Discovering Authentication Credentials in Volatile Memory of Android Mobile Devices

Dimitris Apostolopoulos, Giannis Marinakis, Christoforos Ntantogian, and Christos Xenakis

Department of Digital Systems, University of Piraeus
Piraeus, Greece
{mte1102,mte1116,dadoyan,xenakis}@unipi.gr

Abstract. This paper investigates whether authentication credentials in the volatile memory of Android mobile devices can be discovered using freely available tools. The experiments that we carried out for each application included two different sets: In the first set, our goal was to check if we could recover our own submitted credentials from the memory dump of the mobile device. In the second set of experiments, the goal was to find patterns that can indicate where the credentials are located in a memory dump of an Android device. The results revealed that the majority of the Android applications are vulnerable to credentials discovery even in case of applications that their security is critical, such as web banking and password manager applications.

Keywords: Android, Android applications, mobile security, volatile memory acquisition, credentials discovery.

1 Introduction

We live in a transitional period from desktop computing to ubiquitous computing. Modern mobile devices, such as smartphones and tablets, offer the capability of any time - any place computing. More than enough processing power, at least 512MB RAM, multicolor readable screens, GPS, Wi-Fi and 3G/4G internet access are some of the characteristics of today's mobile devices, putting virtually all the power of a desktop computer in a pocket. These capabilities allow people to carry with them the full contents of the Web along their whole digital life.

The proliferation of mobile devices has led to the birth of mobile digital forensics, a branch of digital forensics that deals with the recovery of digital evidence or data from a mobile device under forensically sound conditions. Most of the mobile devices' forensics research has been focused on areas like the acquisition and analysis of the internal flash NAND memory, SD Cards, understanding the file system and scrutinizing application files for malware analysis. However, little research has been done in the acquisition and analysis of the volatile memory (also referred as RAM) of mobile devices. This is the main reason that our work focus explicitly on the volatile memory of mobile devices.

C. Douligeris et al. (Eds.): I3E 2013, IFIP AICT 399, pp. 178–185, 2013.

The authors in [12] have demonstrated that forensic investigators can discover critical information in the volatile memory of desktop computers like users' authentication credentials. Therefore, an interesting question that arises is whether forensic investigators can discover sensitive data in the volatile memory of mobile devices. However, an important difference between desktop computers and mobile devices is that unlike desktop computers, the applications processes in mobile devices are not properly closed and continue to run in the background. This happens because the applications in order to be closed, the user must manually force to stop them. Thus, the volatile memory of mobile devices may contain a wealth of information which can be used as evidence in a court.

The previous statement is also interesting from a security point of view. 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 and pictures can be disclosed using readily available tools. Symantec's lost cell phone study [1], reveals that people who found a smartphone, violated the loser's privacy a whopping 89% of the time. On nearly half of those phones (43%), the finder attempted to access the owner's online banking application. Considering also the fact that 61% of internet users reuse passwords on multiple websites/services [3], sometimes the disclosure of one password is enough to compromise the privacy of all the user's applications. Thus, from a security point of view, it is motivating to examine if a malicious can discover authentication credentials in the volatile memory of mobile devices and breach the privacy of the device's owner.

Driven by the above observations, this paper investigates whether authentication credentials in the volatile memory of mobile devices can be discovered using freely available tools. We focus on mobile devices that operate with the Android Operating System (OS), since it is one of the most widely used mobile OS [2, 4]. The experiments that we carried out for each application included two different sets: In the first set, our goal was to check if we could recover our own submitted credentials from the memory dump of the mobile device. In the second set of experiments, the goal was to find patterns that can indicate where the credentials are located in a memory dump of an Android device. The results revealed that the majority of the Android applications are vulnerable to credentials discovery even in case of applications that their security is critical, such as web banking and password manager applications.

The rest of this paper is organized as follows. Section 2 provides background information and the related work. Section 3 presents the carried out experiments by analyzing the methodology we followed and elaborating on the obtained results. Finally, section 4 concludes the paper.

2 Background

2.1 Android Mobile Operating System

Android is a Linux-based OS designed primarily for touchscreen mobile devices, such as smartphones and tablet computers. Android uses native open source C libraries to perform OS tasks and uses Java as a language for developing Android applications.

To run applications, Android uses its own Virtual Machine called Dalvik [5]. The latter creates Dalvik executable files (.dex), which are byte codes from .class and .jar files. The compact .dex format is 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 [6].

Android devices have different types of memory used for different purposes. They have a volatile physical memory (i.e., RAM) that loses gradually its data when power is switched off. On the other hand, the internal storage memory of Android devices, which is based on NAND flash technology, is a type of non-volatile storage that does not require power to retain data. Moreover, SD cards store the file system of Android OS named YAFFS2, application files and multimedia files.

In many cases, Android applications can display and process data which are never saved or cached in non-volatile memory storage. For example, in banking and financial applications, passwords are not stored and the user of the device must type and submit his/her authentication credentials each time the application is used. Therefore, the credentials are only transmitted inside the device but not stored. This type of data is defined as data in motion [7].

2.2 Related Work

Research projects on acquiring and examining the volatile memory of Android devices (or in general of mobile devices) are very limited. In [11], the authors focus on specific running processes, and use the *ptrace* functionality of the kernel to dump specific memory regions of a process. However, the analysis of the virtual memory captures is limited in the sense that they are used only to discover evidences related to communication based applications (i.e., incoming and outcoming messages of chat applications). Moreover, a project named *Volatilitux* [8] provides only limited analysis capabilities, including enumeration of running processes, memory maps and open files, and does not provide a method to acquire memory from Android devices.

The most recent and concrete work on this area is found in [13]. The authors present a methodology for acquiring complete memory captures from Android by cross compiling a kernel and loading it into a rooted Android device. They also released a practical tool named *DMD* (now named *Lime Forensics*) [9] to demonstrate the feasibility and forensically soundness of their methodology. However, the authors did not analyze further the memory dumps to discover sensitive data.

Moreover, in [14] a technique is described for dumping the memory of an Android application by sending to the application the command *kill*. Next, analyzing the memory dump the author succeeded to discover an encryption key that used to decrypt SQLite databases containing sensitive data such as contacts, configuration settings, keys, etc. However, this method can be applied only to Android versions until 2.1, because from Android 2.2 and above the command *kill* has been removed.

Finally, the authors in [12] have described a memory acquisition process for desktop computers in order to recover passwords from volatile memory, even when the

computer is switched off. They examined some popular web application such as Facebook, Skype, and Gmail. In 56% of their tests they were able to recover passwords from the memory of the computer 15 minutes (in one case even 60 minutes) after the user disconnected and closed the computer. Moreover, they concluded that the time which someone can recover data depends on the type of RAM, but also by external factors such as the temperature of the room where the computer resides. Contrary to [12], our work focuses on Android devices, where the memory acquisition process presents significant challenges compared to the desktop computer counterparts [13].

3 Experiments

3.1 Methodology

To dump the volatile memory of Android devices we used an open source tool named the Dalvik Debug Monitor Server (DDMS) [10], which is a GUI based debugging application that allows the examination of running processes. Although its primary goal is to help developers to identify bugs in Android applications, we have successfully managed to use DDMS as a tool to dump the volatile memory of a running process.

In a six months period we examined thirty Android applications. The majority of these applications release updates frequently. We mention that all experiments were performed in the latest version of the applications until October 1st, 2012. Based on their functionality, we divide the applications into four categories. The first category consists of m-banking applications, the second category includes e-shopping/financial applications, the third category includes social networks and communication applications, and finally, the fourth category consists of password manager and encryption applications. The applications of the first three categories simply use a username and password to authenticate a user. On the other hand, the applications that belong to the fourth category (i.e., password managers and encryption applications) use the password of the user (or the concatenation of the password with a random string to increase entropy) as a key to encrypt/decrypt passwords of other applications or any other sensitive data.

The experiments that we carried out for each Android application included two different sets: In the first set, our goal was to check if we could recover our own submitted credentials from the memory dump of the mobile device. In the second set of experiments, the goal was to find patterns that will indicate where the credentials are located in a memory dump of an Android device. This would be beneficial in forensics, where the investigator has a memory image of an Android device and he/she may use these patterns to find unknown credentials. On the other hand, as a negative side effect a malicious can stole an Android device and try to disclose the passwords of the owner of the device.

A problem that we had to address was the fact that the majority of mobile devices are shipped with a custom version of the Android OS designed from the mobile device manufacturers. Therefore, the memory dumps can be different for each Android device. To cope with this issue, we decided to experiment in an Android emulator that

uses pure versions of the Android OS. However, to verify our findings from the emulator we have also experimented in a few actual Android devices (i.e., Sony Ericsson Xperia x8, Samsung Galaxy S / S Plus, Motorola Xoom Tablet).

In the first set of experiments we performed the following steps: First, we installed the applications under investigation for the various versions of the Android OS. Next, we chose and open randomly an application and we submitted our own credentials (i.e., username and/or password) to log in. Then we set the application process to run in the background. This was done because most users instead of logging out and closing the application, they press the home button and return to the home screen. After this, we captured a snapshot of the volatile memory used by the specific application. To achieve this we used the DDMS tool that lists all the running processes. We selected the application process and we created a memory dump that includes all the data in motion used by the specific application. Finally, we searched for the users credentials, using the command-line tool *grep* combined with regular expressions to specify various possible combinations of the user's credentials.

The second set of experiments starts where the first set ends. In particular, after we located the credentials inside the memory dump, we tried to identify patterns which are nearby (in terms of bytes offset) to the identified credentials.

3.2 Results

Since we did not observe great discrepancies between the emulator and the actual devices, we do not present the results separately. 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 any modification. The only modification that we observed in some applications was that the characters of the username and password strings had a dot symbol between them. For example, if the password of an application was submitted as "dssec", then we located the password as "d.s.s.e.c." (see Fig. 1). The reason of this trivial modification was due to Unicode encoding (i.e., UTF-16).

Regarding numerical results, in the first category (i.e., m-banking), we tested five mobile banking applications of five major Greek banks. We succeeded to retrieve the passwords in all five applications and the usernames in four of them. In the second category (i.e., e-shopping/financial), we tested three applications. We managed to retrieve the passwords in two applications and the usernames in all three applications. Regarding the third category (social networks and communications), we tested five applications. We successfully retrieved the username and password from all of them. Finally, in the fourth category (i.e., password managers and encryption) we tested seventeen applications. In this category, we managed to retrieve the passwords for all the applications. We summarize the results of our experiments in Table 1. In total, we succeeded to recover the passwords from twenty nine out of thirty examined applications and twelve usernames out thirteen applications that use usernames.

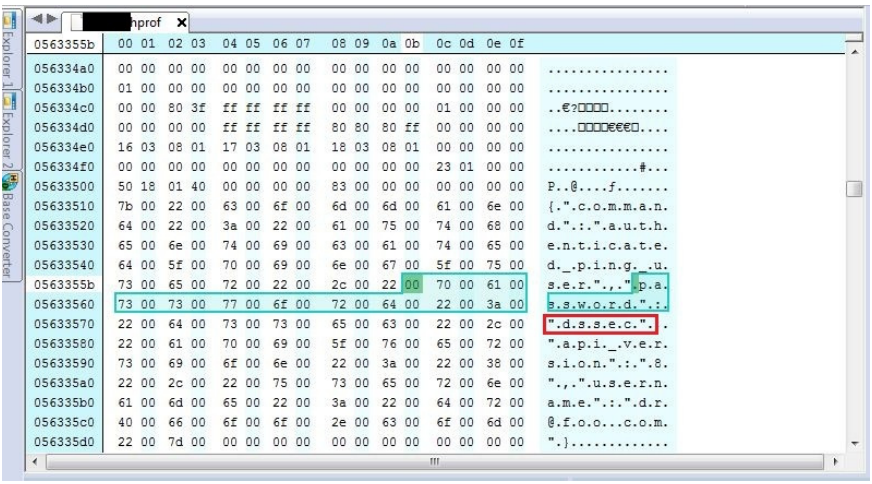

Fig. 1. Locating the password "dssec" and the pattern "password:" of an Android application in the memory dump file

In the second set of experiments, in all the applications we found a specific pattern that indicated the location of the password. For example, as shown in Fig. 1, right before the password that we had submitted (i.e., dssec), we found the string "password:", evidently indicating that the following string will be a password. Some other patterns that we found were "username:", "login name:" and "@Paw". Therefore, a forensic investigator (or a malicious) can simply dump the volatile memory of an Android device and search for these patterns to discover the passwords of the owner of the device.

Table 1. Numerical results for each application category

	Usernames	Passwords
m-banking	4/5	5/5
e-shopping/financial	3/3	2/3
Social networks/communications	5/5	5/5
Password managers/encryption	-	17/17
Total	12/13	29/30

From the analysis of the previous results we can deduce that the majority of the Android applications are vulnerable to credentials discovery. It is alarming that even web banking applications proved to be vulnerable, leaving their costumers unprotected against password disclosure. Moreover, password managers that encrypt passwords of other applications were all vulnerable, since we found the master password in the volatile memory that is used to encrypt all other passwords. Thus, in case a user loses his/her Android device, his/her whole internet activity may be in jeopardy. Given the fact that many users use the same password for different applications, one password discovery is enough to breach the security of all the applications.

4 Conclusions

This paper investigated whether authentication credentials in the volatile memory of Android mobile devices can be discovered using freely available tools. This would be beneficial in forensics, where the investigator has a memory image of an Android device and he/she may use these patterns to find unknown credentials. On the other hand, as a negative side effect a malicious can stole an Android device and try to disclose the passwords of the owner of the device. The experiments that we carried out for each application included two different sets: In the first set, our goal was to check if we could recover our own submitted credentials from the memory dump of the mobile device. In the second set of experiments, the goal was to find patterns that will indicate where the credentials are located in a memory dump of an Android device. Results showed that revealed that the majority of the Android applications are vulnerable to credentials discovery. In particular, we succeeded to recover the passwords from twenty nine out of thirty examined applications and twelve usernames out 13 applications that use usernames.

As a future work, we will extend the experiments to cover scenarios where the goal is to discover authentication credentials in cases where the mobile device is set to sleep mode, when it is closed or even when the battery is removed. Moreover, in all these cases, we will investigate how much time the contents of the memory are preserved before they are completely faded out.

Acknowledgments. This work was sponsored in part by the project "Biotautotita" (GSRT 09SYN-72-597) of the framework "Synergasia", funded by the General Secretariat for Research and Technology Development Department (GSRT).

References

1. http://www.foxnews.com/tech/2012/03/12/symantecs-lost-cell-phone-study-confirms-worst-in-people/ (retrieved on November 2012)
2. https://www.google.com/nexus/4/#play (retrieved on November 2012)
3. Study of Consumer Password Habits (September 2012), http://www.csid.com/news/csid-conducts-study-of-consumer-password-habits-finds-disconnect-in-practices-and-mindset/ (retrieved on November 2012)
4. http://www.idc.com/getdoc.jsp?containerId=prUS23771812 (retrieved on November 2012)
5. Bornstein, D.: Dalvik VM Internals. In: Google I/O Developer Conference (June 2008)
6. http://mobworld.wordpress.com/2010/07/05/memory-management-in-Android/ (retrieved on November 2012)
7. Hoog, A.: Android Forensics: Investigation, Analysis, and Mobile Security for Google Android. Syngress, Elsevier (June 2011)
8. Girault, E.: Volatilitux: Physical memory analysis of Linux systems (December 2010)
9. http://code.google.com/p/lime-forensics/ (retrieved on November 2012)

10. http://developer.Android.com/tools/debugging/ddms.html (retrieved on November 2012)
11. Vrizlynn, T., Ng, K.Y., Chang, E.-C.: Live memory forensics of mobile phones. In: Digital Forensic Research Workshop (2010)
12. Karayianni, S., Katos, V., Georgiadis, C.K.: A framework for password harvesting from volatile memory. International Journal of Electronic Security and Digital Forensics 4(2-3), 154–163 (2012)
13. Sylvea, J., Caseb, A., Marzialeb, L., Richard, G.: Acquisition and analysis of volatile memory from Android devices. Digital Investigation 8(3-4), 175–184 (2012)
14. http://thomascannon.net/projects/android-reversing/ (retrieved on November 2012)