Evaluating the privacy of Android mobile applications under forensic analysis

Christoforos Ntantogian, Dimitris Apostolopoulos, Giannis Marinakis, Christos Xenakis
Department of Digital Systems, University of Piraeus
Piraeus, Greece
{dadoyan, apostolopoulos, marinakis, xenakis}@unipi.gr

Abstract

In this paper, we investigate and evaluate through experimental analysis the possibility of recovering authentication credentials of mobile applications from the volatile memory of Android mobile devices. Throughout the carried experiments and analysis, we have, exclusively, used open-source and free forensic tools. Overall, the contribution of this paper is threefold. First, it thoroughly, examines thirteen (13) mobile applications, which represent four common application categories that elaborate sensitive users' data, whether it is possible to recover authentication credentials from the physical memory of mobile devices, following thirty (30) different scenarios. Second, it explores in the considered applications, if we can discover patterns and expressions that indicate the exact position of authentication credentials in a memory dump. Third, it reveals a set of critical observations regarding the privacy of Android mobile applications and devices.

Keywords: privacy of mobile applications, mobile forensics; android; memory dump; mobile applications; volatile memory; authentication credentials.

1 Introduction

According to recent reports [1], the global adoption of smart phones and tablets has been growing faster than any other consumer technology in history. These small factor devices introduce a new processing and communication paradigm, enabling end-users to access and manage a broad set of data and services, while on the move. To materialize this, a wide range of mobile applications have been developed, which are extending from entertainment and gaming to critical mobile banking and proprietary enterprise applications for accessing corporate resources.

Along with great opportunities, mobile devices reveal new attack vectors for the involved parties (i.e., users, service providers, data owners, etc.) [2]. It is a fact that mobile devices can be easily stolen or misplaced, due to their compact size. The

loss of a mobile device can lead to major privacy breach, since emails, social activities, pictures or any other stored data can be disclosed. A study in 2011, named as the lost smart phone problem [3], determined that in a 12-month period 142,708 out of 3,297,569 employee smart phones were lost or stolen, i.e., 4.3 percent per year. Moreover, in 2012, researchers from Symantec presented their results of the Smartphone Honey Stick Project [4]. In this project, 50 smart phones were, intentionally, lost in cities around the U.S. and Canada. The phones were loaded with logging software, so that Symantec could keep track of all activities. The study came to the result that in the 96 percent of the cases, the finders had accessed the personal data (e.g., email, photos, etc.) that was stored in the lost devices. Moreover, on nearly half of them (43 percent), the finders had attempted to access the owners' online banking applications.

The proliferation of mobile devices has also led to the birth of mobile digital forensics, a branch of digital forensics that deals with the recovery of digital evidence or data from mobile devices, under forensically sound conditions. The latter denotes the acquisition of identical copies of the entire available evidences/data, without causing any alteration to the underlying device. Currently, most of the forensic research on mobile devices has been focused on: (i) the acquisition and analysis of the internal flash NAND memory and SD Cards; (ii) the understanding of the employed file systems; and (iii) the scrutinizing of the application files for identifying malware. However, little attention has been paid to the research on the acquisition and analysis of the volatile memory, also referred as random access memory (RAM), of mobile devices. This is the motivation of the present work, which focuses, explicitly, on the volatile memory of mobile devices. Moreover, this type of memory holds, temporary, the authentication credentials (i.e., *usernames* and *passwords*) submitted by the users to activate security critical applications (e.g., mobile banking, password managers, etc.).

Previous research has proved that forensic investigators can discover critical information in the volatile memory of desktop computers, like users' authentication credentials [5]. Thus, it is motivating to examine if we can discover such information in the volatile memory of mobile devices. Considering that 61 percent of the Internet users reuse authentication credentials on multiple websites/services [6], we realize that sometimes the disclosure of a *username* and/or *password* is sufficient to compromise the privacy of all the user's applications [7]. Especially, in case of applications that deal with sensitive data or functionality (e.g., banking, password managers, e-shopping, etc.), an exposure of authentication credentials can lead to major privacy breach.

In this paper, we investigate and evaluate through experimental analysis whether we can discover authentication credentials of mobile applications in the volatile memory of rooted mobile devices, following thirty (30) different scenarios (i.e., eleven (11) general scenarios with some time variations). We focus on mobile devices that operate with the Android operating system (OS), because it is the most widely used one [8]. To perform the experiments, we follow a procedure for the acquisition of the volatile memory of rooted mobile devices, under forensically sound

conditions. Throughout the carried experiments and analysis, we have, exclusively, used open-source, free forensic tools. In total, we have evaluated the privacy of thirteen (13) popular Android applications, which represent four common application categories (i.e., mobile banking, e-shopping/financial applications, password managers, and encryption/data hiding applications) that elaborate sensitive users' data. For every investigated application and each studied scenario, we have performed two set of experiments with different objective each one. In the first one, our goal was to check if we could recover our own submitted credentials from the memory dump of a mobile device. In the second experiment, the goal was to find out patterns that indicate where the credentials are located in a memory image. Overall, the contributions of this paper are as follows:

(i) Examine for each investigated application and studied scenario whether we can discover authentication credentials in the physical memory of mobile devices;
(ii) Explore in the considered applications, if we can discover patterns and expressions that indicate the position of authentication credentials in a memory dump;
(iii) Derive a set of critical observations that provide insights for the privacy of mobile applications under various mobile usage scenarios.

The rest of the paper is organized as follows. Section 2 gives background information for Android OS and the related work. Section 3 presents the procedure for the acquisition of the volatile memory of Android mobile devices. Section 4 analyzes the carried out experiments. Section 5 elaborates on the results, providing generic observations and remarks regarding the privacy of authentication credentials in Android devices. Finally, section 6 concludes the paper.

2 Background

2.1 Android operating system

Android is a Linux-based OS designed, primarily, for touch screen mobile devices such as smart phones and tablet computers. Since its appearance, Android followed an upward trajectory and wide acceptance, reaching triple-digit of growth for the last year [8]. Today, it holds approximately 75 percent of the world market and there have been more than 48 billion of Android applications' installations so far, characterizing it as the fastest-growing mobile OS.

Android utilizes native open source C libraries to perform OS tasks and Java as a language for developing applications. To execute them, it uses the Dalvik virtual machine [9], which creates Dalvik executable files *.dex*, (i.e., byte codes from *.class* and *.jar* files), designed to be suitable for systems that are constrained in terms of memory and processor speed. Each Android application runs in a separate process within its own Dalvik instance, relinquishing all responsibility for memory and process management to the Android run time, which stops and kills processes as necessary to manage resources [10].

Android devices employ three different types of memory, each of which serves different purposes: (i) the volatile memory (i.e., RAM) that loses gradually its data when power is switched off; (ii) the internal, non-volatile memory that is based on NAND flash technology, which does not require power to retain data; and (iii) the external, expandable, non-volatile memory in the form of SD card. Both flash and SD card memory store the Android file system, named YAFFS2, as well as applications' and multimedia files.

In some security sensitive mobile applications (i.e., mobile banking, financial applications, password managers, etc.), the users' credentials (e.g., *username* and *password*) are never stored or cashed in the non-volatile memory (i.e., flash memory or SD card), trying to eliminate the possibility to be compromised and misused. Each time such an application is activated, the user is obliged to re-type and resubmit its credential to gain access to the provided services. The credentials, which are only resided within the volatile memory of mobile devices, are defined as *data in motion* [11]. In this work, we examine the potential of discovery authentication credentials, which exclusively exists as data in motion, evaluating the security and privacy that these mobile applications support.

2.2 Related work

Mobile devices constitute an important source of evidence for every forensic investigator, since they store a plethora of interesting information, such as locations, incoming and outgoing calls and messages, emails, browsing information, application usage, multimedia files, etc. Driving by this fact, the majority of the current forensic research on Android devices has been focused on the acquisition and analysis of the internal NAND flash memory and SD cards, presenting significant results regarding the YAFFS2 file system, as well as the data stored in it. Using also forensic techniques, researchers have achieved the investigation of Android application package (*.apk*) files, used to distribute and install applications on Android, for malware. However, to the best of our knowledge, little attention has been drawn to the acquisition and analysis of the volatile memory of Android devices.

In our previous work [12], we have employed the Dalvik Debugging Monitor Server (DDMS) tool [13] that comes with the Android software development kit to dump the memory contents of a running process. Using DDMS, we examined the memory snapshots of a sufficient number of mobile applications instances, and we discovered authentication credentials in the majority of them. A limitation of this work is related to the fact that the memory dumps were acquired from an Android emulator, rather than an actual Android device. Moreover, DDMS has limited capabilities for forensic analysis, since it cannot dump the entire memory of the device.

In [14], the authors have proved that cold boot attacks against Android mobile devices, equipped with ARM processors, are possible. To accomplish this, they used a Galaxy Nexus mobile device in which the bootloader had been unlocked and the disk partition of user data was encrypted. They were able to retrieve the employed key of

the encrypted partition of the disk from the volatile memory of the device, and, then, decrypt it. The authors also mentioned that on devices where the bootloader is locked, any attempt to break the disk encryption will result in the deletion of the data stored in it. However, they managed to retrieve personal information such as contact list, emails, photos, etc., from the device's volatile memory. Finally, the authors have developed a recovery image, named FROST, to automate the process of retrieving the employed encryption keys.

In [15], the authors have achieved to dump specific memory regions of a running process using the *ptrace* system call, which enables a process to inspect and control the execution of another process. The carried experiments and the consequent analysis were focusing on discovering evidences from chat applications (i.e., incoming and outgoing messages). A limitation of this approach has to do with the fact that each process of interest requires an exclusive memory extraction, which leads to more than one interaction with the mobile device that may cause overwrites and loss of evidences. Trying to address this, the authors of [16] have presented the first, public analysis of the Android's Dalvik virtual machine. This work includes the Dalvik's design as well as proposes a method of accessing it, leading to the recovery of forensically interesting information, such as history of calls, voicemails, browsing history, and wireless local area network keys.

Volatilitux [17] provides a framework (written in python programming language) for analyzing the volatile memory of a Linux-based system. It enables the extraction of digital artifacts from a memory dump, but it supports a limited set of analysis capabilities, such as enumeration of the running processes, memory map of the running processes, etc. Moreover, up to now, there is no any module for acquiring the volatile memory from Android devices. The work in [18] describes a technique for dumping the memory of an Android application using the *kill* command, which terminates a running process of an application. Analyzing the captured memory snapshot, the authors have succeeded to retrieve an encryption key used by the application under investigation. However, it can only be applied up to Android v2.1, since in the later versions this command has been removed.

The main limitations of the related work have to do with the following two facts: (i) it is acquired, only, a portion of the Android volatile memory; and (ii) the obtained memory snapshots are related to specific applications. Therefore, the carried out analyses as well as the related findings are confined to the above. In this paper, we manage to overcome these limitations by obtaining a full capture of the volatile memory of an Android device, which is forensically sound, and analyzing it, thoroughly. A forensically sound acquisition allows us to obtain copies of the volatile memory, which are identical to the physical memory of the mobile device; while the procedure itself does not alter the original device.

3 Volatile memory acquisition procedure

To dump the volatile memory of a rooted Android mobile device, we used an open-source forensics tool named, Linux memory extractor (LiME) software [19]. LiME is

a loadable kernel module, which allows the acquisition of the volatile memory from Linux and Linux-based devices, such as those powered by Android. LiME is able to acquire the memory pages in a forensically sound manner (approximately 99 percent of memory pages), since it minimizes the impact on the physical memory of the target device when transferring data to and from it. To achieve this, LiME has been designed with the following features:

(i) Only a single binary (i.e., the LiME module) needs to be transferred to the device and executed to perform the memory acquisition.
(ii) The LiME module has a minimal memory footprint, since it is very small (~70 KB)
(iii) LiME requires very few kernel functions to perform the memory dump.
(iv) LiME requires minimal interaction with userland, since data reading and writing is handled within the kernel. In this way, LiME avoids hundreds of system calls and other function invocations that would otherwise need to be performed and modify the volatile memory.

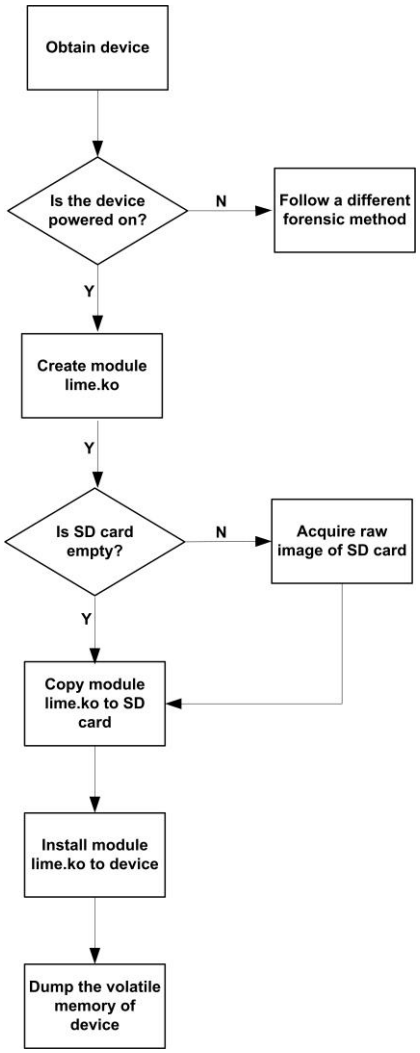

Figure 1: A flowchart of the procedure for the acquisition of the volatile memory of a rooted Android mobile device

Figure 1 depicts the procedure and the required steps that should be followed to acquire a forensically sound memory image from an Android mobile device. In this procedure, we assume that the Android device is rooted. If it is not rooted, we should attempt to gain root privileges on the device, since LiME requires root privileges to perform memory dump [26]. There are various reliable methods for Android OS that allow privilege escalation from normal user to root [20] [25] and we don't analyze them further. The most important part in the rooting procedure is to assure that the mobile device is not rebooted, since volatile memory cannot retain its data without power. Next, we create a loadable kernel module, named *lime.ko*, which is an object file that contains code to extend the running kernel functionality of an OS, such as Android. Technical details for the process of creating the *lime.ko* module can be found in [19]. After creating *lime.ko*, we acquire a forensically sound image of the device's SD memory card, using an open source imaging tool called *dd* [21]. The reason that we have to undertook this step is that the module *lime.ko* should be copied in the SD card. Since an important principle in forensic investigations is to avoid tampering the evidence, we have to acquire a raw image of the SD card, before copying the module *lime.ko* into the device's SD memory card. In this way, we ensure that our results are forensically sound.

In the sequel, we copy the module *lime.ko* into the SD memory card of the mobile device. After that, we, physically, connect the mobile device to a computer, using a universal serial bus (USB) interface. Next, as root, we execute in the mobile device the command *insmod* to insert the *lime.ko* module into the Android kernel. At the same time, the dumping process starts, where the memory dump is downloaded from the Android device to the computer through the Android Debug Bridge (*adb*) [22]. The time, required to end the dumping process, depends on the size of the volatile memory of the examined mobile device.

4 Experiments

In this section, we present and analyze the carried out experiments. In a three months period, we examined thirteen (13) Android applications in total, which elaborate sensitive users' data. The majority of the examined applications release updates frequently. It is worth mentioning that all experiments were performed with the latest version of the applications, until June 1st, 2013. Each one of the considered applications employs a *username* and/or *password* as *data in motion*. Based on the provided functionality, the underlying mobile applications are divided into four categories:

- mobile banking (m-banking) applications;
- e-shopping/financial applications (i.e., applications that facilitate the mobile users to perform online payments or buy goods);
- password managers (i.e., applications that manage in a secure manner the passwords of the user);

- encryption and data hiding applications (i.e., applications that aim at enhancing the user's privacy on its mobile device).

m-banking and e-shopping/financial applications employ *usernames* and *passwords* to authenticate the mobile users, enabling remote access to the provided services. On the other hand, password managers, encryption and data hiding applications use only *passwords* (or the concatenation of a password with a random string that increases entropy) as keys to encrypt/decrypt *passwords* of other applications or other user's sensitive data such as calls, messages, etc. All the tested applications provide logout or termination functionality.

The m-banking category includes six applications of six major banks in Greece (i.e., bank1, bank2, bank3, bank4, bank5, and bank6). The e-shopping/financial category comprises three applications (i.e., financial1, financial2, financial3) of well-known, international, e-commerce businesses that allow financial transactions. The password manager category consists of two applications (i.e., password1, password2) that store and retrieve *passwords* in a secure manner. They employ a *master password* to protect all the other mobile user's *passwords*. Finally, the fourth category incorporates two applications (i.e., encyrption1, encryption2). The encryption1 uses a secret *password* as a key to encrypt messages (i.e., SMSs) and emails, sent and received by the mobile user. The encryption2 application, on the other hand, hides calls and messages from a specified contact from the contact list of the mobile device, and the only way to depict them is the user, first, entering a *secret code* in the dialer pad and then pressing the call button.

Our test bed is equipped with a rooted Samsung Galaxy S Plus (i9001). This smart phone uses an Android v2.3 (Gingerbread), which is the most popular Android version (i.e., based on the relative number of devices that running this Android version), according to the Google's statistics [23]. We have studied thirty (30) different scenarios (see Table 1), which are based on eleven (11) general scenarios with some time variations (i.e., we acquired the volatile memory from the inspected device after different time intervals). In every variation of each scenario, we have examined the entire set of the considered applications, using our own created credentials to login. For all scenarios, the dumping process lasted nine minutes for each one, while the size of the memory dump was approximately 512 MB (it is equal to the size of the physical memory of Samsung Galaxy S plus). The investigated scenarios as well as their variations are, briefly, described below:

Scenario 1: *Login, use and logout from the examined applications. We begin the memory dumping: a) immediately after the logout; b) after waiting for 10 minutes; c) after waiting for 20 minutes; and d) after waiting for 60 minutes. During the waiting time period, we keep the mobile device idle (i.e., powered on without any use).*

Scenario 2: *Login, use and logout out from the examined applications. We begin the memory dumping after waiting, first, for a time interval of: a) 10 minutes, b) 20 minutes and c) 60 minutes. During the waiting time period, we use the mobile device*

only as a phone, which means that it sends and receives phone calls and short messages.

Scenario 3: *Login, use and logout out from the examined applications. We begin the memory dumping after waiting, first, for a time interval of: a) 10 minutes, b) 20 minutes and c) 60 minutes. During the waiting time period, we use the mobile device only as a smart phone, by activating various common Android applications such as Gmail, Play Store, YouTube, Mp3 player, News reader application, etc.*

Scenario 4: *Login and use the considered applications, but we do not logout from them. Instead, we set them to run into the background by pressing the home button of the mobile device. We begin the memory dumping: a) immediately after setting the application into the background; b) after waiting for 10 minutes; c) after waiting for 20 minutes; and d) after waiting for 60 minutes. During the waiting time period, we keep the mobile device idle (i.e., powered on without any use). This scenario was chosen because many users instead of logging out and properly closing the applications, they simply press the home button and return to the home screen.*

Scenario 5: *Login and use the considered applications, but we do not logout from them. Instead, we set them to run into the background by pressing the home button of the mobile device. We begin the memory dumping: a) after waiting for 10 minutes; b) after waiting for 20 minutes; and c) after waiting for 60 minutes. During the waiting time period, we use the mobile device only as a phone, which means that it sends and receives phone calls and short messages.*

Scenario 6: *Login and use the considered applications, but we do not logout from them. Instead, we set them to run into the background by pressing the home button of the mobile device. We begin the memory dumping: a) after waiting for 10 minutes; b) after waiting for 20 minutes; and c) after waiting for 60 minutes. During the waiting time period, we use the mobile device only as a smart phone, by activating various common Android applications such as Gmail, Play Store, YouTube, Mp3 player, News reader application, etc.*

Scenario 7: *Login, use and logout from the examined applications. After logging out, we employ a task killer to terminate all the running processes, and, then, we acquire the volatile memory of the device.*

Scenario 8: *Login, use and logout from the examined applications. After logging out, we switch the device to the airplane mode (i.e., deactivate all the communication interfaces, wireless, 3G, etc.). We begin the memory dumping: a) immediately after the switching; b) after waiting for 10 minutes; c) after waiting for 20 minutes; and d) after waiting for 60 minutes. During the waiting time period, we keep the mobile device idle (i.e., powered on without any use).*

Scenario 9: *Login, use and logout from the examined applications. After logging out, we switch the device to the airplane mode (i.e., deactivate all the communication interfaces, wireless, 3G, etc.). We begin the memory dumping: a) after waiting for 10 minutes; b) after waiting for 20 minutes; and c) after waiting for 60 minutes. During the waiting time period, we only use game applications.*

Scenario 10: *Login, use and logout from the examined applications. After logging out, we switch off the mobile device, and then, switch it on (i.e., rebooting). Immediately after rebooting, we acquire the device's volatile memory.*

Scenario 11: *Login, use and logout from the examined applications. After logging out, we switch off the mobile device and remove the battery for 5 seconds. Then, we install the removed battery, switch on the mobile device, and after the completion of booting, we acquire the device's volatile memory.*

Table 1: Summary of the experiment scenarios

Scenarios	Description of steps
Scenario 1	
S1.a	*Login, use, logout, immediate dump.*
S1.b	*Login, use, logout, device idle for 10 minutes, dump.*
S1.c	*Login, use, logout, device idle for 20 minutes, dump.*
S1.d	*Login, use, logout, device idle for 60 minutes, dump.*
Scenario 2	
S2.a	*Login, use, logout, use it as a phone for 10 minutes, dump.*
S2.b	*Login, use, logout, use it as a phone for 20 minutes, dump.*
S2.c	*Login, use, logout, use it as a phone for 60 minutes, dump.*
Scenario 3	
S3.a	*Login, use, logout, use it as a smart phone for 10 minutes, dump*
S3.b	*Login, use, logout, use it as a smart phone for 20 minutes, dump*
S3.c	*Login, use, logout, use it as a smart phone for 60 minutes, dump*
Scenario 4	
S4.a	*Login, use, set the application into the background, immediate dump.*
S4.b	*Login, use, set the application into the background, device idle for 10 minutes, dump.*
S4.c	*Login, use, set the application into the background, device idle for 20 minutes, dump.*
S4.d	*Login, use, set the application into the background, device idle for 60 minutes, dump.*
Scenario 5	
S5.a	*Login, use, set the application into the background, use the device as a phone for 10 minutes, dump.*
S5.b	*Login, use, set the application into the background, use the device as a phone for 20 minutes, dump.*
S5.c	*Login, use, set the application into the background, use the device as a phone for 60 minutes, dump.*
Scenario 6	
S6.a	*Login, use, set the application into the background, use the device as a smart phone for 10 minutes, dump.*
S6.b	*Login, use, set the application into the background, use the device as a smart phone for 20 minutes, dump.*
S6.c	*Login, use, set the application into the background, use the device as a smart phone*

	for 60 minutes, dump.
Scenario 7	
S7	*Login, use, logout, use task killer, immediate dump.*
Scenario 8	
S8.a	*Login, use, logout, switch the device to airplane mode, immediate dump.*
S8.b	*Login, use, logout, switch the device to airplane mode, device idle for 10 minutes, dump.*
S8.c	*Login, use, logout, switch the device to airplane mode, device idle for 20 minutes, dump.*
S8.d	*Login, use, logout, switch the device to airplane mode, device idle for 60 minutes, dump.*
Scenario 9	
S9.a	*Login, use, logout, switch the device to airplane mode, use gaming applications for 10 minutes, dump.*
S9.b	*Login, use, logout, switch the device to airplane mode, use gaming applications for 20 minutes, dump.*
S9.c	*Login, use, logout, switch the device to airplane mode, use gaming applications 60 minutes, dump.*
Scenario 10	
S10	*Login, use, logout, reboot, immediate dump.*
Scenario 11	
S11	*Login, use, logout, switch off the device, remove battery for 5 seconds, insert battery, switch on, dump.*

For each investigated application and studied scenario or scenario variation, we have carried out two experiments with different objective each one. In the first one, our goal was to check if we could recover our own submitted credentials from the memory dump of the mobile device. In the second experiment, on the other hand, the goal was to find out patterns and expressions that indicate where the credentials are located in a memory dump. This would be beneficial in forensic investigations, where the researchers have memory images of Android devices and they may use these patterns to find unknown credentials. On the contrary, as a negative side effect, if a malicious steals an Android device, it will try these patterns to find out the credentials of the device's owner.

To perform the experiments, we repeated the following steps for each examined application and studied scenario. First, we randomly choose and login to the application under investigation (already installed in our test bed device), by submitting our own created credentials (i.e., *username* and/or *password*). After using the application for an arbitrary time period, which varies from 2 to 10 minutes, we carry out the specific actions, described in each considered scenario. Then, we dump the device's volatile memory, using the procedure described in section 3. Finally, we search for the submitted credentials and nearby patterns, by employing another open-source forensics tool called The Sleuth Kit (TSK) [24]. TSK is used to perform forensic investigations and data extraction from images of Windows, Linux and Unix computers. It includes various utilities to find metadata entries, display data blocks within a file system, and search for allocated and unallocated file names within a file system.

5 Results

In the first set of experiments, we successfully recovered our own submitted credentials in the majority of the applications, since they were in plaintext, without almost any modification. In some cases, the characters of the retrieved credentials within the memory images were separated by the dot symbol. For example, in case that the submitted password of an application was the phrase *"password"*, then we located in the memory image the phrase *"p.a.s.s.w.o.r.d."*. The reason of this trivial modification was due to the employed Unicode encoding (i.e., UTF-16). We also observed in one application that the characters of the password string were HTML encoded. For example, in case that the password of the application was the string *"p@ssword!"*, then we found out in the memory image the string *"p%40ssword%21"*. In the following Figure 2, a memory snapshot using the TSK open-source forensic tool is presented, which includes the discovered password string *".d.s.s.e.c."* in clear text, where its characters are separated by the dot symbol.

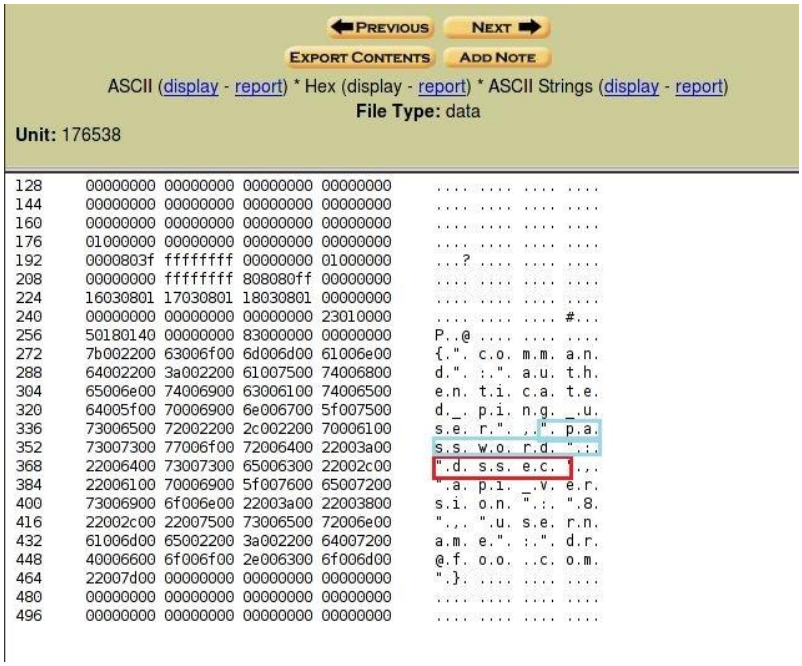

Figure 2: A snapshot of a memory dump presenting the discovered password *"dssec"* as well as the pattern *"password:"* that indicates the exact password location.

The following Figure 3 presents all the findings and summarizes the results of the first set of experiments (i.e., recovery of the own submitted credentials) for each investigated application (also grouped in categories) and studied scenario. In this figure, the cells that include the letters U and P indicate the successful recovery of the username and password, respectively, for the specific application and the considered scenario. Moreover, the grey colored cells that contain the letter X signify the unsuccessful recovery of a username or password. Finally, the cells that comprise the dash symbol denote that the related applications do not use a username (i.e., they only employ a password).

The analysis of the numerical findings in Figure 3 reveals some interesting observations regarding both the privacy level that the examined application support as

well as the behavior of the volatile memory of Android mobile devices, under different usage conditions. More specifically, by studying the results per experimentation scenario, we may deduce that in scenarios 1 and 4 we were able to discover the majority of the submitted authentication credentials (i.e., 80 percent in both of them). This can be attributed by the fact that in both scenarios, the mobile device was idle before performing the memory dump, which leads to the observation1.

Observation1: *As long as the user does not employ the mobile device (i.e., powered on and idle), it is more likely the authentication credentials (i.e., data in motion) to remain intact in the volatile memory of the device.*

On the other hand, in scenarios 10 and 11 the authentication credentials had always been erased from the volatile memory and we couldn't find any of them (i.e., 0 percent in both of them), driving to the observation2.

Observation2: *The best way to ensure that the volatile memory of a mobile device does not contain any authentication credential (or other sensitive data) is either to reboot the device or remove its battery. This has been also proved for desktop/laptop computers [5]. However, there is a fundamental difference in the usage of mobile devices and desktop/laptop computers that makes this observation very critical. That is, users of desktop/laptop computers reboot or shut them down in a daily basis. On the other hand, the users of mobile devices, rarely, close or reboot them. In fact, mobile users try to avoid closing or rebooting their devices, as much as possible, in order not to miss any phone call. Therefore, we can deduce that it is more likely a malicious to discover authentication credentials in the volatile memory of mobile devices than desktop/laptop computers.*

An interesting observation (i.e., observation3) that derives from all the scenarios with time variation (e.g., scenario 1, 2, 3, 4, 5, 6, 8, 9) has to do with the fact that the longer we waited to dump the memory image, the less authentication credentials were discovered in it.

Observation3: *Time is with security. The more time passes from the moment a user submitted his/her authentication credentials, the more likely is the authentication credential to be deleted.*

In scenarios 4, 5 and 6, where the examined applications were set up to the background (i.e., instead of logging out properly), we were able to discover the authentication credentials in a percentage of 80%, 74% and 72%, respectively. Moreover, in scenario 7 we were able to discover the authentication credentials in a percentage of 72%, despite the fact that the investigated applications were ended using a task killer application. From these facts we derive the following observations:

Observation4: *Setting up a running application into the background does not delete the authentications credentials from the volatile memory of the mobile device. This is*

an alarming result, since it is a common practice among users to set up the running applications into the background, instead of logging out properly.

Observation5: *Using a task killer application to end a running application does not wipe out the related authentication credentials from the volatile memory.*

By comparing the results in scenarios 2 and 3, we notice that in the first one we were able to find out the authentication credentials in a percentage of 77%; while in the second, only, in 48%. Based on this we may infer the observation6:

Observation6: *Using a mobile device as a smart phone (i.e., activating various Android applications), it is more likely to erase the authentication credentials that reside in the device's volatile memory, than using it as mobile phone (i.e., make/receive calls and send/receive SMS). This happens because a running Android application overwrites, previously, stored data in the device's volatile memory. On the other hand, actions such as phone calls or sending/receiving SMS, do not engage the volatile memory of the mobile device, and, therefore, the contents of this memory are preserved.*

In scenarios 8 and 9 (i.e., switch the device to the airplane mode), we were able to discover the authentication credentials in a percentage of 58% and 16%, respectively. This significant difference, between these two scenarios, can be attributed by the fact that in scenario 9, after switching the device to the airplane mode, we launched and played a game application; while in scenario 8 the device stayed idle. Consequently, we may reason the following observation7.

Observation7: *Switching the mobile device to the airplane mode, the contents of the devices volatile memory are not necessarily erased, and, thus, authentication credentials can be recovered. However, in cases that after switching, the mobile user activates and runs an application such as a game, we notice that the majority of the authentications credentials, which reside at the volatile memory of the device, are erased.*

From the applications' point of view, we perceive that for the entire set of the tested applications, we were able to find out the authentication credentials, at least once. In particular, in the category of m-banking and financial/e-shopping applications, we discovered the submitted authentication credentials in a percentage of 65% and 51%, respectively. Similarly, in the group of password managers as well as the applications of encryption/hiding, the recovered credentials have reached the percentage of 45% and 71%, respectively. Based on these findings, we may deduce the following observations:

Observation8: *The majority of the examined Android applications are vulnerable to the recovery of authentication credentials from the volatile memory.*

Observation9: *It is alarming that even applications that should take security as a first priority, such as m-banking applications, have been proved to be vulnerable to the discovery of authentication credentials.*

		Applications																																Total	Total per scenario
		m-banking												financial/e-shopping										password managers				encryption/hiding							
		bank1		bank2		bank3		bank4		bank5		bank6		financial1		financial2		financial3		password1		password2		encryption1		encryption2									
Scenario 1	s1.a	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	P	-	P	-	P					20/22	71/88 80%
	s1.b	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	X	-	P	-	P	-	P	-	P					19/22	
	s1.c	U	P	U	P	U	P	U	P	U	P	X	X	U	X	U	P	U	X	-	P	-	P	-	P	-	P					18/22	
	s1.d	U	P	U	P	U	P	U	P	U	P	X	X	U	X	X	X	X	X	-	P	-	P	-	X	-	P					14/22	
Scenario 2	s2.a	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	X	-	P	-	P	-	P	-	P					19/22	51/66 77%
	s2.b	U	P	U	P	U	P	U	P	U	P	X	X	U	X	U	P	U	X	-	P	-	P	-	P	-	P					18/22	
	s2.c	U	P	U	P	U	P	U	P	U	P	X	X	U	X	X	X	X	X	-	P	-	P	-	X	-	P					14/22	
Scenario 3	s3.a	X	X	U	P	U	P	U	P	U	P	X	X	U	X	U	X	U	X	-	X	-	X	-	P	-	P					13/22	32/66 48%
	s3.b	X	X	U	P	U	X	U	P	U	P	X	X	U	X	U	X	U	X	-	X	-	X	-	P	-	P					12/22	
	s3.c	X	X	X	X	U	X	X	X	U	P	X	X	U	X	U	X	U	X	-	X	-	X	-	X	-	P					7/22	
Scenario 4	s4.a	U	P	U	P	U	P	U	P	U	P	U	P	U	P	U	P	U	P	-	P	-	P	-	P	-	P					22/22	71/88 80%
	s4.b	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	X	-	P	-	P					19/22	
	s4.c	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	X	-	X	-	P	-	P					19/22	
	s4.d	U	P	U	P	U	X	X	X	U	P	X	X	U	X	U	X	X	X	-	P	-	X	-	X	-	P					11/22	
Scenario 5	s5.a	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	X	-	P	-	P					19/22	49/66 74%
	s5.b	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	X	-	P	-	P					19/22	
	s5.c	U	P	U	P	U	X	X	X	U	P	X	X	U	X	U	X	X	X	-	P	-	X	-	X	-	P					11/22	
Scenario 6	s6.a	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	X	-	P	-	P					19/22	48/66 72%
	s6.b	U	P	U	P	U	P	U	P	U	P	X	X	U	P	U	P	U	P	-	P	-	X	-	P	-	P					19/22	
	s6.c	U	P	U	P	U	X	X	X	U	P	X	X	U	X	X	X	X	X	-	P	-	X	-	X	-	P					10/22	
Scenario 7	s7	U	P	U	P	U	P	U	P	U	P	X	X	X	X	U	P	X	X	-	P	-	P	-	P	-	P					16/22	16/22 72%
Scenario 8	s8.a	U	P	U	P	U	P	U	P	U	P	X	X	U	P	X	X	U	X	-	X	-	X	-	X	-	P					14/22	51/88 58%
	s8.b	U	P	U	P	U	P	U	P	U	P	X	X	U	X	X	X	U	X	-	X	-	X	-	X	-	P					13/22	
	s8.c	U	P	U	P	U	P	U	P	U	P	X	X	U	X	X	X	U	X	-	X	-	X	-	X	-	P					13/22	
	s8.d	U	P	U	P	U	P	U	P	U	P	X	X	U	X	X	X	X	X	-	X	-	X	-	X	-	P					11/22	
Scenario 9	s9.a	X	X	X	X	X	X	X	X	U	P	X	X	U	X	X	X	U	X	-	X	-	X	-	X	-	P					5/22	11/66 16%
	s9.b	X	X	X	X	X	X	X	X	U	P	X	X	X	X	X	X	X	X	-	X	-	X	-	X	-	P					3/22	
	s9.c	X	X	X	X	X	X	X	X	U	P	X	X	X	X	X	X	X	X	-	X	-	X	-	X	-	P					3/22	
Scenario 10	s10	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-	X	-	X	-	X	-	X					0/22	0/22 0%
Scenario 11	s11	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-	X	-	X	-	X	-	X					0/22	0/22 0%
Total		22/30	22/30	24/30	24/30	25/30	20/30	21/30	21/30	28/30	28/30	1/30	1/30	24/30	11/30	18/30	13/30	19/30	8/30	-	18/30	-	9/30	-	15/30	-	28/30						
Total per category		237/360 - 65%												93/180 - 51%						27/60 - 45%				43/60 - 71%									

Figure 3: Results of the first set of experiments (i.e., recovery of the own submitted credentials) for each investigated application, grouped in categories, and studied scenario.

From the m-banking applications, the most vulnerable was the application of bank5, since we recovered the authentication credentials in almost all scenarios (i.e., except for scenarios 10 and 11). On the other hand, the most secure application was this of bank6, in which we discovered the authentication credentials only in scenario 4.a, (i.e., set the running application to the background and, immediately, dump the volatile memory of the device). In the financial/e-shopping applications, the percentage of recovery of the submitted usernames was higher than this of passwords. The password managers were also vulnerable to the discovery of authentication credentials, but this happened more frequently in the application of password2, than in password1. Finally, the application of encryption1 has been proved more robust than the application encryption2, which was vulnerable to almost all the studied scenarios, except for scenarios 10 and 11. These findings lead to the following observations.

Observation10: *We found out that there are some Android applications that are secure under the threat of discovery of authentication credentials (e.g., bank6 application); while there are some other that are, completely, exposed to this (e.g., encryption2 and bank5 applications). This contradictory results show that some applications have been developed taking into account security precaution, whilst some other not.*

Observation11*: Regardless of the criticality of the considered applications, all developers should use correct and secure programing techniques (i.e., delete the authentication credentials when they are not used from the applications), in order to enhance the level of security provided by mobile platforms.*

Observation12*: Password managers that aim to enhance the privacy of users, by protecting their passwords, were found to be vulnerable. This means that if a user loses his/her device, a malicious may discover all the user's passwords, only by discovering the master password of the employed password manager application.*

Table 2. Discovered Patterns for usernames and passwords

Username	Password
j_username=	*j_password=*
username=	*password=*
userid>	*password:*
login i:type=	*pass i:type:*

In the second set of experiments, we determined specific patterns and expressions that indicate the location of the credentials (i.e., *username* and *password*) within the captured memory images, for the entire set of the examined applications. For example, as shown in Figure 2, right before the submitted password (i.e., dssec), we meet the string *"password:"*, evidently, indicating that the following string is the user's password. Some other patterns that we found out for passwords (in the considered applications) are "*password=*", *"j_password="*, and *"pass i:type"* (see Table 2). Therefore, the expression "*password"* or the excerpt "*pass"* indicate the

physical location, where the submitted passwords are stored, in clear text, for the entire set of the examined applications.

Similarly, the discovered expressions for the location of the submitted usernames in the captured memory images are: *"j_username=", "username=", "userid>",* and *"login i:type="* (see Table 2). Hence, the patterns *username, userid* or *login* signify the location where the usernames are stored in the captured memory images. A forensic investigator (or a malicious) can simply dump the volatile memory of an Android device and search for these patterns to discover the *usernames* and *passwords* of the owner of the device. The identification of such expressions and patterns infers the last observation13.

Observation 13: *We proved the existence of patterns and expressions that pinpoint where the authentication credentials of each application are, exactly, located in a memory dump. Therefore, a malicious may, easily, recover the authentication credentials from a stolen device, simply, by searching in the memory dump for these patterns or expressions. In contrast, the involved developers should avoid using such patterns or expressions in the provided mobile applications.*

6 Conclusions

In this paper, we investigated and evaluated the privacy of Android mobile applications. In particular, we examined whether authentication credentials in the volatile memory of Android mobile devices can be discovered, using open source forensics tools. The analysis of the results revealed that the majority of the considered Android applications are vulnerable to the recovery of authentication credentials from the volatile memory. It is alarming that even applications that should take security as a first priority, such as m-banking applications, have been proved to be vulnerable. Moreover, we observed that the volatile memory did not contain any authentication credential only when we rebooted the device or removed its battery. We also proved the existence of patterns and expressions that pinpoint where the authentication credentials of application are, exactly, located in a memory dump. Finally, taking into account that users tend to reuse password across various websites and applications, we drew the conclusion that regardless of the criticality of the applications, all developers should use correct and secure programing techniques and guidelines (i.e., delete the authentication credentials when they are not used from the applications), in order to hinder the authentication credential discovery and enhance the level of privacy, provided by mobile platforms.

References

[1] http://blog.flurry.com/bid/88867/iOS-and-Android-Adoption-Explodes-Internationally [Accessed on May 2013].

[2] A. Mylonas, A. Kastania, D. Gritzalis, "Delegate the smartphone user? Security awareness in the smartphone platforms," Computer & Security, Elsevier Science, Vol. 34, No. 1, pp. 47 – 66, May 2013.

[3] Ponemon Institute LLC. “The Lost Smartphone Problem: Benchmark study of U.S. organizations”, In Ponemon Institute Research Report, sponsored by McAfee, Oct. 2011.
[4] Wright, S. The Symantec Smartphone Honey Stick Project. Symantec Corporation, Mar. 2012.
[5] S. Karayianni, V. Katos, and C. K. Georgiadis, “A framework for password harvesting from volatile memory”, International Journal of Electronic Security and Digital Forensics, Vol. 4, No. 2-3, pp.154–163, 2012.
[6] Consumer Survey, Password habits, September 2012.
[7] A. Mylonas, M. Theoharidou, D. Gritzalis, “Assessing privacy risks in Android: A user-centric approach", in Proc. of the 1st International Workshop on Risk Assessment and Risk-driven Testing (RISK-2013), Springer, Turkey, November 2013.
[8] IDC worldwide quarterly mobile phone tracker, May 2013.
[9] D. Bornstein, “Dalvik VM Internals”, Google I/O Developer Conference, June 2008.
[10] http://mobworld.wordpress.com/2010/07/05/memory-management-in-Android/ [Accessed on Nov. 2012].
[11] A. Hoog, “Android Forensics: Investigation, Analysis, and Mobile Security for Google Android”, Syngress, Elsevier, June 2011.
[12] D. Apostolopoulos, G. Marinakis, C. Ntantogian, C. Xenakis, “Discovering authentication credentials in volatile memory of Android mobile devices”, the 12th IFIP Conference on e-Business, e-Services, e-Society, (I3E 2013), Athens, Greece April 2013.
[13] http://developer.Android.com/tools/debugging/ddms.html [Accessed on Nov. 2012].
[14] T. Müller, M. Spreitzenbarth, “Frost Forensic Recovery of Scrambled Telephones”, 11th International Conference on Applied Cryptography and Network Security (ACNS 2013), Alberta, Canada, June 2013.
[15] V. Thing, K-Y Ng, E.-C. Chang, “Live memory forensics of mobile phones”, 10th annual conference of digital forensic research workshop (DFRW) 2010.
[16] A. Case, “Memory Analysis of the Dalvik (Android) Virtual Machine”, SOURCE Seattle Dec. 2011.
[17] E. Girault, “Volatilitux : Physical memory analysis of Linux systems”, Dec. 2010.
[18] http://thomascannon.net/projects/android-reversing [Accessed on Oct. 2013].
[19] http://code.google.com/p/lime-forensics [Retrieved on Nov. 2012]
[20] D. Abbott, Linux for embedded and real-time applications-3rd edition, Chapter 7, December 2012
[21] http://www.forensicswiki.org/wiki/dd [Accessed on Oct. 2013].
[22] http://developer.android.com/tools/help/adb.html [Accessed on Nov. 2012].
[23] http://developer.android.com/about/dashboards/index.html [Accessed on 01/05/2013]
[24] http://www.sleuthkit.org/sleuthkit/index.php [Accessed on Oct. 2013].
[25] S. Kramer, http://c-skills.blogspot.com/2010/08/droid2.html, [Accessed on Oct. 2013].

[26] J. Sylve, A. Case, L. Marziale, G. G. Richard, “Acquisition and analysis of volatile memory from android device”, Digital Investigation, Elsevier, Vol. 8, Issues 3-4, pp: 175-184, Feb 2012.