Mobile Forensics: An analysis of Android architecture and evidence integrity

Alejandro Mera

IA5210 Information System Forensics

December 12, 2015

Abstract

The pervasive use of mobile devices have changed our lives. These devices are part of our daily activities in the office, school and home. Moreover, they contain information that describe our behavior and activities. This characteristic is of particular interest for forensic investigators, who can retrieve information of probative value from these devices. The following study analyses Android OS architecture to determine the caveats of different mobile forensics techniques. Findings reveal that mobile forensics techniques for Android devices are invasive procedures that depend on the specific hardware and software of each device. These techniques are effective only if the investigator understand the Android OS characteristics.

Introduction

Mobile devices and the pervasive access to internet are technologies that have changed our lives. Computer tablets, wearables gadgets and specially smartphones, support our daily activities in the office, home, college, or any place where personal information is generated, processed, stored and transmitted. In fact, studies reveal that almost two-thirds of North Americans are now smartphone users and at least 7% of them rely heavily on smartphones for online access (Pew Research Center 2015). This extensive use inevitably relates these devices to all sort of activities, including criminal or illegal ones. Voice and video records, pictures, geo-localization tags, SMS, and MMS are some examples of probative data contained in smartphones that, unlike personal computers, can describe in detail user's behavior and activities.

The market of smartphones is diverse and competitive, that is why vendors customize their products offering different form-factor, hardware specifications, connectivity technologies and applications. However, Operating System (OS) is a common factor that roughly classifies smartphones. Furthermore, studies reveal the Google's Android OS dominates the worldwide market with a 82.8% share in 2015 second quarter, while iOS and Windows Phone control 13.9% and 2.6% respectively. Although adoption of Android is wide and its compatibility standards establish a framework for manufacturers, fragmentation in both hardware and software is a phenomenon that is disturbing the Android environment. Consequently, researchers and investigators face challenging issues to develop a general forensic technique that preserves evidence's integrity and recovers artifacts from a large and heterogeneous collection of devices and applications.

The technology for storing information (NAND flash) or the inability to protect integrity of evidence—write-blockers—have determined that forensic procedures used with smartphones are essentially invasive. For instance, relevant research in this field revealed that rooting Android device (Al Barghouthy, Marrington, and Baggili 2013) or flashing bootable images (Vidas, Zhang, and Christin 2011) is required to retrieve comprehensive data, otherwise it is not possible to access specific artifacts. These procedures violates the main principle of a forensic investigation—preserve evidence's integrity—and can affect the security of the device. However, an adequate forensic methodology that documents every step of the investigation and considers particularities of each device ensure that acquired information from smartphones can be used in a trial as evidence.

Android environment and security design

The origin of Android involves not only the development of an OS, in fact "Android" in the modern mobile ecosystem "refers to a company, an operating system, an open source project, and a development community" (Drake et al. 2014). The Android's journey to become the most popular mobile OS began when Andy Rubin, Chris White, and Nick Sears founded Android Inc. in October 2003. However, the actual Android ecosystem was stablished when Google acquired this company and led the Open Handset Alliance (OHA) with the intent of conquering the mobile market. This alliance is a group of eighty-four technological and mobile companies, who claim a commitment to openness to accelerate innovation in technology and services of the mobile market (OHA 2011).

The openness of the Android platform—free and mostly open source—have attracted different Original Equipment manufacturers (OEM), who have used the Android OS not only to power smartphones or tablets, but wearables, automotive control panels, video game consoles, smart TVs, embedded control systems and domotic applications. This wide adoption makes the Android hardware basis extremely diverse (Drake et al. 2014). Moreover, it represents a huge challenge for researchers and forensics investigators because manufacturers and carriers build their own devices based off the platform, jeopardizing the hardware fragmentation of the Android ecosystem (Vidas, Zhang, and Christin 2011). For example, the same device purchased from different carriers include different software—proprietary software—that struggles with a general methodology for Android forensics analysis.

The updating model of Android is another issue that affects fragmentation. Google's OEM mobile devices (Nexus phones and tablets) are the first that have received new major versions

and patches of the Android OS directly from Google. However, other devices depend on the device manufacturer and the carrier to receive patches and updates, which in many cases are never deployed, leaving the device exposed to security vulnerabilities and trapped in older or buggy versions. Google regularly gathers information from the new Play Store application and publishes a relative number of devices running a given version of the Android OS. Figure 1 depicts data collected during a 7-day period ending on November 2; where the adoption of the latest version Marshmallow is little (0.3%) and only comparable with Froyo (0.2%) the oldest version still tracked.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.3%
4.1.x	Jelly Bean	16	11.0%
4.2.x		17	13.9%
4.3		18	4.1%
4.4	KitKat	19	37.8%
5.0	Lollipop	21	15.5%
5.1		22	10.1%
6.0	Marshmallow	23	0.3%

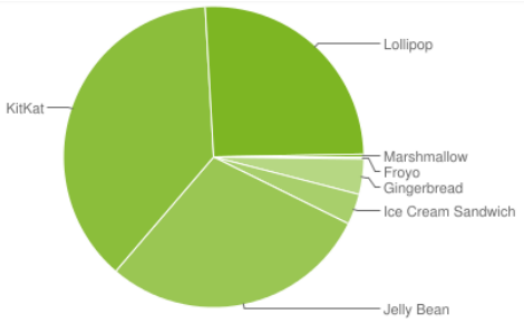

Figure 1 Google platform version. Data from Google's dashboards 2015

The Android system architecture and the security enforcement model

Android OS is a Linux distribution designed for the specific characteristics of mobile devices such as reduced power consumption, mobility, limited resources, and enhanced user experience. Figure 2 depicts the five layers of the Android software stack, which include applications (OEM/preinstalled and user installed), Android framework, Android runtime

(Dalvik/ART VM) and native libraries, hardware abstraction layer (HAL) and the Linux kernel. This architecture is hardware agnostic and gives developers the opportunity to extend devices' functionality maintaining the security schemes offered by APIs and libraries that run under the Java Dalvik Virtual machine.

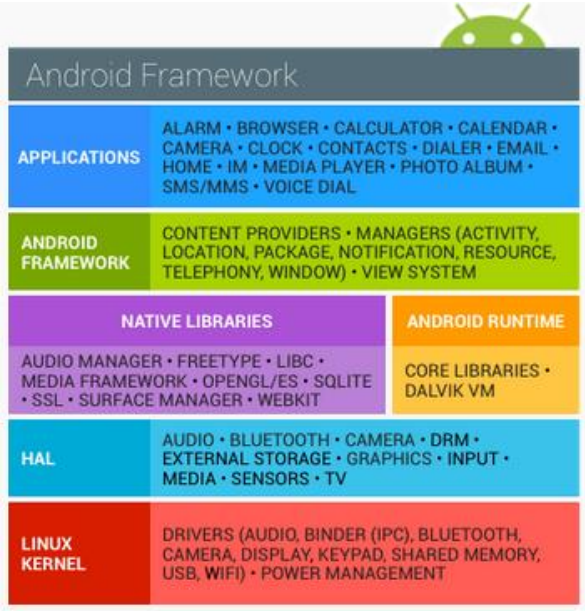

Figure 2 Android Software Stack. Google 2015

Although Android uses the Linux process isolation and principle of least privilege to implement application sandboxing, it does not have the "passwd" and "group" files to source the credentials used in the Linux's User ID / group ID paradigm (Drake et al. 2014). The specific Android's security model include a map of names to unique identifiers called Android identifiers (AID), which are statically assigned to critical-system users and dynamically provisioned as a unique Linux UID for each application at installation time. This model ensures that each application runs on its own sandbox (UID/GID) isolated from other process, while supplementary groups—a common functionality of other Linux distributions—grant access to shared resources such as SD cards.

Starting with Android 4.3 (Jelly Bean), Security-Enhanced Linux (SELinux) is used to further define the boundaries of the Android application sandbox (The Android Open Source Project 2015). SElinux implement a mandatory access control (MAC) over all processes, including process running with root/super-user privileges, which reduces the effects of malware and automates security policy creation. The MAC can operate in one of two modes: permissive mode and enforcing mode. In the first mode, permission denials are logged by the kernel; while in the second mode, permission denials are logged and enforced. The default-operating mode of SELinux is enforcing, therefore a per-domain permissive mode, a label-based scheme, authorizes process (domains) operating in permissive mode while the rest of the system remains in enforcing mode.

Unique user UIDs for each application, restricted file system permission, MAC and a Linux-like process isolation offer a secure sandboxing scheme for Android. However, the pervasive use of mobile devices requires a higher-level isolation that keeps personal and business information coexisting in the same device. Some vendors, which claim to solve this issue, aggregate a new level of isolation extending the Android sandboxing scheme or developing a corporative solution that include virtualized environments. For example, Samsung's Knox® extends isolation scheme using Trusted Boot and ARM TrustZone-based Integrity Measurement Architecture (TIMA) (Samsung 2015), while Vmware® Horizon mobile® offers a "dual persona solution" that creates a virtualized operating system on android smartphones. This virtualized environment is a corporate workspace separated from personal application and data on the device (VMware 2013).

The sandboxing scheme of Android is of particular interest for a forensic investigation because the isolation and data processes of each application stablish a heterogeneous and protected environment that preclude investigators' activities. In an Android device, each application can generate a diverse set of artifacts (session ID, geo-localization, metadata, usernames, password, SQL databases, SMS, call log, etc.) that are specific and different among applications. In fact, some researchers have evidenced that current mobile forensic tools are not able to produce comprehensive analysis of artifacts (Kasiaras et al. 2014) or even it is not possible to recover private web browsing traces (particular artifacts of Orweb) unless the forensic tool is running with root/super-user privilege (Al Barghouthy, Marrington, and Baggili 2013).

The Android partition layout

Android's partition layout, similarly to other Linux distributions or Operating Systems, defines the order, offsets and sizes of logical storage divisions of the device's nonvolatile memory (NAND Flash). Usually, these partitions are mapped to Memory Technology Devices (MTD), which are an abstraction layer that allows software to access different types of raw flash devices (Vidas, Zhang, and Christin 2011). The partition layout is another factor that affects hardware fragmentation. Moreover, vendors implement different partitions layouts among their devices to adapt specific hardware configurations and platforms. However, some of these partitions are common among all Androids devices, and share specific functionalities that are essential for the Android system. Table 1 describe the most common partitions defined in Android devices including the main functionality of each one.

Table 1 Android partition layout (Vidas, Zhang, and Christin 2011) (Drake et al. 2014)

Partition Name	File system	Mount point	Description
System	yaffs2	/system	Stores the system image, which contains the Android framework, libraries, system binaries, and pre-installed applications. This partition normally contain the Google apps (Gmail, Calendar, Maps and Play store)
Data	yaffs2	/data	This is the partition where user data is stored such as downloaded applications, pictures and video. This is considered an internal no removable storage, which differs from SD card.
Cache	yaffs2	/cache	It contains temporary files, logs, Dalvik VM cache and update packages.
Boot	booting	NA	A bootable partition includes the Linux Kernel and the root file system ram disk (initrd).
Recovery	booting	NA	A minimal bootable partition that includes vendor's specific tools to maintenance operations like wiping the cache or user data, installing patches, taking backups and transferring files.
Boot loader	booting	NA	This partition includes the more basic initialization routines that loads the Linux kernel or enables alternative boot modes such as recovery or download.

The partition layout represents one of the key factors analyzed in a forensic investigation. The characteristics that an investigator must know about each partition include type of stored information, access level (user and super-user/root), file system and functionality (boot, storage, recovery). For instance, some mobile forensic techniques take advantage of alternative boot modes (Vidas, Zhang, and Christin 2011), (Fan et al. 2015) to flash and use modified images that include forensics tools to recover data from Android devices. From a forensic perspective,

flashing a device is a no reversible and destructive process that will alter the integrity of the evidence. However, user data is not stored on those partitions, which allows recovering artifacts from other unaltered partitions.

Flashing an Android device is a normal process used by manufacturers and carriers to install customized Android versions or exclusive applications. This process uses specific software contained in the bootloader partition, which is capable of booting, flashing, and interacting with a PC program (fastboot/odin) over a USB. Manufacturers lock the bootloader before releasing their devices, but it can be unlocked under specific manufacturer's conditions and limitations—the process varies between manufacturers and devices. For example, Sony consents unlocking the bootloader after following a validation process (IMEI based) and accepting terms and conditions. Sony does not offer the unlocking process for all of its handsets models or even releases. Figure 3 depicts a verification screen for a Sony Xperia Z3 compact, where the unlocking process is allowed.

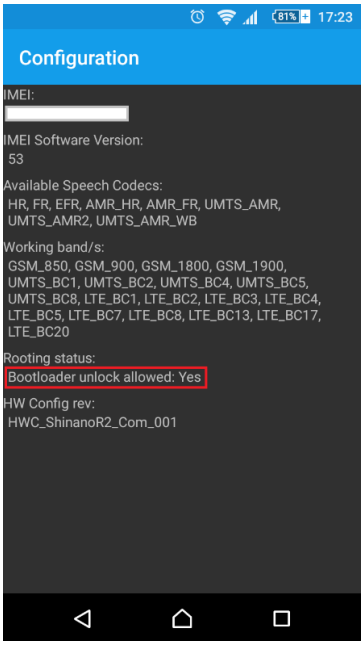

Figure 3 Bootloader unlock Xperia Z3 compact

Rooting an Android Device

The super-user, in a UNIX-like system, is an especial account that has permissions and rights over all the files and programs (Ward 2015). The name of this account is “root”, that is why the process of getting super-user privileges in an Android device is called rooting. Ideally, an Android device uses its sandboxing scheme to run each application with the specific privileges needed to perform its function. In some cases, this may include root privileges, but only preinstalled applications or system processes can run in this mode. Therefore, the main goals of rooting an Android device are granting elevated privileges to user-installed applications, accessing/modifying the file system and change standard behavior.

Although rooting frees device’s characteristics/powers—an effect particularly appreciated by developers and hackers—it severely compromises the security (Son et al. 2013) and stability (Wen and Yubin 2015) of Android devices. Moreover, an application running with elevated privileges potentially can break the Android’s sandbox scheme, expose personal data or even destroy the system. For example, security engineers working on Android Pay (Google’s mobile paying system) consider rooted devices insecure for a paying platform. They claim that only devices that pass the compatibility test suite—a rooting unfriendly test—ensure a well-understood security model for all involved parts in a paying platform (XDA developers 2015).

Rooting an Android device consists in having a “su” binary that include the access right “setuid” in the file system. The setuid right allow users to run binaries as though the file owner (Ward 2015). This is a common procedure of the Linux kernel to elevate privileges and perform administrative tasks; therefore, the access to setuid binaries is sensible. The rooting process normally include the installation of a super-user application, which administers requests for

elevating privileges. The most popular super-user applications are Superuser by ClockworkMod and SuperSU by Chainfire, these applications do not root the device by itself, but offer a convenient way to manage requests, create policies, log events and revoke privileges.

The techniques to get the “su” binary in the Android file system fall into three main categories:

- Bootloader-based technique flashes a custom system image that include the “su” binary in the file system. Since boot loader is locked by default, this procedure is highly dependent on the availability of an unlocked boot loader.
- Android recovery mode uses a custom update package with the “su” binary, and an especial bootable recovery image. This procedure boots the Android device using the especial recovery image, and install the update package. The main drawback of this procedure is the signature verification implemented in some devices, which prevents installation of unsigned packages or updates.
- Software-based techniques principally exploits unpatched vulnerabilities of the Android system to access a shell and install the “su” binary.

From all of the described techniques, software-based procedures are of particular interest for researchers and forensic analysts. This type of techniques are regularly used by commercial mobile forensic software to acquire data, and are becoming more difficult to develop while the Android OS get mature. In fact, finding a universal rooting vulnerability in the Android OS is an event that attracts the attention of researchers and media. For example, a universal root technique developed by Wen Xu and Yubin Fu and presented in the Blackhat 2015 conference in

Las Vegas, demonstrates the exploitation of the CVE-2015-3636 in the Linux. According with the researchers, this was the first documented universal root exploit for 64 bits Android devices.

Forensic techniques for data acquisition from Android devices

The techniques for data acquisition from Android devices traditionally recover information from the devices' persistent memory (NAND Flash). However, new research has demonstrated feasible methods to recover volatile data from physical memory (Sylve et al. 2012), (Fan et al. 2015). According to some studies, forensics techniques for mobile devices can be divided into logical acquisition and physical acquisition (Son et al. 2013), but this rough classification does not include the volatile data acquisition. Therefore, a more general classification of mobile forensic techniques include persistent data acquisition and volatile data acquisition.

Persistent data acquisition techniques

This category includes logical and physical acquisition. The techniques of logical acquisition are file exploring, Android Debug Bridge (ADB), partition imaging, recovery mode, live SD, and Firmware update protocol. On the other hand, techniques of physical acquisition are Join Test Action Group (JTAG) extraction, Chip-off and Micro-read (Ayers, Brothers, and Jansen 2014). The Following section will briefly describe these techniques highlighting evidence integrity concerns.

File exploring uses the Media Transfer Protocol (MTP) and Mass Storage Class device implemented in the USB (Universal Serial Bus) standard. These protocols allows transferring files between portable devices (Android device) and PCs over a USB connection. This type of

acquisition does not require installing especial applications or rooting the device, but only have access to multimedia user data (videos, pictures, and audio files).

The ADB is a versatile tool that communicates an Android device with a developer computer. This tool requires USB debugging right, which is enabled under the developer options of an Android device. ADB in an unrooted device has pretty much the same characteristics of MTP or Mass Storage Class device. However, ADB working with a rooted device can install forensic tools on the device, which running as super-user can image device's partitions. Commercial tools like Cellebrite usually implement this technique including third-part exploits to gain root access.

Recovery mode uses the bootloader to flash a forensic image in the recovery partition. This image boots the device, mounts the partitions in read-only mode and takes forensically sound images of each one. After finishing the acquisition, the forensic tool transmits images to a PC using ADB bootloader (Vidas, Zhang, and Christin 2011). Since recover-partition does not contain personal data, this technique preserve user data integrity but modifies a complete partition of the device (Al Barghouthy and Marrington 2014).

The Live SD method is similar to Recovery mode. However, this uses the concept of live DVD/CD/SD to boot the device using a forensic image flashed in the SD card. This method preserves the integrity of all device's partitions, but this is only compatible with Android devices that support SD memory expansion (Lohrum 2014). In addition, this method does not require the ADB communication, instead uses the SD card to store the acquired images.

Firmware update protocol uses reverse engineering to detect "dump" commands in the USB communication protocol. These commands are part of the proprietary protocol and are used

by manufacturers to update device's firmware (odin/fastboot) (Yang et al. 2015). Dump commands can retrieve a raw copy of the flash memory or individual partitions, but are not implemented in all devices or the functionality has been retired.

JTAG is a standard used by manufacturers for debugging and testing electronic equipment. This procedure requires physical access to device's printed circuit board and access to a JTAG port (Ayers, Brothers, and Jansen 2014). The time required to recover an image with this technique is greater than using other methods such as Live SD or Recovery mode. However, this procedure has been used to compare evidence integrity because of its direct access to flash memory chips (Oliver Buckley 2014).

Chip-off and Micro-read are the most invasive methods. These requires remove physically flash memory chips form the device. In the case of Chip-off technique, investigators uses manufacturing processes and especial tools (hot-air soldering stations) to remove the chips. Sometimes, electronic chips suffer thermal damage or electrostatic discharges that destroy evidence. On the other hand, micro read is a destructive process that analyzes the silicon substrate of each chip to find the logical status of each gate. This process is very expensive and is not commercially available.

Volatile data acquisition techniques

Accessing the physical memory (RAM) of a mobile device is not a well-studied field. One of the reasons is the novel technologies used to accommodate common technologies (desktop-based) in embedded devices. For example, Android OS implements the Dalvik/ART virtual machine to offer a java-like environment for developers. However, Dalvik/ART uses different paradigms—compile in advance—to improve performance and reduce power consumption.

These specific implementations require a complete new set of techniques that take care of mobile particularities. For example, researchers recovered artifacts from RAM dumps of Android devices modifying and developing new profiles for the Volatility framework. These developments take care of the ARM architecture—prevalent in mobile devices—and the Dalvik/ART VM (Sylve et al. 2012).

The first published technique to recover physical memory requires a rooted device and a crafted kernel module. Therefore, it involves elevating privileges on the Android device in order to load a forensic module (Lime) in the Linux Kernel. The kernel module copies the physical memory to a SD card or establish a TCP connection with the investigator's computer to recover the data. This technique modifies the device. However, researchers tried to maintain evidence integrity under a forensic perspective and less invasive procedures (Sylve et al. 2012).

The second known method takes advantage of the RAM chip technology used in mobile devices. This technology (DDRAM) is not very different from the technology implemented in desktop computers. For instance, the main differences between a mobile and a desktop memory RAM are power consumption and access speed. This similarity exposes the vulnerabilities for an effective cold boot attack (Halderman et al. 2008) and a post-mortem analysis. However, the particularities of the Android devices requires flashing the recovery partition to mount a forensic image that recover physical memory (Sylve et al. 2012).

Conclusions

Android is the dominant operating system in the current mobile market. The openness of this operating system and the support of Google—one of the biggest companies in the world—are the main reasons of this broad adoption. The massive use of Android devices can include legit

and illegal activities. Therefore, the study and research of Android devices is of particular interest for forensics investigators and the scientific community.

Fragmentation of both software and hardware is the main consequence of the openness of Android. This characteristic has driven the development of a heterogeneous environment, where a general method to acquire information in a forensically sounded way does not exist. Furthermore, the Android's sandboxing scheme—a Linux-like process isolation—protects user data, but preclude the execution of a forensic application that retrieve artifacts from different applications.

The scientific community has developed techniques that retrieve information from Android in a forensically sounded way. For example, techniques that use the recovery partition, live SD, and firmware update protocol maintain the integrity of the evidence or prat of it. However, these techniques are highly dependent on the device characteristics such as unlocked bootloader, SD memory expansion support, "dump" commands in the update protocol, and root access.

The analysis of a particular device is the key factor to perform a forensic acquisition. Although the heterogeneous Android environment is challenging, a well-trained investigator has to choose the correct technique to retrieve information and preserve evidence's integrity. Moreover, investigators have to understand each technique, even the underlying techniques of commercial software. This knowledge and correct documentation of the process guarantee that evidence retrieved from a mobile device can be used in trial.

Bibliography

Al Barghouthy, N. B., and A. Marrington. 2014. "A Comparison of Forensic Acquisition Techniques for Android Devices: A Case Study Investigation of Orweb Browsing Sessions." New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on, March 30 2014-April 2 2014.

Al Barghouthy, N., A. Marrington, and I. Baggili. 2013. "The forensic investigation of android private browsing sessions using orweb." Computer Science and Information Technology (CSIT), 2013 5th International Conference on, 27-28 March 2013.

Ayers, Rick, Sam Brothers, and Wayne Jansen. 2014. "Guidelines on Mobile Device Forensics." *NIST Special Publication 800-101 Rev. 1*.

Drake, Joshua J., Zach Lanier, Collin Mulliner, Pau Oliva, Stephen A. Ridley, and Georg Wicherski. 2014. "Android hacker's handbook." In. Indianapolis, IN: Wiley,.

Fan, Zhou, Yang Yitao, Ding Zhaokun, and Sun Guozi. 2015. "Dump and analysis of Android volatile memory on Wechat." Communications (ICC), 2015 IEEE International Conference on, 8-12 June 2015.

Halderman, J. Alex, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2008. Lest We Remember: Cold Boot Attacks on Encryption Keys. Princenton University.

Kasiaras, D., T. Zafeiropoulos, N. Clarke, and G. Kambourakis. 2014. "Android forensics: Correlation analysis." Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for, 8-10 Dec. 2014.

Lohrum, Mark. 2014. "Live imaging an Android device: Not as hard as it sounds if you break it down." Last Modified August 10. http://freeandroidforensics.blogspot.com/2014/08/introduction.html.

OHA. 2011. "Open Handset Alliance." OHA Accessed Nov. 28. http://www.openhandsetalliance.com/.

Oliver Buckley, Jason R. C. Nurse, Philip A. Legg, Michael Goldsmith, Sadie Creese. 2014. Reflecting on the Ability of Enterprise Security Policy to Address Accidental Insider Threat. Socio-Technical Aspects in Security and Trust (STAST), 2014 Workshop on.

Pew Research Center. 2015. "U.S. Smartphone Use in 2015." Accessed Nov. 26. http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/.

Samsung. 2015. "KNOX workspace." Accessed Dic. 1. https://www.samsungknox.com/en/products/knox-workspace.

Son, Namheun, Yunho Lee, Dohyun Kim, Joshua I. James, Sangjin Lee, and Kyungho Lee. 2013. "A study of user data integrity during acquisition of Android devices." *Digital Investigation* 10, Supplement:S3-S11. doi: http://dx.doi.org/10.1016/j.diin.2013.06.001.

Sylve, Joe, Andrew Case, Lodovico Marziale, and Golden G. Richard. 2012. "Acquisition and analysis of volatile memory from android devices." *Digital Investigation* 8 (3–4):175-184. doi: http://dx.doi.org/10.1016/j.diin.2011.10.003.

The Android Open Source Project. 2015. "Security." Google Accessed Nov. 30. https://source.android.com/security/index.html.

Vidas, Timothy, Chengye Zhang, and Nicolas Christin. 2011. "Toward a general collection methodology for Android devices." *Digital Investigation* 8, Supplement:S14-S24. doi: http://dx.doi.org/10.1016/j.diin.2011.05.003.

VMware. 2013. "Verizon Rolls Out Dual Persona Solution with VMware " Accessed Nov. 28. https://www.vmware.com/company/news/releases/vmw-horizon-Verizon-051513.

Ward, Brian. 2015. "How Linux works what every superuser should know, 2nd edition." In. San Francisco, CA: No Starch Press,. Text.

Wen, Xu, and Fu Yubin. 2015. "Own your Android! Yet Another Universal Root." *Black Hat 2015*.

XDA developers. 2015. "Google Security Engineer Explains Issues With Root and Android Pay." Accessed Nov. 23. http://www.xda-developers.com/google-security-engineer-explains-issues-with-root-and-android-pay-in-the-xda-forums/.

Yang, Seung Jei, Jung Ho Choi, Ki Bom Kim, and Taejoo Chang. 2015. "New acquisition method based on firmware update protocols for Android smartphones." *Digital Investigation* 14, Supplement 1:S68-S76. doi: http://dx.doi.org/10.1016/j.diin.2015.05.008.