A Forensic Investigation of Android Mobile Applications

Theodoula-Ioanna Kitsaki
Department of Digital Systems
University of Piraeus
theokitsaki@ssl-unipi.gr

Anna Angelogianni
Department of Digital Systems
University of Piraeus
annaaggelogianni@ssl-unipi.gr

Christoforos Ntantogian
Department of Digital Systems
University of Piraeus
dadoyan@unipi.gr

Christos Xenakis
Department of Digital Systems
University of Piraeus
xenakis@unipi.gr

ABSTRACT

This paper performs a forensic investigation to a set of Android mobile applications aiming at discovering sensitive information related to the owner of the mobile device. These applications were chosen based on the fact that: i) they are very popular on Google Play Store, ii) they handle sensitive personal information, iii) they have not been researched by previous works and iv) they are free to download and install. The three chosen applications belong to the following categories: bank, mobile network carrier and public transport. The evaluation of the security of the applications was performed using two techniques: code and disk analysis, as followed in the literature. Based on our findings we derive the conclusion that these applications despite their criticality have failed to incorporate security techniques to protect user's sensitive data and a forensic analysis can reveal crucial and significant information from a forensics point of view.

CCS CONCEPTS

• Applied Computing → Computer Forensics → Evidence Collection, Storage and Analysis

KEYWORDS

Forensics, Android, Mobile applications, Privacy.

1 Introduction

The increased use of mobile devices thus, applications, has created the need for application security. People use their mobile phones to perform most tasks of their everyday life from calling a taxi to booking a flight. According to 2017 statistics the total number of applications in Google and Apple Store were 2.8 and 2.2 million respectively [1] while in the first quarter of 2018 the number of available applications was 3.8 and 2 million respectively [2]. Moreover, according to 2017 statistics, the time that users spent on a desktop in oppose to the time they spent on devices such as smartphones and tablets is significantly less in the age group 18-64 [3].

The need to develop secure applications that protect user's information without omitting a service is imperative. In addition, the new legislation GDPR has been enforced in order to protect user's privacy, which has caused a supplemental need for security. The latest Android OS version offers API's and guidelines for developers with the intent to promote the adoption of secure practices, while the majority of new devices include an isolated hardware/software system named trusted execution environment (TEE) [4] to secure data at rest even in case the mobile device is rooted. Thus, it seems motivating to examine whether mobile applications securely store sensitive information or allow the discovery of valuable evidence in a forensic investigation.

Driven by the above observation, this paper performs a forensic investigation to a set of Android mobile applications aiming at discovering sensitive information related to the owner of the mobile device. These applications were chosen based on the fact that: i) they are very popular on Google Play Store, ii) they handle sensitive personal information, iii) they have not been researched by previous works and iv) they are free to download and install. The three chosen applications belong to the following categories: bank, mobile network carrier and public transport. The evaluation of the security of the applications was performed using two techniques: code and disk analysis, as followed in previous researches [5]. Based on our findings we derive the conclusion that these applications despite their criticality have failed to incorporate security techniques to protect user's sensitive data and a forensic analysis can reveal crucial and significant information from a forensics point of view.

The rest of the paper is organized as follows. In section 2 we present the existing literature on application forensics focusing on Android OS. In section 3 we present the experiments and the methodology used to analyze the chosen applications. In section 4 we present and elaborate on our findings, while in section 5 we draw the conclusions.

2 Related Work

Previous studies have analyzed a wide range of android applications such as social networking, instant messaging, banking, navigation and dating applications. More specifically, Chanajitt et al [6] focused on 7 e-banking mobile applications in Thailand. The forensic analysis conducted, focused on both memory and code inspection with the intent to determine different forms of leakage. This research discovered that the examined

applications posed a great threat on user's sensitive data such as account number, account type, account balance, citizen ID, date of birth, transactions from one bank to another or user's PIN code. Those threats could have been prevented if there was security by design.

Hayes et al [7] studied the geolocation information collected by Uber application. This research tested several scenarios in which Uber or competitor services applications were used by clients in New York City. The experiments proved that Uber was using geolocation information for more time than what was stated in the privacy agreement. It also accessed geolocation information even in cases where a competing service was used. These findings obviously violate user's privacy but could prove very useful in a criminal investigation.

Zhang et al [8] examined the case of instant messaging applications on Android phones such as Facebook Messenger, Line, Hangouts and WhatsApp. Their approach involved memory inspection of the device used in order to determine what kind of data was saved. The outcome of this work is that unencrypted messages and chat details can be retrieved from a rooted device even in the case of end to end messaging offered by WhatsApp which could provide forensic investigators with valuable information. Only in the case of Facebook Messenger secret chat the messages are self-destructed from the database according to the timer.

Another study on messaging applications is from Anglano [9] et. al., which focuses on the Telegram application for Android OS. The presented methodology that could be used for the forensic analysis of different applications and combines both memory and source code analysis. For the experiments, Android Mobile Device Emulator as well as a real mobile device was employed. This research discussed methods to interpret the data stored in the main database and in the user configuration file, to determine the specific Telegram account used on the local device, and to reconstruct the contact list of the user, the chronology and contents of exchanged textual and non-textual messages, as well as to determine whether the user created or administered a secret chat, a (super)group, or a channel. It also proved that the correlation between this information could provide an additional aid to a forensic investigation

Mata et al [10] chose to perform a forensic analysis on Feeld dating application. The investigation of Feeld application was focused on memory analysis using open source tools such as Autopsy. The findings contained personal data such as gender, sexual orientation or longitude and latitude which could victimize the user.

Zhang et al [11] investigated the popular WeChat application for instant messaging. The data acquisition method used was dependent on the version of the application and the device, while the decryption of the examined files was related solely to the version of the application. In all cases this researched managed to extract the exchanged messages despite the encryption techniques employed by WeChat.

Lone et al [12] focused on WhatsApp and Viber Android application used for messaging or calls. The methodology included memory analysis in both rooted and unrooted device mainly by inspecting the database. In the unrooted device, most information was encrypted while in the rooted device most of data such as phone numbers, exchanged messages, media files, location, SQL queries, deleted messages, calls made, duration of calls or profile pictures were found in plaintext view.

Furthermore, Anglano et al [13] studied the case of ChatSecure in Android platforms used for instant messaging. The methodology followed focused on both memory forensics and source code analysis. Android device emulators were used for the experiments to achieve reproducibility. It was discovered that ChatSecure stored the messages in the database encrypted. This research managed to decrypt the messages by studying the source code of the application. This study also managed to extract the passphrase from the volatile memory and propose how to correlate the data found in the database. It was also found that deleted data could not be restored.

3 Forensic analysis and Results

The choice of the 3 examined Android mobile applications was not random. In fact, the selected applications are very popular on Google Play Store and they handle sensitive personal information, while to the best of our knowledge there are no previous works that have analyzed them. The three chosen applications belong to the following categories: bank, mobile network carrier and public transport. It is important to notice that the total number of applications that were examined were 15 and we selected to present 3 specific applications, which yielded useful information from a forensics point of view. All examined applications were obtained from Google play store during June 2018.

Our methodology for the forensic analysis could be separated into two categories: code and disk analysis. The term code analysis refers here to the analysis of the source code to identify how the application manages and stores sensitive data. On the other hand, disk analysis refers to the examination of files that an application creates and manages with the aim to discover sensitive data into these files. In the following sections, we will present the methodology used to examine the applications and the forensically interesting results. Note that the examined applications will not be named; instead we use generic names to refer to them (i.e., Application A, Application B, etc.) for anonymity purposes.

3.1 Code Analysis

The code analysis was performed by obtaining first the .apk file of the application and then extracting its Java source code using *Apktool* [14], *dex2jar* [15] and *jd-gui* [16] tools. In particular, *Apktool* was used to decode the apk files to their original form (see Figure 1).

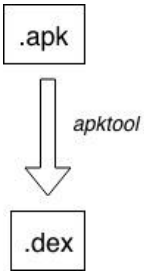

Figure 1: Apktool workflow

Moreover, *dex2jar* tool was used to convert the .apk file to .jar which combined with the *jd-gui* utility, allowed us to have a graphical interface to view and analyze the .class files within the .jar file (see Figure 2).

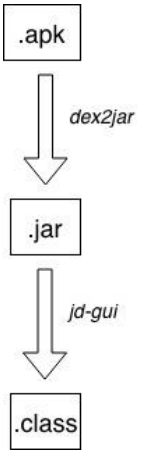

Figure 2: Dex2jar and jd-gui workflow

3.1.1 e-Banking application

The application A is an e-banking application and aims to facilitate user's transaction and inform them about new banking products. Using *Apktool* we were able to review the manifest.xml file that includes the permission of the application. We observed that the application does not use biometrics for authentication,

```
String initVector = "RandomInitVector";
boolean isCorp = false;
String key = "Bar12345Bar12345";
TransparentProgressDialog prDialog;
Boolean ticked = Boolean.valueOf(false);
TextView txtDescFb;
TextView txtDontHaveAnAccount;
TextView txtForgotPassword;
TextView txtTerms;
```

```
public static String encryptCipher(String paramString1, String paramString2, String paramString3)
{
  try
  {
    IvParameterSpec localIvParameterSpec = new IvParameterSpec(paramString2.getBytes("UTF-8"));
    SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramString1.getBytes("UTF-8"), "AES");
    Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    localCipher.init(1, localSecretKeySpec, localIvParameterSpec);
    byte[] arrayOfByte = localCipher.doFinal(paramString3.getBytes());
    System.out.println("encrypted string: " + Base64.encodeToString(arrayOfByte, 0));
    String str = Base64.encodeToString(arrayOfByte, 0);
    return str;
  }
```

Figure 3: encryptCipher method implementation

```
PinLogin.encryptCipher(PinLogin.this.key, PinLogin.this.initVector, localGson.toJson(PinLogin.this.gd.accountsDataList)));
```

Figure 4: Use of encryptCipher method in LoginUsername.class and PINLogin.class

Table: autofill

	name	value	value_lower	date_created	date_last_used	count
	Filter	Filter	Filter	Filter	Filter	Filter
1	user.lastname	Ioanna	ioanna	1522437739	1522437925	6
2	user.firstname	Ioanna	ioanna	1522437739	1522437925	6
3	user.msisdn	6[redacted]7	6[redacted]7	1522437739	1522437925	6
4	user.username	Tzo	tzo	1522437739	1522437739	1
5	user.username	Joanna	joanna	1522437834	1522437834	1
6	user.email	f[redacted]f@gmail.com	f[redacted]f@gmai...	1522437834	1522437925	5
7	user.username	Joanna7	joanna7	1522437859	1522437865	2
8	user.username	Joanna71	joanna71	1522437881	1522437881	1
9	user.username	71Joanna	71joanna	1522437925	1522437925	1
10	username	71Joanna	71joanna	1522438126	1522438149	2
11	passwordtext	#[redacted]+	#[redacted]...	1522438126	1522438126	1
12	passwordtext	7V297ZR	7v297zr	1522438149	1522438149	1
13	card.pan	HHAG-57GV-YC48-5848	hhag-57gv-yc...	1530101970	1530101973	2
14	card.holderName	Nikos	nikos	1530101970	1530101973	2
15	card.cvv	259	259	1529914126	1529914126	1

Figure 5: Structure of Web Data file

contrary to several other applications of this category which use fingerprint or face recognition. By analyzing the source code, we identified that instead of biometrics, application A supports the use of PIN as a faster and easier alternative to username and passwords.

Next, we examined how the PIN of the user is stored into the device. In particular, through code inspection, we found two interesting classes named "LoginUsername" and "PINLogin". Interestingly, the files contained a parameter named "key" and "initVector" which were statically set to "Bar12345Bar12345" and "RandomInitVector" respectively. As their name implies, the parameter "key" is used for the encryption key while "initVector" is the initialization vector of the encryption algorithm. These two parameters were used in the "encryptCipher" method as input in "SecretKeySpec" and "IvParameterSpec" classes without adding any level of randomness. Next, these two classes are used in the "encryptCipher" method to initialize the Java Cipher class, in order to encrypt the PIN of the user using AES in CBC mode (see Figure 3 and 4) and store it in the device. Evidently, since the key and the initialization vector are static, it leads us to the conclusion that the produced ciphertext is deterministic [17] and can be easily decrypted to obtain the user's PIN. As a side note, the latest version released in September 2018, the vulnerabilities described have been fixed.

3.2 Disk Analysis

Disk analysis was performed using a real mobile device. For the inspection of the applications we used a rooted Samsung Galaxy J5. In addition, *Root Browser* Android application [18] was used in order to access rooted device files and *DB Browser* for *SQLite* [19] investigate the files.

3.2.1 Mobile Network Carrier application

Application B is created and owned by a mobile carrier company, which is available for free on Google Play Store. The application aims to facilitate company's prepaid subscribers in their daily usage such as monitor their balance and expenses or receive exclusive offers.

A copy to a computer was made in order to analyze the abovementioned files. The employed Tools were *Droid Explorer* [20] which were used to preview and copy the application files to a computer for analysis. In order to use application B, registration

Figure 6: Data found in \cache folder (the IP address has been blurred)

is required. To put theory into practice, we registered to the application by providing a name and a surname in the registration form. Furthermore, a mobile number, a username and an email address were given. Last but not the least, the application required a credit card in order for users to pay a bill or renew a service. We provided a random data for the credit card (e.g., random card number, random owner, etc.).

The data of this application are saved in *\data\data\operator's_name* folder, which contains several subfolders and is accessed only by root user. An important subfolder is the *\app_webview* which includes files that store valuable information for forensic investigation. *Web Data* file is probably the most important file that was found during investigation of this Android application. It is a file acting as database, with many tables. From all the contents inside, *autofill* table is subject of significant interest as plenty of findings were discovered inside. The table comprises from data that is populated from user inputs to the application. Name, surname, mobile number, username, password, and most importantly all the information of our entered credit card (e.g., number, owner, etc.) was found unencrypted. Also, there are columns storing dates both for their creation and the last time they were used (see Figure 5).

3.2.2 Public transport application

The third application (i.e., Application C) that we examined provides information for public transportation informing users about scheduled bus routes, routes and locations of buses in real time, as well as bus stops. As in the previous application, we performed a disk analysis to discover forensic data.

For Application C, application's data were stored in *\data\data\com.att.android.tfa* folder, which includes the *\cache*, *\code_cache*, *\databases*, *\files*, *\no_backup*, *\shaders* and *\shared_prefs* folders. It is very interesting the fact that the *\databases* folder, the tfa.db database contained information about bus routes and timetables. Some important tables should be also referred. That is, *SearchHistory* table contained user's search history for buses and in the *UVersions* table, the version of each table and the last time the application was updated were saved. Finally, *Favorites* table contained the routes that the user saved as favorites.

In addition, the *\cache* folder included the *\volley* subfolder, which was created because the Volley library was used making networking easier and faster. This folder contained files that store data regarding user's requests to the server. For example, each time user was selected to know in real time the position of a bus, a corresponding request to the server with all the necessary information was created. It contained the IP address to which the Android application communicates and parameters related to the information the user was looking for. For example, Figure 6 illustrates a user request made for the location of bus (*act = getBusLocation*) with 2005 Route ID (p1 GET parameter).

4 Discussion

Our analysis shows that despite their criticality, the examined applications either store personal information in an insecure manner (i.e., Application A stores the PIN using a static key) or in plaintext (i.e., Application B and C store several personal information in tables without encryption). The findings of this research can be summarized in the following table:

Table 1: Summary of findings

Application	Application Category	Findings
Application A	e-Banking	User's PIN
Application B	Mobile Network carrier	User's personal information (username, password, credit card number, owner and CVV) unencrypted in a rooted device
Application C	Public transport	User's personal information regarding public transportation (favorite routes, history of previous searches) in a rooted device

These findings could prove valuable to forensic investigators but could also severely impact user's privacy, if used by adversaries. If intruders gain access to user's device, they could unencrypt user's PIN and perform transactions without user's consent, which could lead to serious financial damage (Application A). The same scenario applies when intruders acquire user's device and consequently extract user's credit card information such as number, owner and CVV (Application B). Even in cases where the data leakage seems less important as there is no financial loss, its effects on user's privacy could prove harmful. For example, if adversaries manage to exploit user's device, they could find their favorite routes and history searches and hence, use this information to physically insult the victim (Application C). Additionally, attackers could use this information to track user's movements, discover their home or job location, identify their habits (for instance, if the user searches the routes of a specific stadium every Friday afternoon, an attacker could deduce that this user supports a specific football team).

However, we argue that in the near future the situation may change in the sense that new devices incorporate additional security technologies including the Trusted Execution Environment (TEE) [4], which stores keys in an isolated - from the main OS - environment. Thus, if the application stores personal data using an encryption key stored in the TEE, an investigator has no means to read the encrypted key even with root privileges. The latest versions of Android (i.e., ≥ 7.0) enforce the use of TEE (if it is available) using the *Android keystore system.* The latter is used to generate and manage key material,

which is bound to the secure hardware and can never be exposed to the outside environment. On top of that, disk encryption in Android uses a key which is located in the TEE and therefore cannot be extracted. Accordingly, new forensic methodologies and research is required for such cases. It is important though, that developers incorporate state-of-the-art security techniques, especially when dealing with user’s personal information.

5 Conclusions

In this research, we have combined different techniques and tools in order to perform a forensic analysis on applications that are widely used in our everyday life. We performed both source code analysis and disk forensics with the intent to examine all aspects of forensic analysis on Android applications. The results of this work revealed various user’s personal information that is stored insecurely and provide important forensic evidences. It is evident that a leakage of crucial information such as user’s credit card information and transactions, could seriously violate user’s privacy. Even in cases where the leaked information seems less crucial, such as user’s favorite bus routes, their privacy hence, their safety could be seriously harmed. The need to develop new security techniques and implement them correctly is imperative. Developers need to stay updated on new security techniques and apply them by design.

ACKNOWLEDGMENTS

This work was supported in part by the Postgraduate Programme: «Digital Systems Security» of the University of Piraeus, by the FutureTPM project of Horizon H2020 Framework Programme of the European Union, under GA number 779391, and by the H2020-MSCA-RISE-2017 SealedGRID project, under GA number 777996

REFERENCES

[1] Mobile Internet - Statistics & Facts https://www.statista.com/topics/779/mobile-internet/ (Last Accessed September 2018)
[2] Number of apps available in leading app stores as of 1st quarter 2018 https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (Last Accessed September 2018)
[3] App Download and Usage Statistics http://www.businessofapps.com/data/app-statistics/ (Last Accessed September 2018)
[4] Trust Execution Environment https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf (Last Accessed September 2018)
[5] C Ntantogian, D Apostolopoulos, G Marinakis, & C Xenakis (2014). Evaluating the privacy of Android mobile applications under forensic analysis. Computers & Security, 42, 66-76.
[6] R Chanajitt., W Viriyasitavat, & KKR Choo (2018). Forensic analysis and security assessment of Android m-banking apps. Australian Journal of Forensic Sciences, 50(1), 3-19.
[7] DR Hayes, C Snow, & S Altuwayjiri (2017). Geolocation Tracking and Privacy Issues Associated with the Uber Mobile Application. In Proceedings of the Conference on Information Systems Applied Research, 2167, 1508.
[8] H Zhang, L Chen, & Q Liu (2018, March). Digital Forensic Analysis of Instant Messaging Applications on Android Smartphones. In 2018 International Conference on Computing, Networking and Communications (ICNC), 647-651.
[9] C Anglano, M Canonico, & M Guazzone (2017). Forensic analysis of telegram messenger on android smartphones. Digital Investigation, 23, 31-49.
[10] N Mata, N Beebe, & KKR Choo (2018). Are Your Neighbors Swingers or Kinksters? Feeld App Forensic Analysis. In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 1433-1439.
[11] L Zhang, F Yu, & Q Ji, (2016). The Forensic Analysis of WeChat Message. In 2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), 500-503.
[12] AH Lone, FA Badroo, KR Chudhary, & A Khalique (2015). Implementation of Forensic Analysis Procedures for WhatsApp and Viber Android Applications. International Journal of Computer Applications, 128(12), 26-33.
[13] C Anglano, M Canonico, & M Guazzone (2016). Forensic analysis of the chatsecure instant messaging application on android smartphones. Digital investigation, 19, 44-59.
[14] Apktool https://ibotpeaches.github.io/Apktool/ (Last Accessed September 2018)
[15] dex2jar https://github.com/pxb1988/dex2jar (Last Accessed September 2018)
[16] jd-gui http://jd.benow.ca/ (Last Accessed September 2018)
[17] I Muslukhov, Y Boshmaf, & K Beznosov (2018). Source Attribution of Cryptographic API Misuse in Android Applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 133-146.
[18] Root Browser http://rootbrowsers.com (Last Accessed September 2018)
[19] DB Broswer for SQLite http://sqlitebrowser.org (Last Accessed September 2018)
[20] Droid Explorer https://github.com/camalot/droidexplorer (Last Accessed September 2018)