does not change. Since the file system that uses MTD acquires a page area, but not the spare area. Because of this, methods that use MTD cannot interpret the file system when raw data is acquired. In order to acquire a file system that uses MTD, nanddump should be used, which enables imaging including the spare area.

FTL is a class that supports the file system for the existing block device so that NAND flash could be used as a block device (FTL (Flash Translation Layer), 1998). The Extended File System X (ExtX) and the Robust File System (RFS) are file systems for block devices that handle NAND flash using FTL. When handling NAND flash using FTL, the spare area cannot be reached, but since data used in the actual file system is not saved in the spare area, file system data can be acquired through an acquisition of the raw data.

Android uses file systems such as Ext2/3/4, YAFFS2, and RFS. Among them, Ext3/4 and YAFFS2 are journaling file systems. Journaling supports restoration when a system crash occurs. When a system runs normally without crashing, it executes commit, and records data in the area that is used for data storage (Journal File Systems, 2002).

Ext3/4 manages the Journal as a file, and YAFFS2 does not have an additional file.

The versions after Android 2.3 (Gingerbread) use Ext4 (Android 2.3 Gingerbread, 2012), and 83.2% of Android devices use Gingerbread or above as of Oct. 2012 (Android 2.3 over reaches 83 percent market share, 2012). This rate is likely to continue to increase as new Android devices are launched. This indicates that 83.2% of Android device users use Ext4 file system. Among the rest of Android devices, most of them use YAFFS2 or Ext3, and of these, only a few Samsung products use RFS. In other words, most Android devices use Ext3/4 for their journaling file system.

Ext3/4 has a journaling function, and this is managed as one file. The journaling function operates from the moment that partitions are mounted, and metadata is changed during this process. Due to journaling, the data of both the unallocated area and journal area can be altered. If the data that remained in the unallocated and journal areas was potential evidence in an investigation, the potentially relevant data could be altered (Kim et al., 2012). Table 1 is a part of the superblock structure of ExtX, which saves information such as free inodes count, free block count, write time, mount time, and mount count. Especially, for the free inodes count, free block count, etc, values change if the journaling function is activated, and unallocated space is altered according to the changed value (Carrier, 2005). The bytes 224–227 in superblock represent inode number where the journal is located.

However, when a partition is mounted as read-only, the metadata of the file system is not altered (Lee et al., 2005).

At the time of acquisition of an Android device, the user data partition is mounted in several situations. Therefore, the following work examines the method of either preventing or avoiding unwanted metadata modification of the file system. Also, an Android device must be turned off in order to confirm data integrity, so this work assumes that data acquisition began in a state where the targeted device was turned off.

4. Acquisition processes

This section examines the procedure and method of acquiring user data from an Android device while protecting data integrity, and then returning the device to the former state after completing data acquisition. This procedure is designed to minimize the time and additional work required to forensically acquire a suspect Android device.

The process is pictured in Fig. 1 with each step being explained below.

4.1. Prepare the custom recovery mode image

The first step is checking the model name of the target device. If the custom recovery mode image (CRMI) that can be used for the checked model is already made, move on to the next step, and if not, produce the CRMI that is suitable for the targeted model. The process of producing the CRMI and the relevant notes are as so:

- When booting Android, the boot media is explored, boot media is copied into RAM, the copied contents are interpreted, and then Android is booted (Hoong, 2011). In other words, the boot media (the CRMI in this study) is dependent on certain firmware. This implies that only one CRMI is required for each device's model.
- In order to approach user data by accessing through Android Debug Bridge (ADB). ADB allows a developer and forensic analyst to communicate and control an Android device over USB. ADB can be used as a method for data acquisition. A data acquisition method using ADB on an Android device is very common. The ADB service can be enabled through which root authority can be obtained (modify init.rc, replace ADB file …) (Vidas et al., 2011).
- After booting using CRMI, the rootfs partition that is in the area used as the root folder should be mounted in read/write mode (rootfs). This is because CRMI should be able to copy files to the rootfs partition. If rootfs is mounted in read-only mode, the mount mode can be changed after remounting, but when it remounts, the user data partition is automatically mounted. Some Android devices apply this method. Fig. 2 shows the file attribute (init.rc) of the basic recovery partition image of Galaxy Note (SHV-E160S), which often includes a command that mounts the rootfs partition as the read-only mode at the time of booting.
- A bootable image of an Android Device consists of a boot header, kernel, ramdisk and second stage. Recovery Mode Images are bootable. To make the CRMI, the ramdisk area is normally edited, but in some cases the kernel area is modified. Some kernel in Recovery Mode images include a function to mount the data partition. In this case the kernel area should be recompiled.
- The partition of the Android device is divided according to the role, and the size of each partition is calculated (Android Partitions, 2011). Mostly, the sizes of the recovery and boot partitions of an Android device are the same, but the size of the recovery partition is bigger because it includes a binary for using the device in recovery mode. However, since the actual CRMI should be used for the boot partition, the CRMI should be created according to the size of the boot partition. Table 2 shows the sizes of the boot and recovery partitions of an Android device that were confirmed.

The binary needed for data acquisition is included in the CRMI. It includes nanddump for imaging the file system that uses MTD like YAFFS2 and the busybox binary that includes two commands in order to send a file using netcat (Hoong, 2011). There is a method that extracts data with ADB push by saving the acquired image file in SD card, but it can take a long time for data acquisition. And since it is hard to apply to every type of Android device, it uses netcat at the time of data acquisition. However, the previous step indicates that the CRMI should be produced according to the size of the

Table 1
Data structure for the ExtX superblock.

Byte range	Description
12–15	Number of unallocated blocks
16–19	Number of unallocated inodes
24–27	Block Size
44–47	Last mount time
48–51	Last written time
52–53	Current mount count
88–89	Size of each inode structure
208–223	Journal ID
224–227	Journal inode

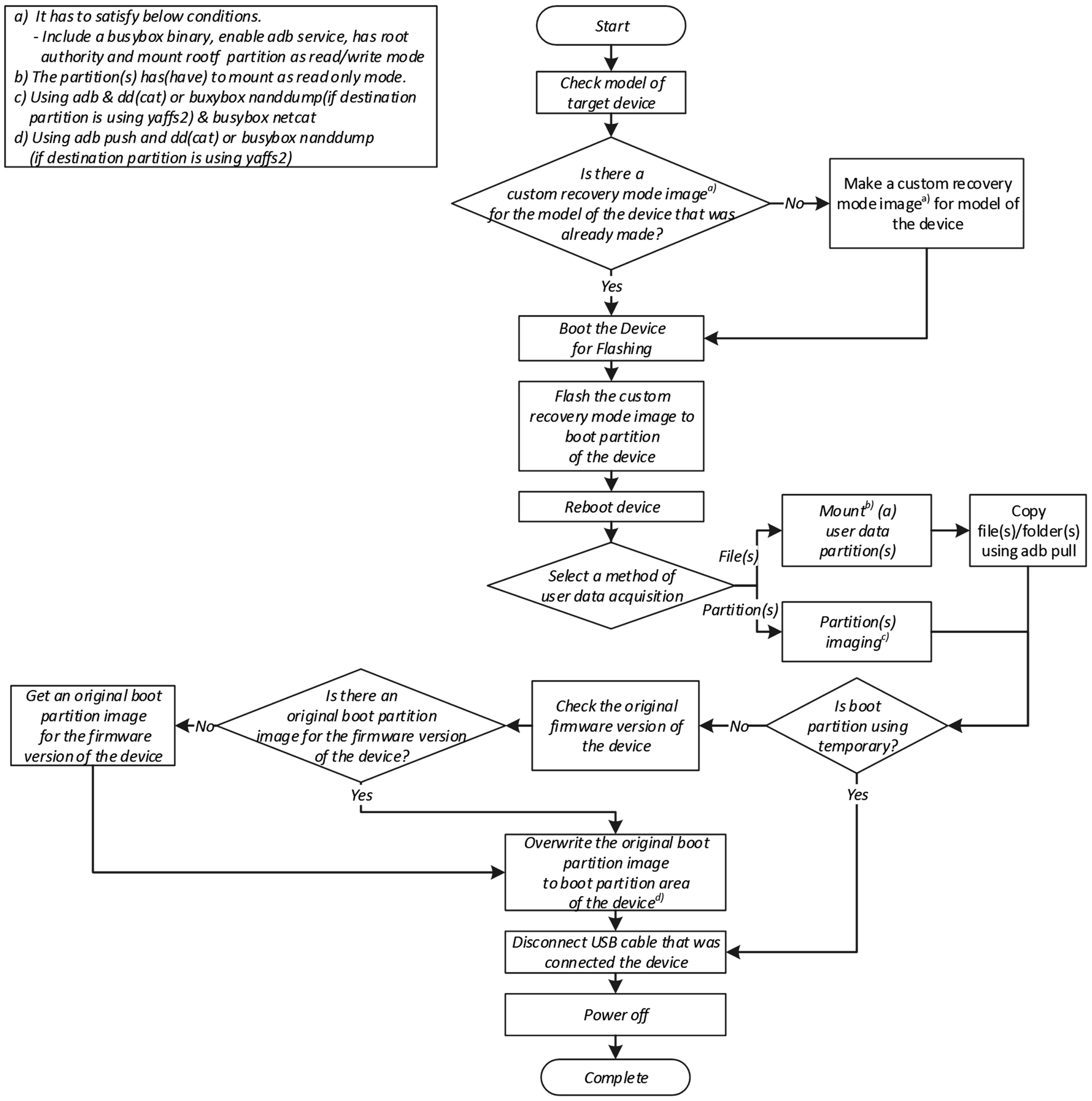

a) It has to satisfy below conditions.
- Include a busybox binary, enable adb service, has root authority and mount rootf partition as read/write mode
b) The partition(s) has(have) to mount as read only mode.
c) Using adb & dd(cat) or buxybox nanddump(if destination partition is using yaffs2) & busybox netcat
d) Using adb push and dd(cat) or busybox nanddump (if destination partition is using yaffs2)
Start
Check model of target device
Is there a custom recovery mode image a) for the model of the device that was already made?
No
Make a custom recovery mode image a) for model of the device
Yes
Boot the Device for Flashing
Flash the custom recovery mode image to boot partition of the device
Reboot device
Select a method of user data acquisition
File(s)
Mount b) (a) user data partition(s)
Copy file(s)/folder(s) using adb pull
Partition(s)
Partition(s) imaging c)
Is boot partition using temporary?
No
Check the original firmware version of the device
Is there an original boot partition image for the firmware version of the device?
No
Get an original boot partition image for the firmware version of the device
Yes
Yes
Overwrite the original boot partition image to boot partition area of the device d)
Disconnect USB cable that was connected the device
Power off
Complete

Fig. 1. Processes of acquisition and return to former state.

boot partition, and if the necessary binary is added at this step, the size could potentially exceed capacity. For such a case, the CRMI is produced without adding the binary, and the necessary binary could be added by using ADB push.

4.2. *Boot the device for flashing*

When the CRMI that can be applied to the targeted device is ready, the device can be booted in flash mode to flash the CRMI. The method for entering flash mode varies for each model (Vidas et al., 2011).

```
on post-fs
    mount rootfs rootfs / ro remount
```

Fig. 2. Read only option for rootfs (SHV-E160S).

If an Android device supports bootloader lock/unlock state, bootloader has to be unlocked before flashing the CRMI. That is because it is infeasible to flash custom images such as CRMI. The method to unlock bootloader varies depending on Android devices.

Moreover, there is an encrypted bootloader. Some Android devices take encrypted bootloader in order to prevent from unlocking like LG OPTIMUS VU product. In case of encrypted one, it is almost impossible to decrypt bootloader without the key.

4.3. *Flash the CRMI to boot partition of the device*

For this step, the CRMI was produced according to the size of the boot partition. When flashing the CRMI to the recovery partition the device should enter into the

Table 2
Partition size of Android devices.

Device	Size(Kb)	
	Boot partition	Recovery partition
Droid (A855)	3584	4608
Galaxy S2 (SHW-M250S)	8192	8192
Galaxy Nexus (SHW-M420K)	8192	12,224
Galaxy Note (SHV-E160S)	10,240	10,240
Galaxy S3 (SHV-E210S)	8192	8192
Galaxy Note 2 (SHV-E250S)	8192	8192
Vega LTE (IM-A800S)	10,240	10,240

Recovery Mode right after flashing is completed. However, the device must be manually entered into Recovery Mode, and if such booting in Recovery Mode fails, the device will proceed to boot normally. If that happens, the user data partition will be used, thus potentially damaging data integrity. On the other hand, if the CRMI is flashed to the boot partition, then the device could enter into the recovery mode right away without the need for manual control.

There are two methods of flashing the CRMI to the boot partition. The first method requires an investigator to check the file that is used as the boot partition in the firmware that each manufacturer provides. An investigator can change the CRMI file name to the file name that is used as boot partition before flashing. The second method allows flashing the device regardless of the CRMI file name by using a tool that is supported by the vendor. Table 3 shows the file names that are used as boot partitions in flash mode, and the firmware for each manufacturer.

Devices that use fastboot mode can be booted into Recovery Mode by directly using the CRMI at the time of booting without flashing it to the boot partition (*fastboot boot kernel_filename*), or it can be booted into Recovery Mode by flashing the CRMI to the boot partition (*fastboot flash:raw boot kernel_filename*).

4.4. User data acquisition

After successfully conducting CRMI flashing and rebooting the device, the device will automatically enter into Recovery Mode. With root authority in this mode an investigator can access the device by using the ADB command. From here, an investigator can acquire all the desired data.

Table 3
File name of boot partition in firmware.

Vendor	Flash mode	File name in firmware
Motorola	Bootloader	CG35.smg
HTC	Fastboot	–
Samsung	Odin	Boot.img
Samsung	Omap (Galaxy Nexus)	–
Pantech	Fastboot	–

First, the method of acquisition should be decided. There are essentially two acquisition methods: imaging a data partition and copying files. Since it is not necessary to mount the partition for the acquisition of the partition unit, the user data partition can be selected and copied remotely using netcat (Hoong, 2011). The user data partition's path on each device can be different. Some devices have a folder that has partition information, for example, the partition names and block names that were managed by the device, such as: /dev/block/platform/dw_mmc/by-name, /dev/block/platform/omap/omap_hsmmc.0/by-name.

If there is no folder that has partition information, the size of each partition should be checked (cat /proc/partitions). After that, part of the partition can be images and checked if it is a user data partition. When the file system in use is YAFFS2, which uses MTD, the partition can be imaged with nanddump. When the file system uses FTL, an investigator can acquire the partition using dd or cat.

For the acquisition of the file unit, the user data partition should be mounted, and, as previously mentioned, the partition should be mounted in read only mode in order to guarantee data integrity. When the partition is mounted, an investigator can acquire data by using ADB pull and netcat.

4.5. Return to former state

If the device is not booted right away using CRMI, the device can be returned to the former state after completing data acquisition. To continue acquiring data, this step can be skipped and acquisition can be completed as described later.

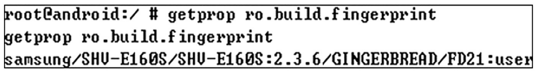
Since the modified data before the previous step is in the boot partition area, this area should be changed to the original boot partition. If a file for the boot partition exists, a check should be conducted to determine if it is the correct original boot partition. When checking if it is correct original boot partition, the firmware version that is used in the targeted device is important to note.

When the boot partition of the firmware has a different version, the Android kernel version or settings file might not be appropriate. In this case, the device might not boot normally.

The following command may be used in order to confirm the firmware information that is used in the general Android device: adb shell getprop ro.build.fingerprint. This command shows information about the Android device, but if the CRMI is in use, then it shows the information on the firmware that was used when making the CRMI.

In order to confirm accurate firmware information, the /system/build.prop file must be analyzed. This file is generated when flashing the firmware that was used previously. Fig. 3 shows what has been confirmed by using the prior command in recovery mode after flashing the CRMI of FD21 in a Galaxy Note (SHV-E160S) that uses the firmware version UH24. When the getprop command is used, the result shows that the current firmware version is FD21. Fig. 4 shows opening the /system.build.prop file, which stores the firmware information of the accurate UH24 version.

After confirming the appropriate original boot partition, an investigator may copy the boot partition to the device by



```
root@android:/ # getprop ro.build.fingerprint
getprop ro.build.fingerprint
samsung/SHV-E160S/SHV-E160S:2.3.6/GINGERBREAD/FD21:user
```

Fig. 3. Wrong results by getprop (Original Firmware Version: UH24; CRMI Version: FD21).

using ADB push, overwriting the original boot partition with the boot partition that uses the copied file in the targeted device. Table 4 includes the block names of the boot partition that were confirmed for each device.

Another method is flashing the original boot partition that is acquired by rebooting in the flash mode. However, this method takes additional time because it requires rebooting in the flash mode. Also, since the file format required for the flash mode for each manufacturer is different, this method requires converting the boot partition data into specific file formats that are supported by the vendor. Because this method is more difficult and time consuming, this work prefers the prior methods for reverting a device to its original state.

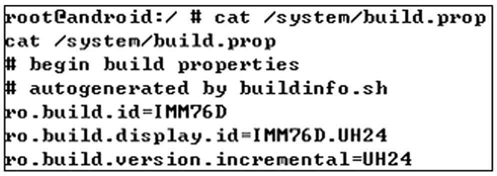
4.6. Restoring a device to its original state

When the targeted device is completely returned to its former state, then power should be removed until the device is used again in order to prevent data modification. If the device is an all-in-one type with the battery, then cut off the power by using the power button, and if the battery can be removed, then remove it.

- In case of turning the device off by using the power button, then an investigator should not use the menu functions of the recovery mode. The menu can be different for each vendor, device, or firmware version, but the ‘reboot system now’ is usually included in the menu of the recovery mode. If this menu is selected and performed, then it mounts the user data partition in order to log the details of performance in the recovery mode.
- Before removing the battery, the USB cable must be separated first. Certain devices (ex: Galaxy S2) mount the user data partition by using the power provided by the USB cable if the battery is separated when the USB cable is still connected.

5. Android Extractor

Based on the procedure aforementioned in Section 4, Fig. 5 illustrates a simple GUI tool, Android Extractor, written by C++. Design of Android Extractor was improved through forensic experts and non-experts’ feedback. The enhanced tool through feedback is easy to use even for the non-specialist, although. Before a user use the tool, the correct driver on the device must be first installed for adb command. This section presents seven steps to extract user data stored in an Android device, maintaining integrity.



```
root@android:/ # cat /system/build.prop
cat /system/build.prop
# begin build properties
# autogenerated by buildinfo.sh
ro.build.id=IMM76D
ro.build.display.id=IMM76D.UH24
ro.build.version.incremental=UH24
```

Fig. 4. Right information in /system/build.prop (Original Firmware Version: UH24; CRMI Version: FD21).

Table 4
Boot block name of Android devices.

Device	Boot block name
Droid (A855)	/dev/block/mtdblock5
Galaxy S2 (SHW-M250S)	/dev/block/mmcblk0p5
Galaxy Nexus (SHW-M420K)	/dev/block/mmcblk0p7
Galaxy Note (SHV-E160S)	/dev/block/mmcblk0p8
Galaxy S3 (SHV-E210S)	/dev/block/mmcblk0p5
Galaxy Note 2 (SHV-E250S)	/dev/block/mmcblk0p8
Vega LTE (IM-A800S)	/dev/block/mmcblk0p8

5.1. Select an Android device

The first step is to select an Android device for investigation. There are seven Android devices which support CRMI as followings: Galaxy S2, Galaxy S3, Galaxy Note, Galaxy Note 2, Galaxy Nexus, Droid, and Vega LTE.

5.2. Turn off the Device & enter the flash mode

After entering the flash mode in order to flash CRMI, USB cable from an Android device needs to be connected to the system running Android Extractor.

5.3. Flash CRMI & wait recovery mode

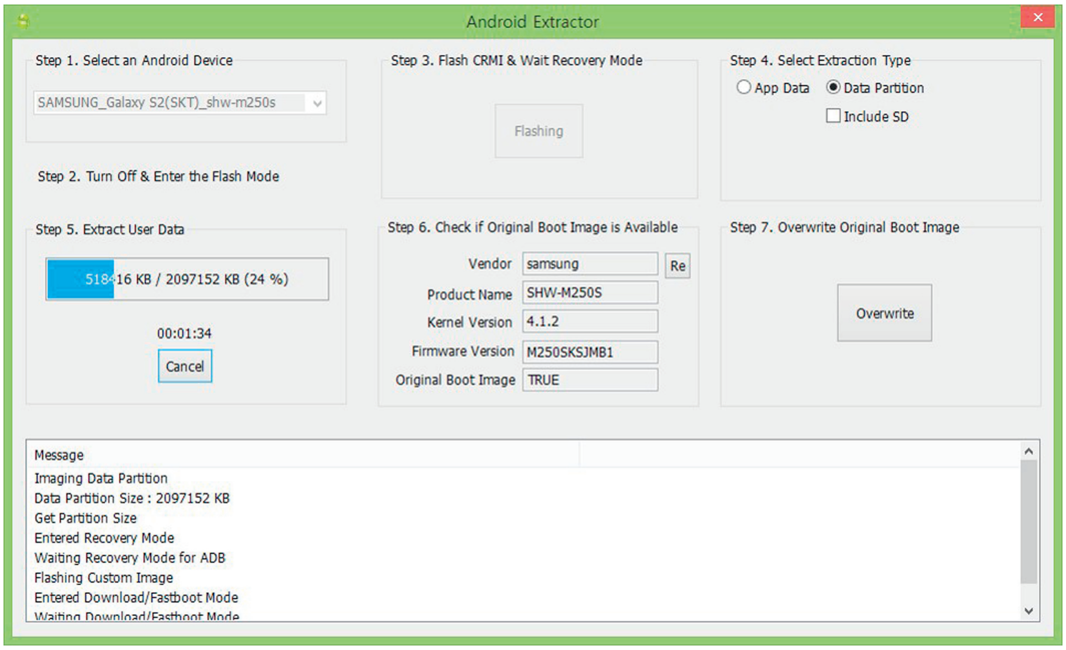
Next, CRMI is flashed into the boot partition using tools which device vendors provide with. Once successfully finished CRMI flashing, the Android device is then rebooted. It leads to enter into the recovery mode with root privilege. Then the dialog window pops up to choose file path to store user data.

5.4. Select extraction type

There are two different extraction types: App Data and Data Partition. App Data extraction type helps to extract user data selectively on the provided list. This list consists of application names and their file paths. Data Partition extraction type helps to dump entire data partition of the target device.

5.5. Extract user data

This step allows the selected user data extraction. In case of App Data type, Android Extract displays the progress bar based on the number of data and elapsed time.


Android Extractor

Step 1. Select an Android Device
SAMSUNG_Galaxy S2(SKT)_shw-m250s

Step 2. Turn Off & Enter the Flash Mode

Step 3. Flash CRMI & Wait Recovery Mode
Flashing

Step 4. Select Extraction Type
App Data [✓ Data Partition]
☐ Include SD

Step 5. Extract User Data
518416 KB / 2097152 KB (24 %)
00:01:34
Cancel

Step 6. Check if Original Boot Image is Available

Vendor	samsung
Product Name	SHW-M250S
Kernel Version	4.1.2
Firmware Version	M250SKSJMB1
Original Boot Image	TRUE

Step 7. Overwrite Original Boot Image
Overwrite

```
Message
Imaging Data Partition
Data Partition Size : 2097152 KB
Get Partition Size
Entered Recovery Mode
Waiting Recovery Mode for ADB
Flashing Custom Image
Entered Download/Fastboot Mode
Waiting Download/Fastboot Mode
```

Fig. 5. Android Extractor.

In case of the other type, Data Partition, it shows the progress bar based on the image size and elapsed time. User data will be stored in a designated path in Section 5.3.

5.6. *Check if original boot image is available*

This step simply informs the original firmware information including vendor, product name, kernel and firmware version, and whether there is the original boot image or not. If the original boot image is not found, it is necessary to obtain the original boot image from elsewhere to restore to the formal state.

5.7. *Overwrite original boot image*

Lastly, this step involves overwriting the original boot image to the boot partition of the device. Once it is done, USB cable and battery are removed from the target device.

6. Experiment using Android Extractor

Using Android Extractor, this study tested whether user data integrity was maintained during the acquisition process. Seven different models were used in test: Galaxy S2, Galaxy S3, Galaxy Note, Galaxy Note 2, Galaxy Nexus, Droid, and Vega LTE. First five devices were included in the top 10 Android devices used globally in November, 2012 (Android device market share, 2012), and extra two ones.

6.1. *Experiment method*

Whether or not data integrity is guaranteed can be judged by simply repeating the process of the entire data acquisition in this study multiple times (Section 4). However, before conducting the described process, data was obtained using JTAG from the devices that support JTAG. This data will be used for comparison after acquisition.

If the integrity of the user data partition is guaranteed, then the integrity of each file is also guaranteed. This study did not conduct data extraction experiments to evaluate the data in the file unit, but evaluated only data acquisition from the partition. The experimentation processes is shown in Fig. 6.

First, we confirm whether or not the targeted device can obtain data by using JTAG. If it is possible, then we obtain data with JTAG and by using the CRMI method. If it is not possible, then obtain data by only using the CRMI method. For accurate experiment results, this study was repeated five times in total and compared the hash values of all the user data partition that were acquired after repeating five times.

In the experiment, the devices from which data could be acquired through JTAG were Galaxy S2 (SHW-M250S) and Galaxy Note (SHV-E160S).

6.2. *Experimentation results*

The experiments using seven Android devices were completed according to the conditions described in this study. This study confirmed that the hash value of the user data partition that was extracted from each device was the same in all observations. Because of this, we feel that the data acquisition method suggested in this study preserves the integrity of the data. In addition, it indicates that acquiring data through JTAG also preserved integrity.

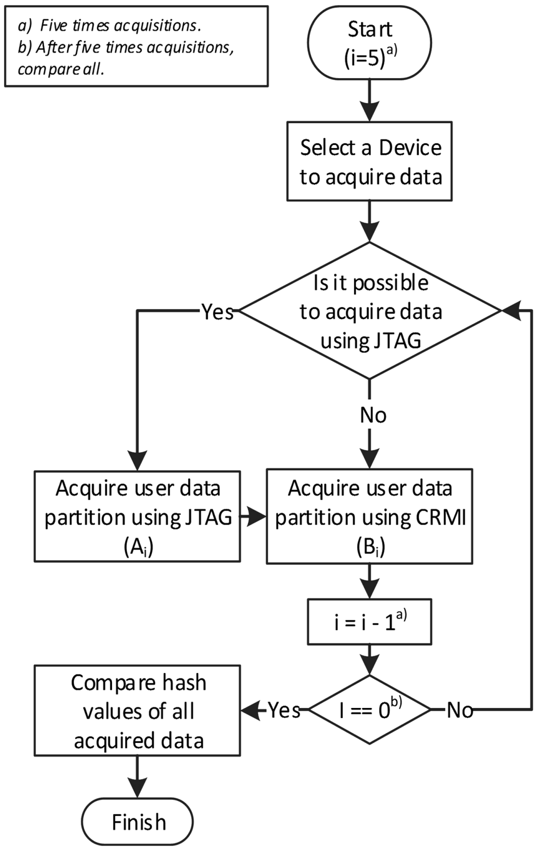

a) Five times acquisitions.
b) After five times acquisitions, compare all.
Start (i=5)a)
Select a Device to acquire data
Is it possible to acquire data using JTAG
Yes
No
Acquire user data partition using JTAG (Ai)
Acquire user data partition using CRMI (Bi)
i = i - 1a)
I == 0b)
Yes
No
Compare hash values of all acquired data
Finish

Fig. 6. Experimentation processes.

7. Conclusions

To the author's knowledge, prior works have not considered the integrity of data acquired from an Android device. However, if the integrity of user data is not guaranteed, data potentially relevant to an investigation, such as data in unallocated or journal areas, can be altered.

This study explained a method of preserving integrity at the time of acquisition of user data by using the previously studied Recovery Mode, from which a tool to automate this process was developed. By using the produced tool, experiments were conducted, targeting various devices to determine if data can be acquired from an Android device while also preserving the integrity of such a device. The results of such tests show that the integrity of the user data area was preserved using Recovery Mode acquisition methods.

Further, the results of the experiment also confirmed that acquiring data acquired using JTAG also preserved the integrity of the user data area.

For accurate experiment results, this study attempted to conduct tests regarding all the file systems that have been used in Android devices, but since we could not get access to all the targeted Android devices, we were only able to test YAFFS2 and Ext4.

Also, in order to return to the former state after flashing the CRMI and acquiring data, an investigator must get the original copy of the boot partition from the previous firmware version. But when the original firmware is hard to obtain, it is difficult to apply this method. As for the device whose original firmware is hard to obtain, this study recommends flashing the CRMI to the recovery partition instead of flashing it to the boot partition.

As mentioned, there are several limitations, but the CRMI method is more efficient compared to other existing methods of forensically sound data acquisition from Android devices.

8. Future research

Although this study attempted to conduct tests targeting a diverse array of Android devices, the tests were mostly conducted targeting the Samsung family of products. Additional tests targeting other products such as Sony and HTC are necessary. Also, since the method of producing the CRMI varies for each vendor, this process should be verified for each vendor.

This study focused on the method of data acquisition. Future work will expand to methods of acquisition and analysis at the same time.

Acknowledgments

This work was supported by the IT R&D program of MOTIE/KEIT. [10035157, Development of Digital Forensic Technologies for Real-Time Analysis]

References

ADB (Android Debug Bridge), http://developer.android.com/tools/help/adb.html.

Android 2.3 Gingerbread uses ext4 file system. http://blog.gsmarena.com/android-2-3-gingerbread-uses-ext4-file-system-promises-better-dual-core-performance/; 2012.

Android 2.3 over reaches 83 percent market share. http://www.zdnet.com/android-4-1-jelly-bean-reaches-1-8-percent-market-share-7000005096/; 2012.

Android App Store. http://dottech.org/73218/android-now-has-over-600000-apps-installed-20-billion-times/; 2012.

Android boot image format, http://www.xinotes.org/notes/note/1048/.

Android device market share, http://blog.w3i.com/2012/12/17/android-device-marketshare-percentages/.

Android Market Share. http://techcrunch.com/2012/11/02/idc-android-market-share-reached-75-worldwide-in-q3-2012/; 2012.

Android Partitions, http://www.enfeuman.com/2011/06/10/android-partitions-explained/.

Breeuwsma MF. Forensic imaging of embedded systems using JTAG (boundary-scan). Digital Investigation 2006;3(1):32–42.

Carrier B. File system forensic analysis. Addison-Wesley; 2005.

Cellebrite UFED. http://www.cellebrite.com/mobile-forensic-products/ufed-touch-ultimate.html; 2012.

Chen S, Yang C. Design and implementation of live SD acquisition tool in Android smart phone. In: Fifth international conference on genetic and evolutionary computing 2011. p. 157–62.

FTL(Flash Translation Layer). Understanding the Flash Translation Layer (FTL) specification. Intel; 1998.

Han S-W. Flash memory wear leveling system and method Issued 2000. US patent 6,016,275.

Hoong A. Android forensics: investigation, analysis and mobile security for google Android. Syngress; 2011.

Journal File Systems. http://www.linux-mag.com/id/1180/; 2002.

Jovanovic Z. Android forensics techniques. International Academy of Design and Technology; 2012.

Kim K, Hong D, Ryu J. Forensic data acquisition from cell phones using JTAG interface. Information Security Research Division 2008:410–4.

Kim D, Park J, Lee K, Lee S. Forensic analysis of Android phone using Ext4 file system journal log. Future information technology. Application and Service 2012;164:435–46.
Lee S, Kim H, Lee S, Lim J. Digital evidence collection process in integrity and memory information gathering. Systematic Approaches to Digital Forensic Engineering 2005:236–47.
Micro Systemation XRY. http://www.msab.com/; 2012.
MTD (Memory technology devices), http://www.linux-mtd.infradead.org/faq/nand.html.
rootfs, https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt.
Vidas T, Zhang C, Christin N. Toward a general collection methodology for Android devices. Digital Investigation 2011;8:S14–24.
YAFFS2 (Yet Another Flash File System), http://www.yaffs.net/yaffs-2-specification.

Namheun Son received his Master's degree in Information Security, Korea University. He is now studying doctor course in Graduate School of Information Security, Korea University. He is currently working for Digital Forensic Research Center in Korea University. He has performed projects related to Windows Memory Forensics, Relational Database Management System (RDBMS) Forensics, and Smartphone Forensics. His research interests are Smartphone Forensics, Windows Memory Forensics and User Behavior Analysis Forensics.

Yunho Lee received his B.S. degree in Computer Science from Pukyong National University. He is now studying master course in Graduate School of Information Security, Korea University. He is currently working for Digital Forensic Research Center in Korea University. He has performed projects related to Cloud Forensics and Android Forensics. His research interests are digital forensics, cloud forensics and smartphone forensics.

Dohyun Kim received his B.S. degree in Computer Science from Seoul National University of Science and Technology, He is now studying master course in Graduate School of Information Security, Korea University. He is currently working for Digital Forensic Research Center in Korea University. He has performed projects related to Android Forensics, Analysis of Ext4 File System and Reference Data Set (RDS). His research interests are Smartphone Forensics, File System and Digital Forensics.

Joshua I. James is a researcher with the University College Dublin Digital Forensic Investigation Research Group and an adjunct researcher with the Korean National Police University International Cybercrime Research Center. He works closely with Law Enforcement, and lectures on Live Data Forensics and Digital Forensic Practice. Coming from a background in Network Security and Administration, his focus is now on automatic digital evidence identification and correlation. He is specifically working in the area of rigorous automated investigation and analysis techniques for digital investigations with an emphasis on ease of use. To this end, he contributes to a number of open source projects for automatic evidence acquisition and analysis, such as project ATOM [Cybercrime-Tech.com], Goldfish memory analysis, and the FIREBrick write blocker [digitalfire.ucd.ie].

Sangjin Lee received his Ph.D. degree from Korea University. He is now a Professor in Graduate School of Information Security at Korea University and the head of Digital Forensic Research Center in Korea University since 2008. He has published many research papers in international journals and conferences. He has been serving as chairs, program committee members, or organizing committee chair for many domestic conferences and workshops. His research interests include digital forensic, steganography, cryptography and cryptanalysis.

Kyung-ho Lee received his Ph.D. degree from Korea University. He is now a Professor in Graduate School of Information Security at Korea University, and leading the Risk management Laboratory in Korea University since 2012. He has a high level of theoretical principles as well as on-site experience. He was a former CISO in NHN corporation, and now he takes as the CEO of SecuBase corporation. His research interests include information security management system (ISMS), risk management, information security consulting, privacy policy, and privacy impact assessment (PIA).