Contents lists available at ScienceDirect

Digital Investigation

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

Live acquisition of main memory data from Android smartphones and smartwatches

Seung Jei Yang [a, *], Jung Ho Choi [a], Ki Bom Kim [a], Rohit Bhatia [b], Brendan Saltaformaggio [c], Dongyan Xu [b]

[a] *The Affiliated Institute of ETRI, 1559, Yuseong-daero, Yuseong-gu, Daejeon 34044, Republic of Korea*
[b] *Department of Computer Science and CERIAS, Purdue University, West Lafayette, IN 47907, USA*
[c] *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*

A R T I C L E I N F O

Article history:
Received 7 April 2017
Received in revised form
28 August 2017
Accepted 13 September 2017
Available online 19 September 2017

Keywords:
Android forensics
Main memory
Physical acquisition
Firmware update mode
Smartphones
Smartwatches

A B S T R A C T

Recent research in Android device forensics has largely focused on evidence recovery from NAND flash memory. However, pervasive deployment of NAND flash encryption technologies and the increase in malware infections which reside only in main memory have motivated an urgent need for the forensic study of main memory. Existing Android main memory forensics techniques are hardly being adopted in practical forensic investigations because they often require solving several usability constraints, such as requiring root privilege escalation, custom kernel replacement, or screen lock bypass. Moreover, there are still no commercially available tools for acquiring the main memory data of smart devices. To address these problems, we have developed an automated tool, called AMD, which is capable of acquiring the entire content of main memory from a range of Android smartphones and smartwatches. In developing AMD, we analyzed the firmware update protocols of these devices by reverse engineering the Android bootloader. Based on this study, we have devised a method that allows access to main memory data through the firmware update protocols. Our experimental results show that AMD overcomes the usability constraints of previous main memory acquisition approaches and that the acquired main memory data of a smartphone or smartwatch can be accurately used in forensic investigations.

Introduction

As of the second quarter of 2016, the Android operating system (OS) is the most widely used mobile OS, accounting for 87.6% share of the total smartphone OS market (Smartphone OS Market Share, 2016). In addition, development of various smartwatches based on the Android OS is expected to grow rapidly in the future (World Smartwatch Market, 2016). Although forensic research on smart devices has been actively conducted, most digital forensic commercial tools (Cellebrite UFED, 2017; Mobile Phone Examiner Plus, 2017; MSAB XRY, 2017; Oxygen Forensics, 2017) focus mainly on the acquisition and analysis of NAND flash data from smart devices. There is currently no support for forensic investigation of the main memory of smart devices. At the same time, the main memory of smart devices has increased in speed and capacity, and the prevalence of malware that resides only in main memory has also increased (Mobile Malware, 2017). Moreover, device manufacturers tend to use complex security functions such as Full-Disk Encryption (FDE) (Full-Disk Encryption, 2017), File-Based Encryption (FBE) (File-Based Encryption, 2017), Social Networking Service (SNS) conversation encryption (Facebook Messenger Secret Conversations, 2016; WeChat Message Cryptography, 2017; WhatsApp Security, 2017), KNOX (Samsung KNOX, 2017), and Secure Boot (Secure Boot, 2017). For these reasons, it will be difficult to rely only on NAND flash-based forensic technologies.

To overcome the above limitations, new advances in field-deployable main memory forensics are necessary, especially the technology to acquire main memory data from smart devices for use in forensic investigations. Currently, the technologies used for main memory data acquisition have several usability

* Corresponding author.
E-mail addresses: sjyang@nsr.re.kr, yangs7256@gmail.com (S.J. Yang).

http://dx.doi.org/10.1016/j.diin.2017.09.003
1742-2876/© 2017 Elsevier Ltd. All rights reserved.

limitations that make them difficult to apply in forensic investigations, and current research technologies on the subject are hardly applied in practice because they do not solve the following limitations.

First, a system restart may be required when acquiring main memory data (Sylve et al., 2012; Taubmann et al., 2015). During the process of replacing the kernel with a custom kernel or a custom recovery image the system is restarted which causes a significant loss of evidence.

Second, to overcome the limitation of kernel replacement, some methods for acquiring main memory data work by injecting executable code into a smart device (Yang et al., 2016). However, to use this method, it is necessary to acquire root privilege in advance (Rooting, 2016). In recent years, with the introduction of security technologies such as Secure Boot and KNOX, root privilege escalation has become just as difficult as acquiring main memory data. Moreover, during root privilege escalation, the system may also need to be restarted.

Third, an Android Debug Bridge-based protocol may be used to acquire data by connecting to an Android smart device with a data cable (Android Debug Bridge, 2017). However, if a screen lock (such as a pattern lock, user passwords, etc.) is applied on a smart device (and therefore USB debugging mode is disabled) access to main memory and data acquisition are both impossible.

In this study, we have solved these limitations by providing support for the physical acquisition of main memory data from smart devices in forensic investigations. To this end, we have investigated main memory acquisition based on the firmware update protocols shared by smartphones and smartwatches. The firmware update protocol is used to update to a new Android OS or to patch software problems, and each manufacturer provides a firmware update program to take advantage of the firmware update protocol used in the Android bootloader (LG Software and tools Download, 2017; Pantech Self-Upgrade, 2017; Samsung Kies, 2017; Samsung Odin, 2017).

We first analyzed the firmware update command by reverse engineering the Android bootloader firmware update protocol. It is important to note that the firmware update process runs only when a smartphone enters a special mode called firmware update mode, and it cannot run in normal operation mode. From analyzing these firmware update programs, we have confirmed that it is possible to **switch from normal operation mode to firmware update mode without a system restart** through a software command. Moreover, we have developed a way to access the main memory and obtain its data during this firmware update process. In the case of Samsung smartphones and smartwatches, we have found a vulnerability that can access and acquire data in the main memory. In the case of LG models, we have found that a main memory read command exists in the manufacturer's undocumented protocol, despite it not being used in the firmware update process.

Based on these methods, we have developed AMD, a forensic tool that provides automated acquisition of main memory data from smartphones and smartwatches through their firmware update protocols. The operation of AMD is fully automated, and AMD presents a simple GUI interface for investigators to use (see Section Android main memory dump tool). The following are the key characteristics of the AMD acquisition method:

-It is possible to switch the device to firmware update mode for the acquisition of main memory data without a system restart through a software command.
-It is possible to acquire the main memory data through a firmware update command in firmware update mode without kernel replacement, without a device restart, and without root privilege.
-The proposed method does not use the Android ADB protocol because it acquires the data through the main memory read command. Therefore, it is affected by neither the USB debugging mode nor by a screen lock or user password.
-Our experiments show that smart device memory images acquired by AMD can be accurately used in forensic investigations.

Related work

There are currently no commercially available tools for acquiring the main memory data of smart devices in forensic studies. Because the Android system structure conforms to that of the Linux operating system, most studies on main memory data acquisition for Android have been performed by extending the Linux-based data acquisition methods. The traditional Linux methods for main memory acquisition can be performed by software directly through the memory device file/*dev/mem* or by using a Loadable Kernel Module (LKM) such as Fmem (How to acquire memory from a running Linux system, 2017). However, these acquisition methods cannot be applied to Android smart devices because Android has recently blocked direct access to/*dev/mem* and does not support ARM-based LKMs that target Android.

Sylve et al. (2012) proposed the Linux Memory Extractor (LiME) method (LiME, 2017). This method is the most widely used for acquiring the main memory of Android smartphones, and it is an LKM-based acquisition method like Fmem. However, to use the LiME method, it is necessary to gain root privilege (persistent or temporary root) to install the module. In some cases, it may also be necessary to create a custom kernel to use LiME: In some Android models, the default kernel does not allow inserting kernel modules (the *insmod/modprobe* operation does not exist in the default kernel). For such cases, a custom kernel is required. There are techniques to compile a kernel without the original kernel source. However, recently released Android models need the original kernel source to compile a custom kernel because the kernel configuration and toolchains must be exactly the same as the stock firmware running on the device. Any change prevents the module from being loaded due to a CRC checksum mismatch. Therefore, the original kernel source code of the device is required for such cases. Since a system restart may be required to obtain root privileges and to use a custom kernel, there is a limitation in the applicability of the LiME method to smart devices in forensic investigations.

To overcome the limitations of custom kernel replacement, recent research has proposed the AMExtractor tool for obtaining main memory data, which converts the data acquisition function into injectable code (AMExtractor, 2016; Yang et al., 2016). This method has the advantage of not requiring a kernel module or the original kernel source since it does not need to replace the current kernel with a custom kernel. However, to acquire main memory data, this method requires a root privilege escalation in advance. Unfortunately, because it is difficult to acquire root privilege (due to the strengthening of security functions and the fact that the system may need to be restarted during privilege escalation), this method is hardly being adopted in forensic investigations.

Similarly, the Bare Metal Application (BMA) method for obtaining main memory data (Taubmann et al., 2015) leverages recovery mode by uploading the acquisition code to the kernel area of the Android recovery image. This method has the advantage of requiring neither a kernel mod nor a root privilege escalation. However, because it is necessary to create a custom recovery image, this method requires a recovery image file for the OS version of the device and a process for flashing the created custom recovery image. Since the system is restarted in this process, this method can result in significant loss of evidence. Recently, security technologies such as KNOX and Secure boot have been strengthened, making it difficult to acquire data based on a custom recovery image.

The TrustDump method, a TrustZone-based memory acquisition mechanism, has been also been proposed (Sun et al., 2014). This method can acquire the main memory data and CPU register values of a mobile device even when the operating system has crashed or has been compromised. However, this method is only supported on Freescale i.MX53 QSB, an embedded development board — making it an impractical tool for acquisition on commodity smart devices during forensic investigations.

Yang et al. (2015) proposed a NAND flash acquisition method based on analyzing the firmware update protocols of Android smartphones. They developed a physical acquisition method for Android smartphones by leveraging the NAND flash read command in the bootloader. However, this approach is hardly applicable to main memory acquisition. Since the NAND flash read commands are used during the firmware update process, they can be easily isolated. Thus, main memory data acquisition is impossible with this method because only a part of the main memory is read and the main memory read command is not used during the firmware update process. In this work we take an entirely new approach: we first performed reverse engineering to identify an unknown protocol that can access and dump main memory data during the firmware update process. This protocol is leveraged in our development of the AMD tool.

There are relevant studies on the acquisition of memory by analyzing Samsung SBOOT (Exploiting Android S-Boot, 2017; Sboot_dump, 2017). These approaches have been shown to be both robust and reliable, but they remain limited to only device models using the Samsung Exynos chipset — a small subset of the models that AMD supports.

Additionally, there are various studies to acquire NAND flash data from Android smartphones. Logical acquisition methods acquire user data stored on a smartphone via ADB Backup (Android Backup Extractor, 2014) or Content Provider (Hoog, 2011). Several rooting-based acquisition studies were introduced in scholarly works (Hoog, 2009; Lessard and Kessler, 2010). An acquisition method based on changing the custom recovery image has also been studied (Vidas et al., 2011; Son et al., 2013). The analysis of social networking applications (Mutawa et al., 2012), a prototype enterprise monitoring system for Android smartphones (Grover, 2013), and Kindle forensics (Hannay, 2011; Iqbal et al., 2013) were also studied. However, these methods only recover NAND flash memory and are not applicable to main memory acquisition, which involves different challenges such as requiring root privilege escalation, custom kernel replacement, or screen lock bypass.

In addition to physical dumping methods for acquiring the entire main memory data in Android, it is also possible to connect a smart device to a PC and then use ADB and Google Dalvik Debug Monitor Service (DDMS) tools to acquire and analyze the heap memory of Android apps (DDMS, 2017; HPROF, 2017). However, these methods are performed using the ADB protocol and, therefore, USB debugging mode must be enabled. Further, because this mode is disabled by default, it is impossible to apply these methods when a pattern lock or a user password are set. Moreover, the information that can be obtained with this method is very limited compared to that obtained with physical acquisition (i.e., the entire content of physical memory) methods.

There are hardware-based acquisition methods for acquiring main memory data using JTAG (Willassen, 2005; Guri et al., 2015). The acquisition is done using the JTAG debug interface that exists on the PCB of a smart device. Representative examples of hardware-based tools include Trace32 (Trace32, 2017), Riff Box (RIFF box, 2017), and the ORT tool (ORT tool, 2017). These tools are mainly used by manufacturers to develop smart devices or to repair smart devices with hardware/firmware problems. However, not all smart devices support the JTAG interface, and there is a risk of damage when a smart device is disassembled; therefore, the applicability of these techniques in forensic investigations is quite limited.

Once collected, the Volatility tool is often used to analyze the physical image of Android's main memory (Android volatility, 2016). It has supported the analysis of Android main memory since version 2.3. This tool is compatible with the image format dumped by the LiME method, which acquires separate system RAM areas of an Android device's main memory; therefore, most acquisition tools create an image in the LiME format and use Volatility to validate the integrity of the acquired image. In our experiments, we use Volatility and a number of other metrics to verify the correctness of AMD's memory images.

Android main memory acquisition in firmware update mode

Android's main memory is a module that corresponds to the main memory of a computer. When a user executes a program, the program fetches the related data from the NAND flash memory, uploads this data to main memory, executes its operations on that data, and then stores any persistent data back to the NAND flash memory. Therefore, the main memory contains evidence of program executions and actions that the user has performed. One key feature of evidence in main memory is that any encrypted data stored to NAND flash memory will remain decrypted in the main memory. Decrypted data such as the FDE key and user password can be obtained from main memory. Moreover, manipulated information in the kernel area, such as a hidden process, can be obtained from main memory data. However, since main memory is volatile, it only keeps data when the system is powered on, it loses the data if power is lost or if the system is restarted. Therefore, the physical acquisition of the main memory data should be performed as fast as possible in the live state, without a system restart of the smart devices in forensic studies.

Main memory acquisition from Samsung devices

In normal boot mode, there is no way to access the main memory. We analyzed the firmware update program, firmware update protocol, and firmware update process of Samsung to look for a method to access the main memory. The firmware update protocol is the only protocol that can access the memory directly by software, and we analyzed the commands that are used in the firmware update process and also analyzed how to access the main memory.

The firmware update mode and protocol are called Odin mode and Odin protocol, respectively. To acquire the main memory of smart devices in forensic studies, the device should be switched to Odin mode without a system restart. For this, we analyzed the

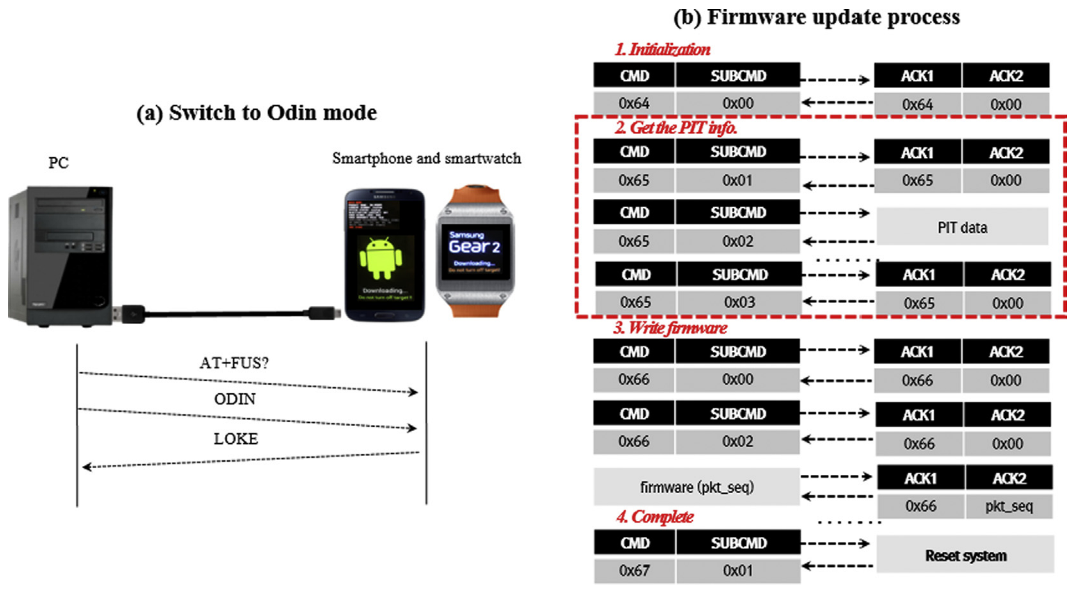


Fig. 1. Firmware update process after the device switches to Odin mode (Samsung).

Odin protocol and found that software commands exist that can switch the device to Odin mode without a system restart. If the AT command "AT + FUS?" and the string "ODIN" are sent in sequence from a PC to the Samsung smart device while connecting the device to the PC with a data cable, the smart device will return the string "LOKE" to the PC in the ACK message. If this process is successful, the Samsung smart device will switch to Odin mode. After this mode is entered, the firmware update process is performed.

Fig. 1 shows the firmware update process after the device switches to Odin mode. The 0x64 command is sent to the smart device to initialize the ODIN protocol communication process. After the initialization, it proceeds to partition the configuration for firmware update by using the 0x65 command/subcommands. Samsung manages every partition via the Partition Information Table (PIT) structure. PIT is an integral element of all firmware programs that contain the map of storage allocations for different system partitions. The firmware update process is executed according to the partition information through the 0x66 command/subcommands. When the firmware update process is completed, the Odin protocol is terminated via the 0x67 command/subcommands and the system is restarted to complete the firmware update process. During this process, we found a method for acquiring the main memory data in the process of reading the PIT partition information from a memory buffer (0x65).

PIT read command: 0x65

The PIT information is uploaded to the *pit_buffer* of the main memory according to the 0x65 command/subcommands, and it is read or written, and then finally written to the NAND flash memory. Table 1 shows the subcommands of 0x65.

This command reads the partition information by dividing it into blocks of 500 bytes starting from the *pit_buffer* address in the main memory. For the Galaxy S2 model, the PIT size (*pit_size*) is 4096 bytes, and for models after Galaxy S2, the PIT size (*pit_size*) is 8192 bytes. In both cases, the size of the block that can be read from the main memory at once is 500 bytes. Therefore, in the case of the Galaxy S2 device, the area from 0 to 8 blocks after the *pit_buffer* address contains the PIT information. Similarly, for models after the Galaxy S2, the area from 0 to 16 blocks after the *pit_buffer* address contains the PIT information. Fig. 2 shows the process of reading the partition information from a Galaxy S2 model (CMD: 0x65; SUBCMD: 0x02).

PIT read algorithm

The IDA Pro tool (Hex-Rays, 2016) was used to decompile the PIT reading logic within the Samsung bootloader. Listing 1 shows the 0x65 command logic obtained from decompiling the Samsung bootloader. The reverse-engineered code of the PIT read command (CMD: 0x65; SUBCMD: 0x02) in Listing 1 is also represented in pseudocode steps in Table 2.

When the Odin program requests to read *y* blocks from the *pit_buffer* through the PIT read command, the *send_address*, that is,

Table 1
Subcommands of 0x65.

Command	Subcommands	Parameter	Description
0x65	0x00	–	Start writing the partition information
	0x01	–	Start reading the partition information
	0x02	Block number *y*	Read or Write the PIT from the main memory
	0x03	–	Finish the read operation/write to flash memory

```
unsigned __int8 *__fastcall process_packet_101_set_partition(unsigned __int8 *result)
{
  int v_len; // ST0C_4@7
  int v2; // r3@9
  int v3; // r0@15
  int v4; // r1@15
  int v5; // r2@15
  int v_cmd; // [sp+8h] [bp-14h]@1
  int v7; // [sp+10h] [bp-Ch]@9
  signed int v8; // [sp+14h] [bp-8h]@3

  v_cmd = (result[3] << 24) | (result[2] << 16) | (result[1] << 8) | *result;
  switch ( (result[7] << 24) | (result[6] << 16) | (result[5] << 8) | result[4] )
  {
    case 0:
      pit_loaded = 0;
      result = (unsigned __int8 *)upload_ack(v_cmd, 0);
      break;
    case 1:
      pit_loaded = 1;
      v8 = v_cmd;
      read_part_info_(dit.pit_buffer);
      if ( check_pit_integrity(dit.pit_buffer) != 1 )
        v8 = -1;
      result = (unsigned __int8 *)upload_ack(v8, 4096);
      break;
    case 2:
      if ( pit_loaded )             // PIT read command (CMD:0x65, SUBCMD: 0x02)
      {
        if ( pit_loaded == 1 )
        {
          v7 = (result[11] << 24) | (result[10] << 16) | (result[9] << 8) | result[8];
          v2 = -500 * v7 + 4096;
          if ( v2 >= 500 )
            v2 = 500;
          result = (unsigned __int8 *)upload(dit.pit_buffer + 500 * v7, v2);
        }
      }
      else
      {
        v_len = (result[11] << 24) | (result[10] << 16) | (result[9] << 8) | result[8];
        upload_ack(v_cmd, 0);
        download(&byte_4EE00000[18874368], v_len);
        FillChar(dit.pit_buffer, 0, 4096);
        t_copy_data(dit.pit_buffer, (int)&byte_4EE00000[18874368], v_len);
        result = (unsigned __int8 *)upload_ack(v_cmd, 0);
      }
      break;
```

Listing 1. Code for reverse engineering of the 0x65 command in the bootloader.

the start address of each block, is calculated with Equation (1) using the address of *pit_buffer*. The variable *y* indicates the blocks number in the PIT section, and its value is also used to compute the *send_address* of the block. Equation (2) uses *pit_size* (4096 bytes for Galaxy S2 and 8192 bytes for models after Galaxy S2) to calculate how much memory should be read from *send_address*. The value is stored in *send_size*. Note that for the last block read this size will be less than 500 bytes. For Galaxy S2, the eighth block is 96 bytes, and for models after Galaxy S2, the 16th block is 192 bytes. If *send_size* from Equation (2) is greater than 500 bytes, then *send_size* is set to 500 according to Equation (3). Then, through Equation (4), *send_size* bytes are read from main memory at *send_address*. During the correct usage of this process, the partition information is read from the main memory.

Main memory acquisition

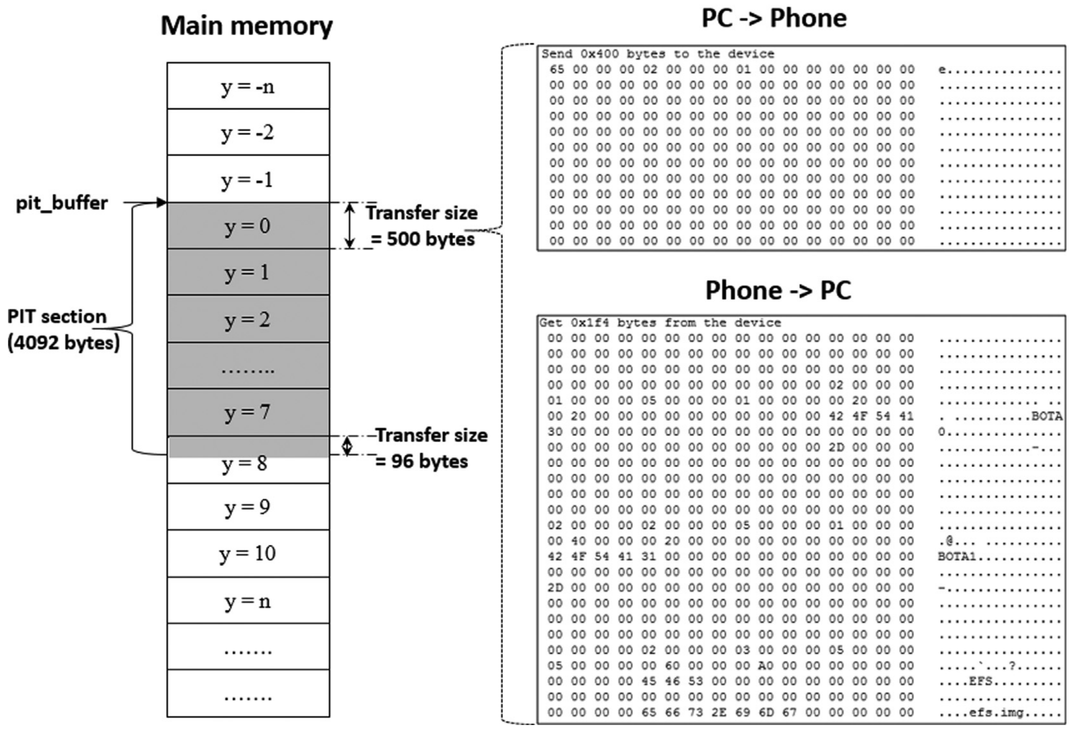
As a result of analyzing the PIT read code in Section PIT read algorithm, it was confirmed that a desired block of main memory can be read according to the value of *send_address* that is computed. Even if a block value outside of the *pit_buffer* area is specified as a parameter in the PIT read command, the main memory data of the corresponding area can be acquired. Fig. 3 shows the values of *send_address* and *send_size* according to the value of *y* (the block number) for the entire main memory of the Galaxy S2 model. The addresses are calculated via the PIT read algorithm in Table 2.

In this way, the main memory can be divided into A, B, and C areas. Area A indicates the area above the PIT area in main memory, area B is the PIT area, and area C represents the addresses below the PIT area in the main memory. To obtain the main memory data above the PIT area (area A), it is necessary to know the block number (y) of the physical main memory starting address. For this, we can compute the starting block number using the start address of the *pit_buffer* (the top of area B) and the physical start address of the main memory. For example, in the case of a Samsung device, the physical start address of the main memory is 0x40000000.

However, the start address of the *pit_buffer* is set when the Odin protocol is initialized, it differs from device to device, and changes at every boot — so it cannot be known *a priori*. By further analyzing the protocol, we have found how to calculate the start address of the *pit_buffer* via the following two steps:

Samsung PIT read command format (1024 bytes)

CMD (4bytes)	SUBCMD (4bytes)	Block number (4bytes)	0x00 (1012bytes)
65 00 00 00	02 00 00 00	01 00 00 00	00 00 00 … 00 00



```
Send 0x400 bytes to the device
 65 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00   e...............
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
Get 0x1f4 bytes from the device
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00   ................
 01 00 00 00 05 00 00 00 01 00 00 00 00 20 00 00   ............. ..
 00 20 00 00 00 00 00 00 00 00 00 00 42 4F 54 41   . ..........BOTA
 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0...............
 00 00 00 00 00 00 00 00 00 00 00 00 2D 00 00 00   ............-...
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 02 00 00 00 02 00 00 00 05 00 00 00 01 00 00 00   ................
 00 40 00 00 00 20 00 00 00 00 00 00 00 00 00 00   .@... ..........
 42 4F 54 41 31 00 00 00 00 00 00 00 00 00 00 00   BOTA1...........
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   -...............
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 02 00 00 00 03 00 00 00 05 00 00 00   ................
 05 00 00 00 00 60 00 00 00 A0 00 00 00 00 00 00   .....`...?......
 00 00 00 00 45 46 53 00 00 00 00 00 00 00 00 00   ....EFS.........
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 65 66 73 2E 69 6D 67 00 00 00 00 00   ....efs.img.....
```

Fig. 2. The process of reading the PIT (Galaxy S2).

Step 1 First, we must acquire the *pit_buffer* information. The start address of the *pit_buffer* is calculated using the last 16 bytes of the −1 block, which is the block immediately preceding the *pit_buffer*. Luckily, we can acquire the contents of −1 block through the PIT read command.

Step 2 Using the −1 block, we can calculate the *pit_buffer* start address. Fig. 4 shows the process of calculating the start address of the pit_buffer for the Galaxy S3 device. The four 4-byte values stored in the 16 bytes of the −1 block correspond to the values of PIT_Size, PIT_Alloc, PIT_Prev, and PIT_Next, respectively. Among them, the address of pit_buffer can be obtained using PIT_Size and PIT_Next. The value of PIT_Next means the next address at the end of the PIT, and PIT_Size means the size of the PIT. Therefore, by subtracting PIT_Size from PIT_Next, the algorithm can calculate the start address of the pit_buffer.

$$\textit{pit\_buffer}\ \text{start address} = \text{PIT\_Next} - \text{PIT\_Size}$$

Table 2
PIT read algorithm.

Input: *pit_buffer* address, block number *y*

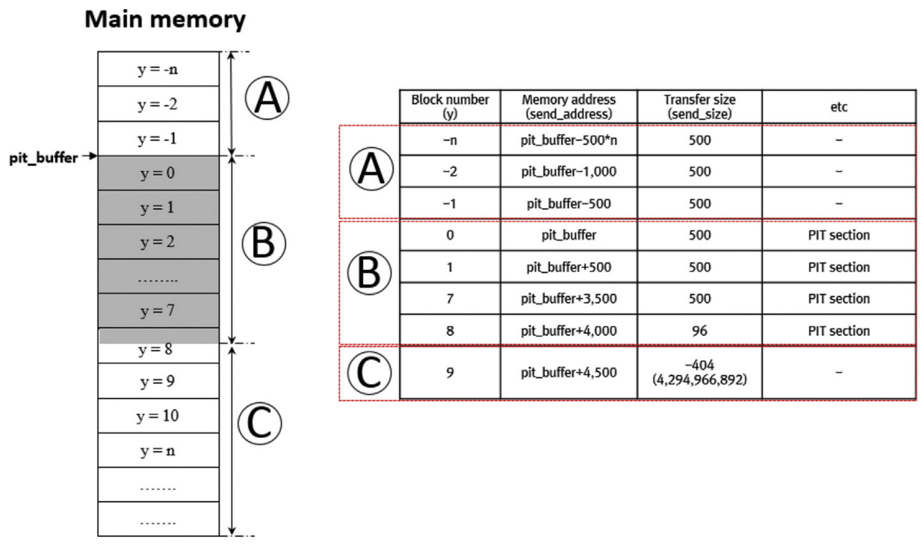
Step	Equation pseudocode
(1)	*send_address* = *pit_buffer* + 500**y*;
(2)	*send_size* = *pit_size* − 500**y*;
(3)	if *send_size* > 500 then *send_size* = 500;
(4)	upload (*send_address*, *send_size*);

The structure of the −1 block is different across the Samsung models. The *pit_buffer* start addresses of the Galaxy S4 and Galaxy Note 3 models are set to the −1 block, unlike the Galaxy S3 model. Fig. 5 shows the *pit_buffer* start address of the Galaxy S4 that we tested.

When the *y* value of the start block in the main memory is obtained, the main memory data above the PIT area can be dumped. The values of *send_address* and *send_size* are calculated according to the algorithm in Table 2 even if the *y* block value of area A has a negative value. As shown in Fig. 3, even if a negative *y* block is requested by the PIT read command, the data acquisition is performed via 500 byte blocks from the calculated *send_address*.

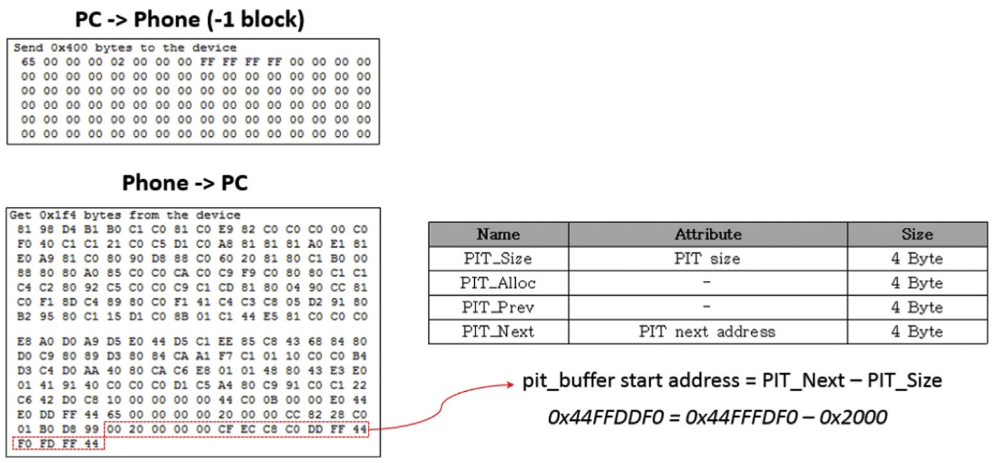
The main memory data of the PIT area (area B) can be acquired from the zeroth block directly via the PIT read command.

To acquire the main memory data below the PIT area (area C), both *send_address* and *send_size* are calculated according to the algorithm shown in Table 2, and therefore *send_size* becomes a negative value. As shown in Fig. 3, *send_size* has a negative value of −404. There is no part of the PIT read algorithm that handles the case where *send_size* is negative. Therefore, a value of −404 in *send_size* is recognized as an unsigned value, and as a result, a value of −404 is calculated as 4,294,966,892 and a command to dump that size is executed. In this case, the algorithm will read from the specified block to the end of the main memory if a command to read block 9 is sent. After reading to the end of the main memory, the command performs no more read operations. Therefore, the acquisition of the main memory data below the PIT area is performed by a single read operation.



	Block number (y)	Memory address (send_address)	Transfer size (send_size)	etc
A	−n	pit_buffer−500*n	500	–
	−2	pit_buffer−1,000	500	–
	−1	pit_buffer−500	500	–
B	0	pit_buffer	500	PIT section
	1	pit_buffer+500	500	PIT section
	7	pit_buffer+3,500	500	PIT section
	8	pit_buffer+4,000	96	PIT section
C	9	pit_buffer+4,500	−404 (4,294,966,892)	–

Fig. 3. Values of *send_address* and *send_size* according to *y* block values (Galaxy S2).



```
Send 0x400 bytes to the device
 65 00 00 00 02 00 00 00 FF FF FF FF 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

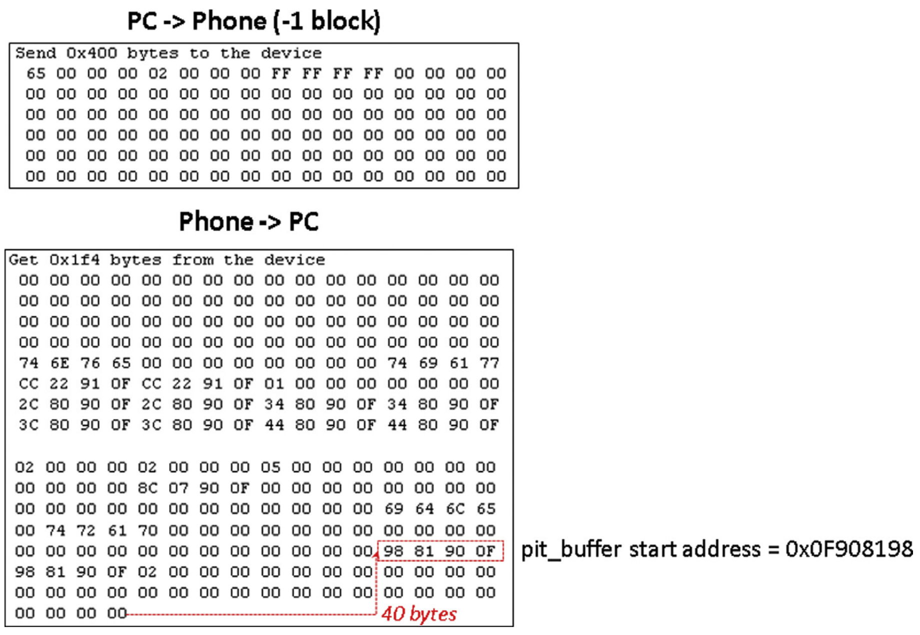
```
Get 0x1f4 bytes from the device
 81 98 D4 B1 B0 C1 C0 81 C0 E9 82 C0 C0 C0 00 C0
 F0 40 C1 C1 21 C0 C5 D1 C0 A8 81 81 81 A0 E1 81
 E0 A9 81 C0 80 90 D8 88 C0 60 20 81 80 C1 B0 00
 88 80 80 A0 85 C0 C0 CA C0 C9 F9 C0 80 80 C1 C1
 C4 C2 80 92 C5 C0 C0 C9 C1 CD 81 80 04 90 CC 81
 C0 F1 8D C4 89 80 C0 F1 41 C4 C3 C8 05 D2 91 80
 B2 95 80 C1 15 D1 C0 8B 01 C1 44 E5 81 C0 C0 C0

 E8 A0 D0 A9 D5 E0 44 D5 C1 EE 85 C8 43 68 84 80
 D0 C9 80 89 D3 80 84 CA A1 F7 C1 01 10 C0 C0 B4
 D3 C4 D0 AA 40 80 CA C6 E8 01 01 48 80 43 E3 E0
 01 41 91 40 C0 C0 C0 D1 C5 A4 80 C9 91 C0 C1 22
 C6 42 D0 C8 10 00 00 00 00 44 C0 0B 00 00 E0 44
 E0 DD FF 44 65 00 00 00 00 20 00 00 CC 82 28 C0
 01 B0 D8 99 00 20 00 00 00 CF EC C8 C0 DD FF 44
 F0 FD FF 44
```

Name	Attribute	Size
PIT_Size	PIT size	4 Byte
PIT_Alloc	–	4 Byte
PIT_Prev	–	4 Byte
PIT_Next	PIT next address	4 Byte

pit_buffer start address = PIT_Next – PIT_Size

0x44FFDDF0 = 0x44FFFDF0 – 0x2000

Fig. 4. The process of calculating the *pit_buffer* start address (Galaxy S3).



```
Send 0x400 bytes to the device
 65 00 00 00 02 00 00 00 FF FF FF FF 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Get 0x1f4 bytes from the device
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 74 6E 76 65 00 00 00 00 00 00 00 00 74 69 61 77
 CC 22 91 0F CC 22 91 0F 01 00 00 00 00 00 00 00
 2C 80 90 0F 2C 80 90 0F 34 80 90 0F 34 80 90 0F
 3C 80 90 0F 3C 80 90 0F 44 80 90 0F 44 80 90 0F

 02 00 00 00 02 00 00 00 05 00 00 00 00 00 00 00
 00 00 00 00 8C 07 90 0F 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 69 64 6C 65
 00 74 72 61 70 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 98 81 90 0F
 98 81 90 0F 02 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00
```

pit_buffer start address = 0x0F908198

Fig. 5. The *pit_buffer* start address (Galaxy S4).

As a result, in the case of a Samsung smart device, access to both the entire main memory area and data acquisition is possible using the PIT read command. Fig. 6 shows an example of the main memory data acquisition from the Galaxy S3 model.

Main memory acquisition in LG devices

The LG firmware update program and firmware update protocol were reverse engineered and then analyzed for main memory access. In the case of LG, the firmware update mode and protocol are called download mode and download protocol, respectively. The download protocol was analyzed by decompiling the LG bootloader (SBL3).

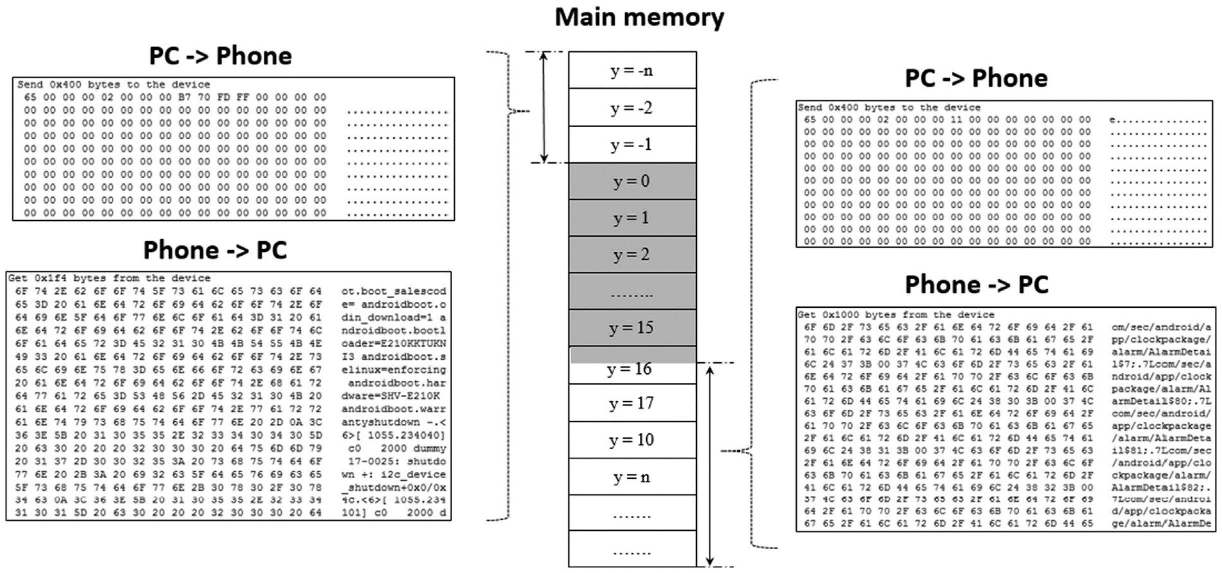
The commands used in download mode are LG's own implementation and follow the frame structure of the High-Level Data Link Control (HDLC) protocol (High-Level Data Link Control, 2017), which starts with the HDLC flag (0x7E); followed by a 1-byte command, variable-sized data, and 2 bytes of CRC-16; and ends with the HDLC flag (0x7E). If download commands are sent to a smart device, the ACK (0x02) or NAK (0x03) reply packet is sent.

In the case of the LG model, the device should be switched to download mode without a system restart. In the analysis of the download protocol, we found software commands that could switch the device to download mode without a system restart. If the “0x3A” command is sent from a PC to a smart device while connecting the LG smart device to the PC with a data cable, the smart device switches to download mode.

As a result of reverse engineering the LG bootloader using IDA Pro, we were able to analyze LG download commands that are not available to the public. These include commands implemented by LG for software debugging purposes as well as commands used for firmware updates. From our analysis of the SBL3 bootloader, we found that there is a main memory read command that can access the main memory and acquire its data. These commands are expected to be used for memory debugging purposes. Fig. 7 shows the main memory read command which was found by decompiling the SBL3 bootloader in the LG Optimus G pro model. After the device switches to download mode, we can acquire the entire main memory data using the main memory information acquisition commands and main memory read commands. Table 3 shows the commands used for the main memory data acquisition in LG.

Main memory information command: 0x10

After the device enters the download mode, the command for main memory information acquisition (0x10) is sent to the smart



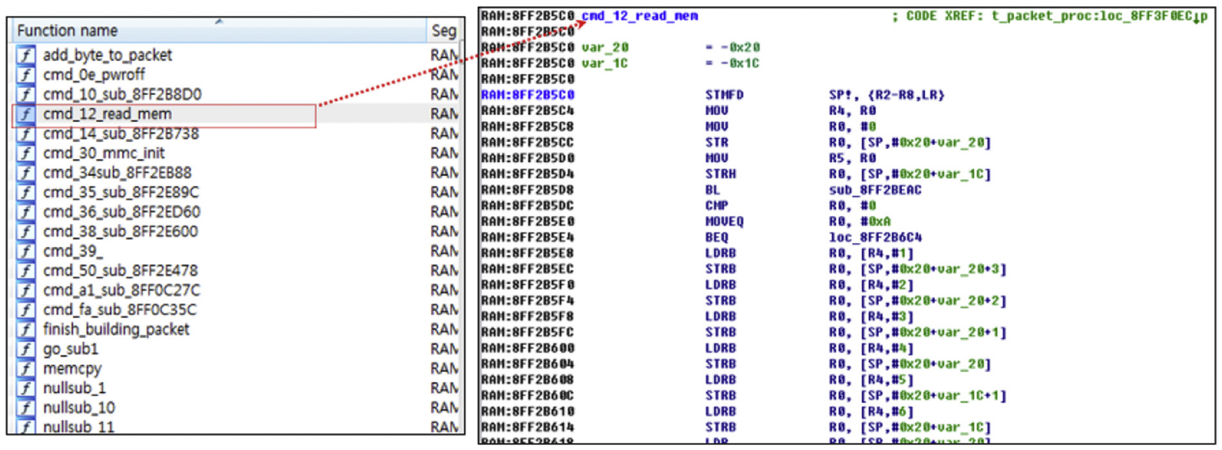
```
Send 0x400 bytes to the device
 65 00 00 00 02 00 00 00 B7 70 FD FF 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
Get 0x1f4 bytes from the device
 6F 74 2E 62 6F 6F 74 5F 73 61 6C 65 73 63 6F 64   ot.boot_salescod
 65 3D 20 61 6E 64 72 6F 69 64 62 6F 6F 74 2E 6F   e= androidboot.o
 64 69 6E 5F 64 6F 77 6E 6C 6F 61 64 3D 31 20 61   din_download=1 a
 6E 64 72 6F 69 64 62 6F 6F 74 2E 62 6F 6F 74 6C   ndroidboot.bootl
 6F 61 64 65 72 3D 45 32 31 30 4B 4B 54 55 4B 4E   oader=E210KKTUKN
 49 33 20 61 6E 64 72 6F 69 64 62 6F 6F 74 2E 73   I3 androidboot.s
 65 6C 69 6E 75 78 3D 65 6E 66 6F 72 63 69 6E 67   elinux=enforcing
 20 61 6E 64 72 6F 69 64 62 6F 6F 74 2E 68 61 72    androidboot.har
 64 77 61 72 65 3D 53 48 56 2D 45 32 31 30 4B 20   dware=SHV-E210K 
 61 6E 64 72 6F 69 64 62 6F 6F 74 2E 77 61 72 72   androidboot.warr
 61 6E 74 79 73 68 75 74 64 6F 77 6E 20 2D 0A 3C   antyshutdown -.<
 36 3E 5B 20 31 30 35 35 2E 32 33 34 30 34 30 5D   6>[ 1055.234040]
 20 63 30 20 20 20 32 30 30 30 20 64 75 6D 6D 79    c0   2000 dummy
 20 31 37 2D 30 30 32 35 3A 20 73 68 75 74 64 6F    17-0025: shutdo
 77 6E 20 2B 3A 20 69 32 63 5F 64 65 76 69 63 65   wn +: i2c_device
 5F 73 68 75 74 64 6F 77 6E 2B 30 78 30 2F 30 78   _shutdown+0x0/0x
 34 63 0A 3C 36 3E 5B 20 31 30 35 35 2E 32 33 34   4c.<6>[ 1055.234
 31 30 31 5D 20 63 30 20 20 20 32 30 30 30 20 64   101] c0   2000 d
```

```
Send 0x400 bytes to the device
 65 00 00 00 02 00 00 00 11 00 00 00 00 00 00 00   e...............
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
Get 0x1000 bytes from the device
 6F 6D 2F 73 65 63 2F 61 6E 64 72 6F 69 64 2F 61   om/sec/android/a
 70 70 2F 63 6C 6F 63 6B 70 61 63 6B 61 67 65 2F   pp/clockpackage/
 61 6C 61 72 6D 2F 41 6C 61 72 6D 44 65 74 61 69   alarm/AlarmDetai
 6C 24 37 3B 00 37 4C 63 6F 6D 2F 73 65 63 2F 61   l$7;.7Lcom/sec/a
 6E 64 72 6F 69 64 2F 61 70 70 2F 63 6C 6F 63 6B   ndroid/app/clock
 70 61 63 6B 61 67 65 2F 61 6C 61 72 6D 2F 41 6C   package/alarm/Al
 61 72 6D 44 65 74 61 69 6C 24 38 30 3B 00 37 4C   armDetail$80;.7L
 63 6F 6D 2F 73 65 63 2F 61 6E 64 72 6F 69 64 2F   com/sec/android/
 61 70 70 2F 63 6C 6F 63 6B 70 61 63 6B 61 67 65   app/clockpackage
 2F 61 6C 61 72 6D 2F 41 6C 61 72 6D 44 65 74 61   /alarm/AlarmDeta
 69 6C 24 38 31 3B 00 37 4C 63 6F 6D 2F 73 65 63   il$81;.7Lcom/sec
 2F 61 6E 64 72 6F 69 64 2F 61 70 70 2F 63 6C 6F   /android/app/clo
 63 6B 70 61 63 6B 61 67 65 2F 61 6C 61 72 6D 2F   ckpackage/alarm/
 41 6C 61 72 6D 44 65 74 61 69 6C 24 38 32 3B 00   AlarmDetail$82;.
 37 4C 63 6F 6D 2F 73 65 63 2F 61 6E 64 72 6F 69   7Lcom/sec/androi
 64 2F 61 70 70 2F 63 6C 6F 63 6B 70 61 63 6B 61   d/app/clockpacka
 67 65 2F 61 6C 61 72 6D 2F 41 6C 61 72 6D 44 65   ge/alarm/AlarmDe
```

Fig. 6. Example of main memory data acquisition (Galaxy S3).



Function name	Seg
add_byte_to_packet	RAM
cmd_0e_pwroff	RAM
cmd_10_sub_8FF2B8D0	RAM
cmd_12_read_mem	RAM
cmd_14_sub_8FF2B738	RAM
cmd_30_mmc_init	RAM
cmd_34sub_8FF2EB88	RAM
cmd_35_sub_8FF2E89C	RAM
cmd_36_sub_8FF2ED60	RAM
cmd_38_sub_8FF2E600	RAM
cmd_39_	RAM
cmd_50_sub_8FF2E478	RAM
cmd_a1_sub_8FF0C27C	RAM
cmd_fa_sub_8FF0C35C	RAM
finish_building_packet	RAM
go_sub1	RAM
memcpy	RAM
nullsub_1	RAM
nullsub_10	RAM
nullsub_11	RAM

```
RAM:8FF2B5C0 cmd_12_read_mem                          ; CODE XREF: t_packet_proc:loc_8FF3F0EC↓p
RAM:8FF2B5C0
RAM:8FF2B5C0 var_20          = -0x20
RAM:8FF2B5C0 var_1C          = -0x1C
RAM:8FF2B5C0
RAM:8FF2B5C0                 STMFD           SP!, {R2-R8,LR}
RAM:8FF2B5C4                 MOV             R4, R0
RAM:8FF2B5C8                 MOV             R0, #0
RAM:8FF2B5CC                 STR             R0, [SP,#0x20+var_20]
RAM:8FF2B5D0                 MOV             R5, R0
RAM:8FF2B5D4                 STRH            R0, [SP,#0x20+var_1C]
RAM:8FF2B5D8                 BL              sub_8FF2BEAC
RAM:8FF2B5DC                 CMP             R0, #0
RAM:8FF2B5E0                 MOVEQ           R0, #0xA
RAM:8FF2B5E4                 BEQ             loc_8FF2B6C4
RAM:8FF2B5E8                 LDRB            R0, [R4,#1]
RAM:8FF2B5EC                 STRB            R0, [SP,#0x20+var_20+3]
RAM:8FF2B5F0                 LDRB            R0, [R4,#2]
RAM:8FF2B5F4                 STRB            R0, [SP,#0x20+var_20+2]
RAM:8FF2B5F8                 LDRB            R0, [R4,#3]
RAM:8FF2B5FC                 STRB            R0, [SP,#0x20+var_20+1]
RAM:8FF2B600                 LDRB            R0, [R4,#4]
RAM:8FF2B604                 STRB            R0, [SP,#0x20+var_20]
RAM:8FF2B608                 LDRB            R0, [R4,#5]
RAM:8FF2B60C                 STRB            R0, [SP,#0x20+var_1C+1]
RAM:8FF2B610                 LDRB            R0, [R4,#6]
RAM:8FF2B614                 STRB            R0, [SP,#0x20+var_1C]
```

Fig. 7. Main memory read command found from reverse engineering of the LG SBL3 bootloader.

Table 3
Commands used for the main memory data acquisition in LG.

Command	Description
0x3A	Switch to download mode
0x10	Get main memory information
0x14	Read main memory (1 GB)
0x12	Read main memory (2 GB)
0x0A	Reset system

device to obtain information about main memory. The obtained information includes the start address and the size of the main memory connected to the External Bus Interface (EBI) (External Bus Interface, 2015). It also contains debug information that can be used for showing the CPU register values and for memory debugging. Fig. 8 shows the process of acquiring the main memory information of the Optimus G Pro model.

Main memory read command: 0x12, 0x14

Based on this information, we can obtain the main memory data using the main memory read command. If the size of the main memory is 2 GB (from the obtained main memory information), the main memory data can be obtained using the 0x12 read command. The main memory information also reports that the start address of the memory is 0x80000000. When the read commands are sent to the smart device, the dump data (4 KB segments per command) are sent to the PC. Similarly, if the size of main memory is reported to be 1 GB, the main memory data can be obtained using the 0x14 read command. In this case, the main memory information reports that the start address of the memory is 0x40000000. Fig. 9 shows the format of the main memory read command and the process of acquiring the main memory data of the Optimus G Pro model.

Main memory image

The acquisition method proposed in this study can physically acquire the entire main memory, and the acquired physical image can be used for forensic analysis. For this, we used the Volatility tool to analyze the dumped images acquired by AMD. Volatility supports the analysis of a main memory image acquired through the LiME method. Therefore, we implemented a function in AMD to convert the acquired data to the LiME image format.

LiME creates an acquisition file according to the system RAM areas identified in the*/proc/iomem* structure. The format of the main memory file acquired by LiME consists of a header that stores the physical address information of each system RAM area. Fig. 10 shows the LiME header structure. The entire main memory area obtained by AMD is converted to the LiME-based image format by attaching LiME headers according to the system RAM areas in*/proc/iomem*.

Android main memory dump tool

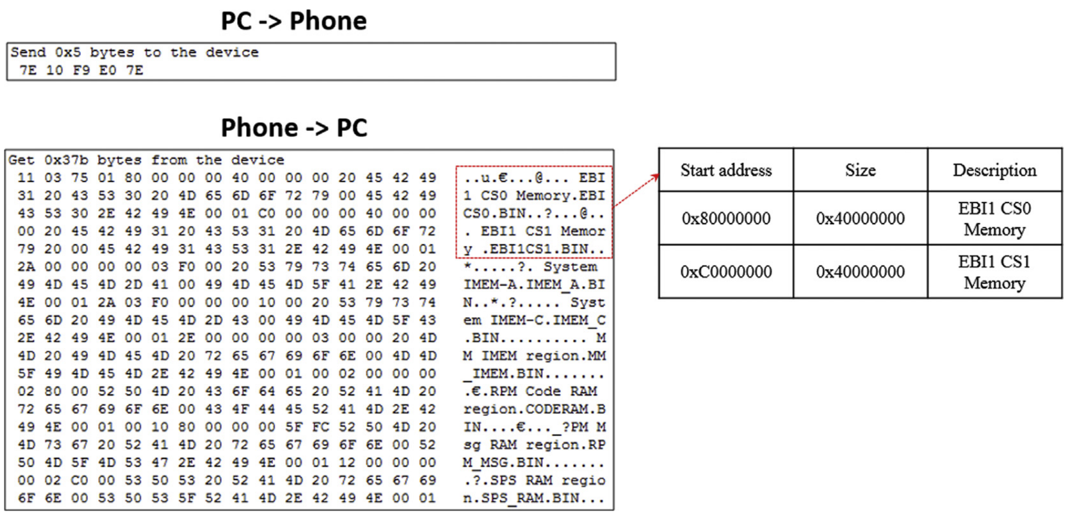
We developed the Android Main memory Dump (AMD) tool to acquire the main memory data of smart devices in forensic investigations, as described in Section Android main memory acquisition in firmware update mode. This tool was developed using the C++ language. After a smart device is connected to a PC using a data cable, the tool provides a simple GUI environment that allows a user to perform main memory acquisition by clicking on the buttons. The tool was developed so that even a user who is not a professional forensic expert can easily and quickly acquire main memory data for use in forensic investigations. Fig. 11 shows the process of acquiring the main memory data of a Galaxy Gear, a Samsung smartwatch, using the AMD main memory data acquisition tool.

Main memory data acquisition

The current AMD prototype supports physical memory data acquisition for Samsung smartphones and smartwatches and LG smartphones. As of writing, we have verified that AMD can correctly acquire a main memory image from the 50 device models listed in Table 4. In the future, we hope that our ongoing research will continue to expand this list.

Before connecting a smart device to a PC using a USB data cable, the USB driver must be installed in advance. After connecting the USB data cable, the user should click the button "Switch to FU mode." This sends download-mode-switching commands to the smart device which switches the device into download mode. After the switch to download mode, the tool obtains the main memory information using the main memory information commands and outputs the information to the Partition Information window.

When the user clicks the "Full Dump" button, the entire physical acquisition process is performed from the main memory start address to the end address using the Samsung PIT read command or the LG main memory read command. Once the acquisition

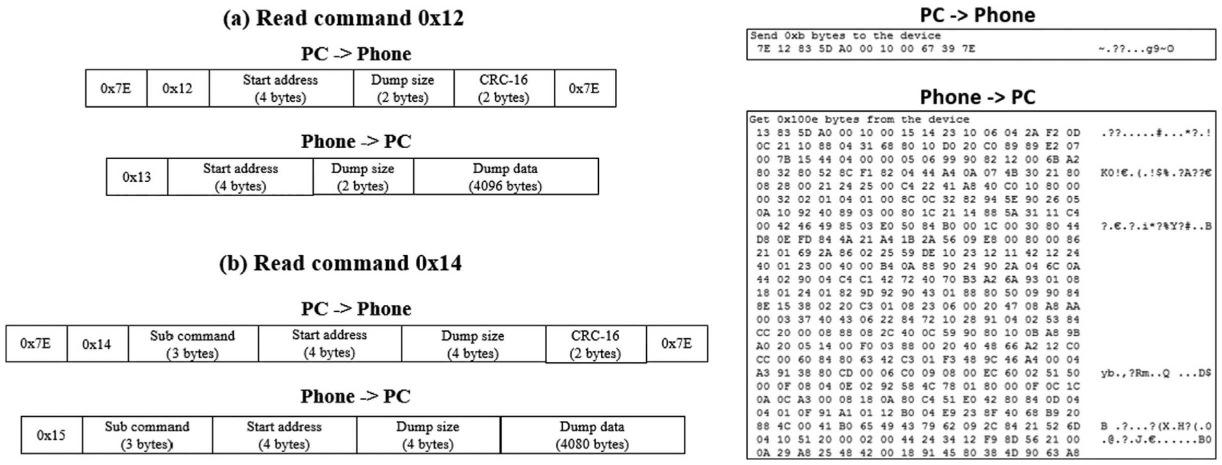


```
Send 0x5 bytes to the device
 7E 10 F9 E0 7E
```

```
Get 0x37b bytes from the device
 11 03 75 01 80 00 00 00 40 00 00 00 20 45 42 49   ..u.€...@... EBI
 31 20 43 53 30 20 4D 65 6D 6F 72 79 00 45 42 49   1 CS0 Memory.EBI
 43 53 30 2E 42 49 4E 00 01 C0 00 00 00 40 00 00   CS0.BIN..?...@..
 00 20 45 42 49 31 20 43 53 31 20 4D 65 6D 6F 72   . EBI1 CS1 Memor
 79 20 00 45 42 49 31 43 53 31 2E 42 49 4E 00 01   y .EBI1CS1.BIN..
 2A 00 00 00 00 03 F0 00 20 53 79 73 74 65 6D 20   *.....?. System
 49 4D 45 4D 2D 41 00 49 4D 45 4D 5F 41 2E 42 49   IMEM-A.IMEM_A.BI
 4E 00 01 2A 03 F0 00 00 00 10 00 20 53 79 73 74   N..*.?..... Syst
 65 6D 20 49 4D 45 4D 2D 43 00 49 4D 45 4D 5F 43   em IMEM-C.IMEM_C
 2E 42 49 4E 00 01 2E 00 00 00 00 03 00 00 20 4D   .BIN.......... M
 4D 20 49 4D 45 4D 20 72 65 67 69 6F 6E 00 4D 4D   M IMEM region.MM
 5F 49 4D 45 4D 2E 42 49 4E 00 01 00 02 00 00 00   _IMEM.BIN.......
 02 80 00 52 50 4D 20 43 6F 64 65 20 52 41 4D 20   .€.RPM Code RAM
 72 65 67 69 6F 6E 00 43 4F 44 45 52 41 4D 2E 42   region.CODERAM.B
 49 4E 00 01 00 10 80 00 00 00 5F FC 52 50 4D 20   IN....€..._?PM M
 4D 73 67 20 52 41 4D 20 72 65 67 69 6F 6E 00 52   sg RAM region.RP
 50 4D 5F 4D 53 47 2E 42 49 4E 00 01 12 00 00 00   M_MSG.BIN.......
 00 02 C0 00 53 50 53 20 52 41 4D 20 72 65 67 69   .?.SPS RAM regio
 6F 6E 00 53 50 53 5F 52 41 4D 2E 42 49 4E 00 01   n.SPS_RAM.BIN...
```

Start address	Size	Description
0x80000000	0x40000000	EBI1 CS0 Memory
0xC0000000	0x40000000	EBI1 CS1 Memory

Fig. 8. The process of acquiring the main memory information (Optimus G Pro).



```
Send 0xb bytes to the device
 7E 12 83 5D A0 00 10 00 67 39 7E                       ~.??...g9~O
```

```
Get 0x100e bytes from the device
 13 83 5D A0 00 10 00 15 14 23 10 06 04 2A F2 0D   .??.....#...*?.!
 0C 21 10 88 04 31 68 80 10 D0 20 C0 89 89 E2 07
 00 7B 15 44 04 00 00 05 06 99 90 82 12 00 6B A2
 80 32 80 52 8C F1 82 04 44 A4 0A 07 4B 30 21 80   KO!€.(.!$%.?A??€
 08 28 00 21 24 25 00 C4 22 41 A8 40 C0 10 80 00
 00 32 02 01 04 01 00 8C 0C 32 82 94 5E 90 26 05
 0A 10 92 40 89 03 00 80 1C 21 14 88 5A 31 11 C4
 00 42 46 49 85 03 E0 50 84 B0 00 1C 00 30 80 44   ?.€.?.i*?%Y?#..B
 D8 0E FD 84 4A 21 A4 1B 2A 56 09 E8 00 80 00 86
 21 01 69 2A 86 02 25 59 DE 10 23 12 11 42 12 24
 40 01 23 00 40 00 B4 0A 88 90 24 90 2A 04 6C 0A
 44 02 90 04 C4 C1 42 72 40 70 B3 A2 6A 93 01 08
 18 01 24 01 82 9D 92 90 43 01 88 80 50 09 90 84
 8E 15 38 02 20 C3 01 08 23 06 00 20 47 08 A8 AA
 00 03 37 40 43 06 22 84 72 10 28 91 04 02 53 84
 CC 20 00 08 88 08 2C 40 0C 59 90 80 10 0B A8 9B
 A0 20 05 14 00 F0 03 88 00 20 40 48 66 A2 12 C0
 CC 00 60 84 80 63 42 C3 01 F3 48 9C 46 A4 00 04
 A3 91 38 80 CD 00 06 C0 09 08 00 EC 60 02 51 50   yb.,?Rm..Q ...D$
 00 0F 08 04 0E 02 92 58 4C 78 01 80 00 0F 0C 1C
 0A 0C A3 00 08 18 0A 80 C4 51 E0 42 80 84 0D 04
 04 01 0F 91 A1 01 12 B0 04 E9 23 8F 40 68 B9 20
 88 4C 00 41 B0 65 49 43 79 62 09 2C 84 21 52 6D   B .?...?(X.H?(.O
 04 10 51 20 00 02 00 44 24 34 12 F9 8D 56 21 00   .@.?.J.€......BO
 0A 29 A8 25 48 42 00 18 91 45 80 38 4D 90 63 A8
```

Fig. 9. Format of the memory read command and the process of acquiring the main memory data (Optimus G Pro).

process is completed, the Dump Log window shows the investigator information, acquisition tool information, and acquired image information, as shown in Fig. 11. The acquired image information includes the acquisition start time, acquisition image storage location, and acquisition end time, and the MD5 hash value of the imaged file. If the "Stop Dump" button is clicked during the acquisition process, the process is stopped. After acquisition completes, the system can be restarted by clicking the "Reboot" button. Note that AMD makes no persistent changes to the device, but a reboot is required to continue using the device in normal operation mode.

Conversion to the LiME image format

The image file acquired with the AMD tool has a raw data format that represents the physical acquisition of the entire main memory area. To more easily analyze the image file acquired by the AMD tool, when the user can click the button "Convert to LiME format", the entire main memory image file acquired by AMD is converted to the LiME image format.

Experiments

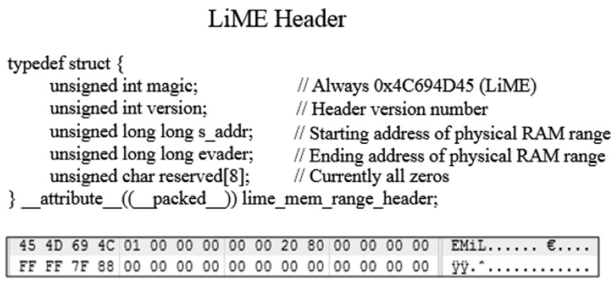
The requirements for the physical acquisition of the main memory data of a smart device in forensic investigations are as follows. First, the main memory data acquisition process should be performed without a system restart for the smart device. Second, it must support a rapid evidence collection and an integrity check of the acquired image. Third, the analysis of the acquired image must be provided using standard forensic analysis tools. Fourth, it should be possible to extract evidence of the user's actions from the acquired image. With these requirements, we compared the AMD acquisition method proposed in this study with existing research methods, namely, the LiME, AMExtractor, and BMA methods. Table 5 shows the experimental results obtained. The experimental results below were obtained by averaging across 5 acquisitions per device. From the 50 device models (shown in Table 4) that we have verified are supported by AMD, we selected 12 representative models for these experiments. The 12 representative models are: SHV-E210S, SHV-E210K, SHV-E330S, SHV-E330K, SM-N900S, SM-N9005, LG-F180S, LG-F180L, LG-F240S, LG-F240K, LG-E960, SM-V700.

Live acquisition

We performed live main memory acquisition experiments using the existing methods and AMD, as shown in Table 5. LiME requires root privilege to acquire main memory data. A custom kernel and original kernel source code are only necessary for cases where the default kernel does not allow inserting kernel modules and the kernel configuration and toolchains must be exactly the same as the stock firmware running on the device, respectively. During our evaluation, we tested a number of LG devices (Optimus G, Optimus G Pro, Nexus 4) which require flashing a custom kernel and the original kernel source to use LiME. Moreover, because LiME uses the ADB protocol, it is impossible to acquire the data from a smart device with a screen lock (USB debugging mode disabled). The AMExtractor method also requires root privilege and uses the ADB protocol to upload the dump code to the smart device. It cannot access the main memory of a smart device that has a screen lock. The BMA method requires the original recovery image and a process to flash a custom recovery image, which needs to restart the system and therefore limits its application in forensic investigations. AMD can acquire data through the main memory read command — after switching the device to firmware update mode through a software command without a system restart. In this way, AMD solves the limitations of existing methods such as obtaining a root privilege, replacing with a custom image, and bypassing the screen lock. Even in the case of a smart device with a screen lock, since the acquisition process is performed by switching the device to firmware update mode, the proposed method can still acquire a main memory image.

Evaluation of AMD

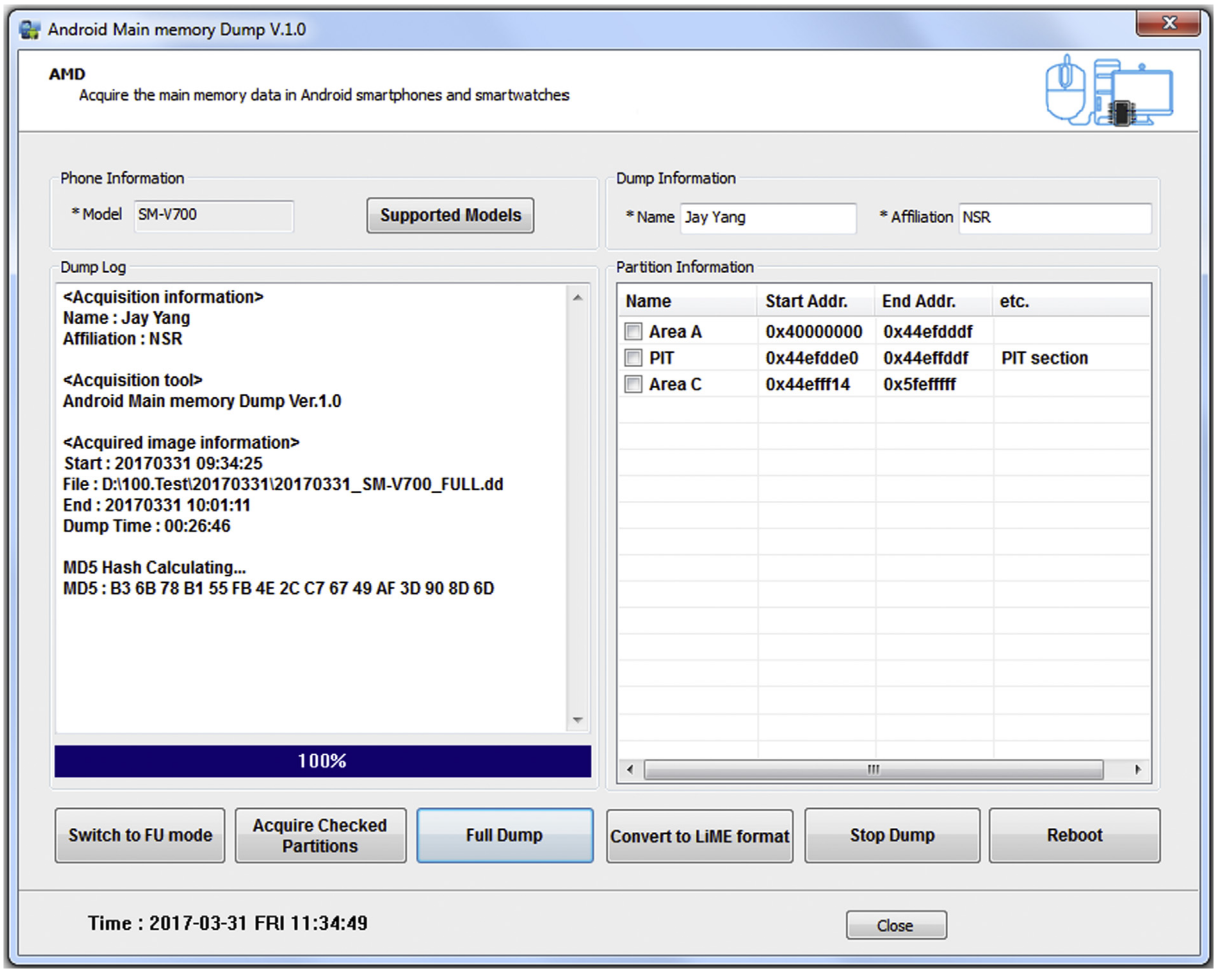
We verified the integrity of the images acquired with the AMD tool. In these experiments, main memory images were acquired with LiME first and then AMD. Both LiME and AMD recovered a


LiME Header

```
typedef struct {
        unsigned int magic;             // Always 0x4C694D45 (LiME)
        unsigned int version;           // Header version number
        unsigned long long s_addr;      // Starting address of physical RAM range
        unsigned long long evader;      // Ending address of physical RAM range
        unsigned char reserved[8];      // Currently all zeros
} __attribute__((__packed__)) lime_mem_range_header;
```

```
45 4D 69 4C 01 00 00 00 00 00 20 80 00 00 00 00   EMiL...... €....
FF FF 7F 88 00 00 00 00 00 00 00 00 00 00 00 00   ÿÿ.^............
```

Fig. 10. LiME header structure.



*Model: SM-V700
*Name: Jay Yang
*Affiliation: NSR

```
<Acquisition information>
Name : Jay Yang
Affiliation : NSR

<Acquisition tool>
Android Main memory Dump Ver.1.0

<Acquired image information>
Start : 20170331 09:34:25
File : D:\100.Test\20170331\20170331_SM-V700_FULL.dd
End : 20170331 10:01:11
Dump Time : 00:26:46

MD5 Hash Calculating...
MD5 : B3 6B 78 B1 55 FB 4E 2C C7 67 49 AF 3D 90 8D 6D
```

Name	Start Addr.	End Addr.	etc.
☐ Area A	0x40000000	0x44efdddf	
☐ PIT	0x44efdde0	0x44effddf	PIT section
☐ Area C	0x44efff14	0x5fefffff	

Time : 2017-03-31 FRI 11:34:49

Fig. 11. Screenshot of the Android Main memory Dump tool (Galaxy Gear).

total of 3,870,712 pages. The memory contents dumped by LiME and AMD were nearly the same. Both techniques collected the same number of pages from the device and 99.4% (Identical pages are 3,847,487) of those pages were identical. However, several pages are not identical. This occurs because some code and data areas of the kernel are changed while switching to the firmware update mode. By comparison, the loading of the LiME module also modifies small portions of kernel code and data during module insertion, but LiME's modifications are not reflected here.

In order to show the performance of the AMD tool, the acquisition times of 12 Android models using the AMD tool are shown in Table 6.

Table 4
Models that we have verified are supported by the AMD tool.

Model	Name			
Galaxy S2	SHW-M250S	SHW-M250K	SHW-M250L	
Galaxy S3	SHV-E210S	SHV-E210K	SHV-E210L	SHW-M440S
Galaxy S4	SHV-E330S	SHV-E330K	SHV-E330L	GT-I9506
Galaxy Note 2	SHV-E250S	SHV-E250K	SHV-E250L	GT-N7100
Galaxy Note 3	SM-N900S	SM-N900K	SM-N900L	SM-N9005
Galaxy Grand	SHV-E270S	SHV-E270K	SHV-E270L	
Galaxy Pop	SHV-E220S			
Galaxy Note 8	SHW-M500W			
Galaxy Note 10.1	SM-P600			
Galaxy Gear	SM-V700			
Optimus G	LG-F180S	LG-F180K	LG-F180L	LG-E975
Optimus GK	LG-F220K			
Optimus Gx	LG-F310L			
Optimus G Pro	LG-F240S	LG-F240K	LG-F240L	LG-E985
Optimus Pad 8.3	LG-V500			
Optimus Vu1	LG-F100S	LG-F100K	LG-F100L	
Optimus Vu2	LG-F200S	LG-F200K	LG-F200L	
Optimus LTE1	LG-LU6200	LG-SU640		
Optimus LTE2	LG-F160S	LG-F160K	LG-F160L	
Optimus LTE3	LG-F260S			
Nexus 4	LG-E960			

Volatility analysis

In addition to comparing the contents of the pages in LiME and AMD memory images, we also verified that the evidence recovered by Volatility plug-ins from AMD memory images is correct. Before collecting an AMD memory image, we took note of the following information which would serve as the ground truth for the Volatility plugins' results: 1) the process list entries for the *linux_pslist* and *linux_psscan* plugins, 2) the output of/*proc/iomem* for the *linux_iomem* plugin, and 3) the memory maps of the processes for the *linux_proc_maps* plugin. We then collected a memory image with AMD and verified that the results of those plugins correctly matched our ground truth. The results were as follows: *linux_pslist* reported 194 entries, *linux_iomem* found 108 entries, *linux_psscan* located 2,596 entries (which we manually verified were all the live processes as well as valid previously-killed processes), and *linux_proc_maps* recovered 26,968 entries. These results confirm that AMD correctly images the data in main memory.

Table 5
Experimental results.

Requirements	AMD	LiME	AMExtractor	BMA
Root privilege	No	Yes	Yes	No
Original kernel source	No	Some Models[a]	No	No
Process for flashing of custom image	No	Some Models[b]	No	Yes
Prior disabling of screen lock or user password	No	Yes	Yes	No

[a] Older Android models may not require original kernel source because only modern Android firmware enforces CRC checks.
[b] Creating and flashing a custom kernel is necessary when the default kernel does not support inserting kernel modules.

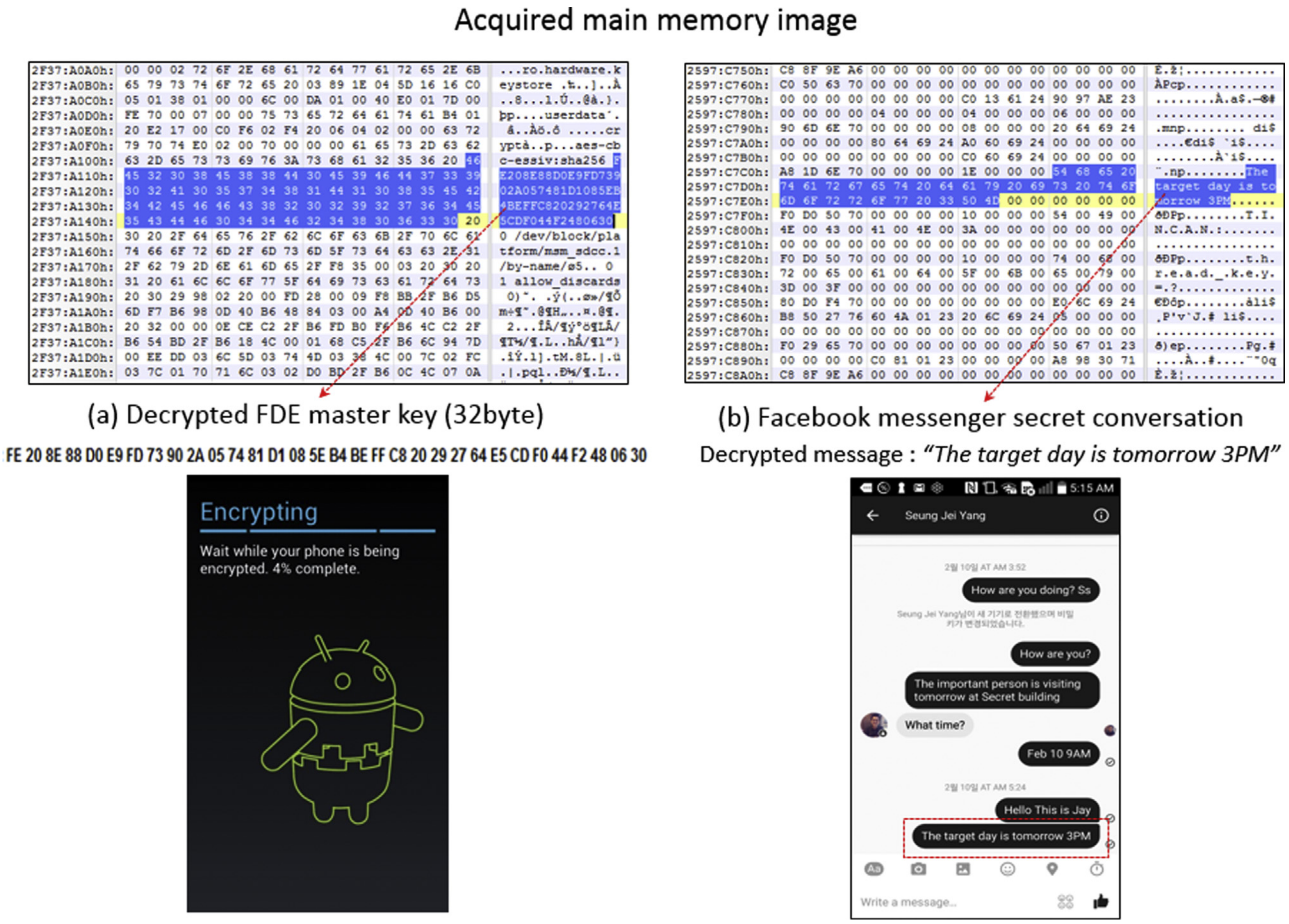
Table 6
Acquisition time of main memory data by using AMD tool.

Model	Name	OS Version	Size	Acquisition time
Galaxy S3	SHV-E210S	4.1.2	2 GB	23 min
	SHV-E210K	4.4.4	2 GB	23 min
Galaxy S4	SHV-E330S	4.2.2	2 GB	50 min
	SHV-E330K	4.4.2	2 GB	50 min
Galaxy Note 3	SM-N900S	4.3	3 GB	52 min
	SM-N9005	4.4.2	3 GB	52 min
Optimus G	LG-F180S	4.4.4	2 GB	13 min
	LG-F180L	4.4.2	2 GB	13 min
Optimus G Pro	LG-F240S	5.0.1	2 GB	12 min
	LG-F240K	5.0	2 GB	12 min
Nexus 4	LG-E960	5.1.1	2 GB	13 min
Galaxy gear	SM-V700	4.1	512 MB	26 min

Acquisition of user information

Recently, FDE, FBE, and SNS conversation encryption technologies have been applied to protect NAND flash data. In this case, even if the NAND flash data are acquired, the data encryption cannot be deciphered by existing NAND flash forensic technologies. To solve this problem, it is necessary to extract important user information such as the FDE master key, user conversation messages, etc., from the main memory where they remain decrypted. However, since the main memory is a volatile memory, its data are only retained when the system is powered on and are lost if the power is lost or if the system is restarted. AMD can extract evidence left by the user as is because it can obtain the entire main memory data from a smart device without a system restart. The decrypted FDE master key, user passwords, and decrypted SNS conversation message from the image obtained by the proposed method can be then extracted.

Fig. 12 shows a portion of an image acquired with the AMD tool. This shows that the user's data (such as FDE keys and SNS conversation messages) remain intact in the image acquired by the AMD method. In the case of recovering the FDE master key, after encrypting the Android NAND flash, we dumped the main memory with the AMD tool. We extracted the FDE master key (shown in Fig. 12) from the acquired image by searching for the cipher identification string “AES-CBC-ESSIV:SHA256”. After acquiring the


Acquired main memory image
(a) Decrypted FDE master key (32byte)
(b) Facebook messenger secret conversation
Encrypting
Wait while your phone is being encrypted. 4% complete.
5:15 AM
Seung Jei Yang
Write a message...

```
2F37:A0A0h: 00 00 02 72 6F 2E 68 61 72 64 77 61 72 65 2E 6B  ...ro.hardware.k
2F37:A0B0h: 65 79 73 74 6F 72 65 20 03 89 1E 04 5D 16 16 C0  eystore .‰..]..À
2F37:A0C0h: 05 01 38 01 00 00 6C 00 DA 01 00 40 E0 01 7D 00  ..8...l.Ú..@à.}.
2F37:A0D0h: FE 70 00 07 00 00 75 73 65 72 64 61 74 61 B4 01  þp....userdata´.
2F37:A0E0h: 20 E2 17 00 C0 F6 02 F4 20 06 04 02 00 00 63 72   â..Àö.ô .....cr
2F37:A0F0h: 79 70 74 E0 02 00 70 00 00 00 61 65 73 2D 63 62  yptà..p...aes-cb
2F37:A100h: 63 2D 65 73 73 69 76 3A 73 68 61 32 35 36 20 46  c-essiv:sha256 F
2F37:A110h: 45 32 30 38 45 38 38 44 30 45 39 46 44 37 33 39  E208E88D0E9FD739
2F37:A120h: 30 32 41 30 35 37 34 38 31 44 31 30 38 35 45 42  02A057481D1085EB
2F37:A130h: 34 42 45 46 46 43 38 32 30 32 39 32 37 36 34 45  4BEFFC820292764E
2F37:A140h: 35 43 44 46 30 34 34 46 32 34 38 30 36 33 30 20  5CDF044F2480630 
2F37:A150h: 30 20 2F 64 65 76 2F 62 6C 6F 63 6B 2F 70 6C 61  0 /dev/block/pla
2F37:A160h: 74 66 6F 72 6D 2F 6D 73 6D 5F 73 64 63 63 2E 31  tform/msm_sdcc.1
2F37:A170h: 2F 62 79 2D 6E 61 6D 65 2F F8 35 00 03 20 30 20  /by-name/ø5.. 0 
2F37:A180h: 31 20 61 6C 6C 6F 77 5F 64 69 73 63 61 72 64 73  1 allow_discards
2F37:A190h: 20 30 29 98 02 20 00 FD 28 00 09 F8 BB 2F B6 D5   0)˜. .ý(..ø»/¶Õ
2F37:A1A0h: 6D F7 B6 98 0D 40 B6 48 84 03 00 A4 0D 40 B6 00  m÷¶˜.@¶H„..¤.@¶.
2F37:A1B0h: 20 32 00 00 0E CE C2 2F B6 FD B0 F6 B6 4C C2 2F   2...ÎÂ/¶ý°ö¶LÂ/
2F37:A1C0h: B6 54 BD 2F B6 18 4C 00 01 68 C5 2F B6 6C 94 7D  ¶T½/¶.L..hÅ/¶l”}
2F37:A1D0h: 00 EE DD 03 6C 5D 03 74 4D 03 38 4C 00 7C 02 FC  .îÝ.l].tM.8L.|.ü
2F37:A1E0h: 03 7C 01 70 71 6C 03 02 D0 BD 2F B6 0C 4C 07 0A  .|.pql..Ð½/¶.L..
```

FE 20 8E 88 D0 E9 FD 73 90 2A 05 74 81 D1 08 5E B4 BE FF C8 20 29 27 64 E5 CD F0 44 F2 48 06 30

```
2597:C750h: C8 8F 9E A6 00 00 00 00 00 00 00 00 00 00 00 00  È.ž¦............
2597:C760h: C0 50 63 70 00 00 00 00 00 00 00 00 00 00 00 00  ÀPcp............
2597:C770h: 00 00 00 00 00 00 00 00 C0 13 61 24 90 97 AE 23  ........À.a$.—®#
2597:C780h: 00 00 00 00 04 00 00 00 04 00 00 00 06 00 00 00  ................
2597:C790h: 90 6D 6E 70 00 00 00 00 08 00 00 00 20 64 69 24  .mnp........ di$
2597:C7A0h: 00 00 00 00 80 64 69 24 A0 60 69 24 00 00 00 00  ....€di$ `i$....
2597:C7B0h: 00 00 00 00 00 00 00 00 C0 60 69 24 00 00 00 00  ........À`i$....
2597:C7C0h: A8 1D 6E 70 00 00 00 00 1E 00 00 00 54 68 65 20  ¨.np........The 
2597:C7D0h: 74 61 72 67 65 74 20 64 61 79 20 69 73 20 74 6F  target day is to
2597:C7E0h: 6D 6F 72 72 6F 77 20 33 50 4D 00 00 00 00 00 00  morrow 3PM......
2597:C7F0h: F0 D0 50 70 00 00 00 00 10 00 00 00 54 00 49 00  ðÐPp........T.I.
2597:C800h: 4E 00 43 00 41 00 4E 00 3A 00 00 00 00 00 00 00  N.C.A.N.:.......
2597:C810h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
2597:C820h: F0 D0 50 70 00 00 00 00 10 00 00 00 74 00 68 00  ðÐPp........t.h.
2597:C830h: 72 00 65 00 61 00 64 00 5F 00 6B 00 65 00 79 00  r.e.a.d._.k.e.y.
2597:C840h: 3D 00 3F 00 00 00 00 00 00 00 00 00 00 00 00 00  =.?.............
2597:C850h: 80 D0 F4 70 00 00 00 00 00 00 00 00 E0 6C 69 24  €Ðôp........àli$
2597:C860h: B8 50 27 76 60 4A 01 23 20 6C 69 24 05 00 00 00  ¸P'v`J.# li$....
2597:C870h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
2597:C880h: F0 29 65 70 00 00 00 00 00 00 00 00 50 67 01 23  ð)ep........Pg.#
2597:C890h: 00 00 00 00 C0 81 01 23 00 00 00 00 A8 98 30 71  ....À..#....¨˜0q
2597:C8A0h: C8 8F 9E A6 00 00 00 00 00 00 00 00 00 00 00 00  È.ž¦............
```

Decrypted message : *“The target day is tomorrow 3PM”*

2월 10일 AT AM 3:52
How are you doing? Ss
Seung Jei Yang님이 새 기기로 전환했으므로 비밀 키가 변경되었습니다.
How are you?
The important person is visiting tomorrow at Secret building
What time?
Feb 10 9AM
2월 10일 AT AM 5:24
Hello This is Jay
The target day is tomorrow 3PM

Fig. 12. Example of extracting the user information from the image acquired by the AMD tool (Optimus G Pro).

NAND flash memory, it could directly be decrypted with the extracted FDE master key. We then confirmed that the decrypted NAND flash memory image could be correctly analyzed by the Cellebrite forensic tool.

In the case of the Facebook user conversation messages, we searched for the messages that had been sent to the smartphone within the memory image acquired by the AMD method. We then confirmed that the conversation messages remain intact within the acquired image. This result highlights the fact that the AMD method acquires the main memory data without restarting the system.

The FDE master key and Facebook conversation messages are encrypted when stored to NAND flash memory. So there are only brute-force encryption-breaking methods to recover their decrypted values. The existing NAND flash forensic technologies cannot solve these issues. However, since they are decrypted in main memory, the data can be acquired with AMD.

Conclusion

Existing research on acquiring Android main memory data are hardly applied in practice because they do not solve the constraints of root privilege escalation, custom kernel replacement, and screen lock bypass. To solve this problem, we proposed AMD, a tool for acquiring the main memory data of smart devices, which can easily be deployed in forensic investigations. In designing AMD, we reverse engineered the firmware update program, firmware update protocol, and firmware update process provided by the manufacturers. From the results of this analysis, we found that physical acquisition of all data in main memory is possible after the device is switched to firmware update mode without a system restart. Based on these findings, we developed AMD to support the acquisition of the main memory from Android smartphones and smartwatches

Comparison to existing acquisition methods showed that the proposed method can acquire main memory data without a system restart, root privilege escalation, custom kernel, and screen lock bypass. We confirmed that, from the obtained image, the evidence stored in the main memory that the user had left can be extracted. We also verified the integrity of the acquired image and confirmed that it can be further analyzed through the Volatility tool.

References

AMExtractor, 2016. https://github.com/ir193/AMExtractor (Accessed 24 February 2017).
Android Backup Extractor, 2014. http://sourceforge.net/projects/adbextractor (Accessed 02 June 2017).
Android Debug Bridge, 2017. http://developer.android.com/tools/help/adb.html (Accessed 24 February 2017).
Android volatility, 2016. https://github.com/volatilityfoundation/volatility/wiki/Android (Accessed 12 September 2016).
Cellebrite UFED, 2017. http://www.cellebrite.com (Accessed 24 February 2017).
DDMS, 2017. https://developer.android.com/studio/profile/ddms.html (Accessed 24 February 2017).
Exploiting Android S-Boot, 2017. http://hexdetective.blogspot.nl/2017/02/exploiting-android-s-boot-getting.html (Accessed 21 August 2017).
External Bus Interface, 2015. https://en.wikipedia.org/wiki/External_Bus_Interface (Accessed 14 September 2015).
Facebook Messenger Secret Conversations, 2016. https://www.bustle.com/articles/188119-how-to-use-facebook-messenger-secret-conversations-encrypt-all-your-messages-easily (Accessed 06 October 2016).
File-Based Encryption, 2017. https://source.android.com/security/encryption/file-based.html (Accessed 24 February 2017).
Full-Disk Encryption, 2017. https://source.android.com/security/encryption/full-disk.html (Accessed 24 February 2017).
Grover, J., 2013. Android forensics: automated data collection and reporting from a mobile device. Digit. Investig. 10, S12–S20.
Guri, M., Poliak, Y., Shapira, B., Elovici, Y., 2015. JoKER: trusted detection of Kernel Rootkits in android devices via JTAG interface. IEEE Trust. 65–73.
Hannay, P., 2011. Kindle forensics: acquisition & analysis. Proc. Conf. Digit. Forensics Secur Law 6 (2), 143–150.
Hex-Rays, 2016. https://www.hex-rays.com/index.shtml (Accessed 24 February 2017).
High-Level Data Link Control, 2017. https://en.wikipedia.org/wiki/High-Level_Data_Link_Control (Accessed 07 March 2017).
Hoog, A., 2009. Android Forensics. Mobile Forensics World.
Hoog, A., 2011. Android Forensics: Investigation, Analysis and Mobile Security for Google Android. Syngress.
How to acquire memory from a running Linux system, 2017. https://gist.github.com/adulau/5094750 (Accessed 24 February 2017).
HPROF, 2017. https://developer.android.com/studio/profile/am-hprof.html (Accessed 24 February 2017).
Iqbal, A., Alobaidli, H., Baggili, I., Marrington, A., 2013. Amazon kindle fire HD forensics. In: Digital Forensics and Cyber Crime, pp. 39–50.
Lessard, J., Kessler, G., 2010. Android forensics: simplifying cell phone examinations. Small Scale Digit. Device Forensics J. 4 (1), 1–12.
LG Software and tools Download, 2017. https://www.mylgphones.com/lg-software-tools-download (Accessed 24 February 2017).
LiME, 2017. https://github.com/504ensicsLabs/LiME (Accessed 24 January 2017).
Mobile Malware, 2017. https://usa.kaspersky.com/internet-security-center/threats/mobile-malware#.WKTKYFUrKUk (Accessed 24 February 2017).
Mobile Phone Examiner plus, 2017. http://accessdata.com/solutions/digital-forensics/mpe (Accessed 24 February 2017).
MSAB XRY, 2017. https://www.msab.com (Accessed 24 February 2017).
Mutawa, N., Baggili, I., Marrington, A., 2012. Forensic analysis of social networking applications on mobile devices. Digit. Investig. 9, S24–S33.
ORT tool, 2017. http://www.orttool.com (Accessed 24 February 2017).
Oxygen Forensics, 2017. https://www.oxygen-forensic.com (Accessed 24 February 2017).
Pantech Self-Upgrade, 2017. http://www.pantechservice.co.kr/down/self/main.sky (Accessed 24 February 2017).
RIFF box, 2017. http://www.riffbox.org (Accessed 24 February 2017).
Rooting, 2016. http://en.wikipedia.org/wiki/Rooting_(Android_OS) (Accessed 31 December 2016).
Samsung Kies, 2017. http://www.samsung.com/us/support/owners/app/kies (Accessed 24 February 2017).
Samsung KNOX, 2017. http://www.samsungknox.com (Accessed 24 February 2017).
Samsung Odin, 2017. http://odindownload.com (Accessed 24 February 2017).
Sboot_dump, 2017. https://github.com/nitayart/sboot_dump (Accessed 21 August 2017).
Secure Boot, 2017. https://source.android.com/devices/tech/security/secureboot/index.html (Accessed 24 February 2017).
Smartphone OS Market Share, 2016. Q2. http://www.idc.com/promo/smartphone-market-share/os (Accessed 24 February 2017).
Son, N., Lee, Y., Kim, D., James, J., Lee, S., Lee, K., 2013. A study of user data integrity during acquisition of android devices. Digit. Investig. 10, S3–S11.
Sun, H., Sun, K., Wang, Y., Jing, J., Jajodia, S., 2014. TrustDump: reliable memory acquisition on smartphones. ESORICS 202–218.
Sylve, J., Case, A., Marziale, L., Richard, G., 2012. Acquisition and analysis of volatile memory from android devices. Digit. Investig. 8, 175–184.
Taubmann, B., Huber, M., Wessel, S., Heim, L., Reiser, H., Sigl, G., 2015. A lightweight framework for cold boot based forensics on mobile devices. In: International Conference on Availability, Reliability and Security, pp. 120–128.
Trace32, 2017. http://www.lauterbach.com/frames.html?home.html (Accessed 16 March 2017).
Vidas, T., Zhang, C., Christin, N., 2011. Toward a general collection methodology for Android devices. Digit. Investig. 8, S14–S24.
WeChat Message Cryptography, 2017. http://open.wechat.com/cgi-bin/newreadtemplate?t=overseas_open/docs/oa/encryption/overview (Accessed 24 February 2017).
WhatsApp Security, 2017. https://www.whatsapp.com/security/?l=en (Accessed 24 February 2017).
Willassen, S., 2005. Forensic Analysis of Mobile Phone Internal Memory. In Advances in Digital Forensics. Springer International Publishing, pp. 191–204.
World Smartwatch Market, 2016. https://www.idc.com/getdoc.jsp?containerId=prUS41736916 (Accessed 24 February 2017).
Yang, H., Zhuge, J., Liu, H., Liu, W., 2016. A Tool for Volatile Memory Acquisition from Android Devices. In Advances in Digital Forensics XII. Springer International Publishing, pp. 365–378.
Yang, S., Choi, J., Kim, K., Chang, T., 2015. New acquisition method based on firmware update protocols for android smartphones. Digit. Investig. 14, S68–S76.