ISBN 978-981-14-1455-8
Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering
WCSE_2019_SPRING
Yangon, Myanmar, February 27-March 1, 2019, pp.171-177
doi:10.18178/wcse.2019.03.029

Security Analysis of Android Application by Using Reverse Engineering

May Thu Kyaw [1], Yan Naung Soe [2] and Nang Saing Moon Kham [1+]

[1]University of Computer Studies, Yangon
[2]University of Computer Studies, Yangon

Abstract. Mobile devices have developed tremendous popularity over the last few years. The most popular usage is the smart phones because they are capable of providing services such as banking, social network, and so on. The Android platform is the fastest growing market in smart phone operating systems to date. The malicious applications targeting the Android system have exploded in recent years. It needs to detect the malicious code on Android applications. This paper focus on the analysis of the android apps by using the reverse engineering tools for checking the malicious activities. There are mainly two parts this analysis such as permissions and java source codes analysis. The results show that most of malware apps are located the unnecessary permission on AndroidManifest.xml to inject the malicious codes in the apps.

Keywords: Android Security, Reverse Engineering, Static Analysis, Android Malware

1. Introduction

Most of malware attacks are targeting Android operating system because of the growing market of smart phones, called Android, and this is a most popular operating system, open source platform of Google. Android is mainly used in mobile devices such as smart phone and tablets. They support several features such as Wi-Fi, Bluetooth, voice, data, GPS, etc. And, they also provide the useful services such as gaming, internet browsing, banking, social networking, etc.

According to the data from International Data Corporation (IDC), the world-wide smartphone market grew 0.7% year over year, with 344.7 million shipments. The worldwide smart phone market will reach a total 1.53 billion units shipped in 2017. Android dominates the market with an 85.0% share in 2017 [1].

Android is one of the most popular operating system because it is an open source operating system. It has some basic features such as middleware in the form of virtual machines, system utilities and applications. The most attractive feature is the ability to extend its functionality with third-party applications. But this feature brings with it the threat, attacks of malicious applications. The increase of mobile applications causes the challenges of security that is the vulnerable of the applications and these become the target of malicious application developers.

According to the report, the large population of potential victims give malware writers to target mobile devices and states that the number of new smart phone malware simples detected has doubled from 1000 per day in 2013 to 2000 per day in 2014 [2]. Based on these facts, the android malware increased to the double rate within 2014 and 2015. In the Trend Micro 2016 Security Predictions report, CTO, Raimund Genes predicted the following: China will drive mobile malware growth to 20 million by the end of 2016 [3].

+ Corresponding author.
E-mail address: moonkham@ucsy.edu.mm

Table I: The five biggest android malware attack in 2017[4]

Name	Form of Attack
Expensive Wall	A form of malware
Marcher	A form of adobe flash player update
Xavier	A form of Trojan adware
Dvmap	Injected puzzle game, Colourblock
Bankbot	Injected a game, Jewels Star Classic

In the proposed system, it is used the reversed engineering tool such as apktool, dex2jar and jdgui for static malware analysis. This paper is organized as follows. In section II, it will discuss about the related work of the previous research work. Section III will be expressed background theory about android architecture, security and malware. The implementation and analysis results will be explained in Section IV and this paper will conclude in Section V.

2. Related Works

There are many researches works that are implemented on android malware analysis. Most of them are based on static analysis and these are used reverse engineering methodology to analyze the apps. C.Y. Huang et al. [5] proposed their research work with the performance evaluation on permission-based detection for android malware. They analyzed the required and requested the permissions for application and labels the apps as benign or malware using site based, scanner based and mixed labeling. And then, they used machine learning algorithms on three data sets and evaluates the permission-based malware detection performance. It can detect 81% of malicious application just upon their dataset.

S. M. A. Ghani et al. [6] presented the static analysis technique that extracted the android apps including benign and malware for getting their original source code. They compared API and manager classes from these apps and categorized them. The most frequent API and manager class used in malware will be detected. They extracted the feature by using Androguard, a reverse engineering tool and compared the extracted source code by categorized the APIs and manager classes. Their result show that there is relationship between API and manager classes in malicious apps.

Y. Cuixia et al. [7] proposed the tool to design a UI modeling method in Android. It based on attribute graph by using reverse engineering and program analysis for applications. Their method is to detect repackaging detection for malware and assessmentation of apps family. Therefore, their method can also be used to detect repackaged apps by checking the UI, functions and appearances similarity between member families. Their approach achieved 94.74% detection rate at UI and 26.13% at repackaging detection. And, they show tht 50% of repackaged apps use the same UI. Their result shows that the UI modeling method helps to detect repackaged applications include malicious apps. J. Y. Pan et al. [8] proposed the framework to eliminate the advertisement by filtering or redirection for targeted application. Some of them require root permission. They develop an advertisement removal program with reverse engineering. However, this proposed method cannot work on customized code of loading advertisement.

A number of researchers introduced the permission-based malware detection. The performance evaluation of permission-based detection [5] is also implemented this type of detection. But the permission list is still the minimum defense for a user to detect whether an app could be harmful. These works can't completely grantee for detection malware because the benign app can also use the same permission like that malware. In [6], [7], and [8], the static analysis is used the reverse engineering tools to detect malicious nature such as repackaging and advertisement.

But it is still needed to implement the effective malware detection framework because the previous researches works are partially effective in their proposed works and some gaps such as detecting of unknown malware and reducing of false positive alarm are still remained. The main gap of the current research works is only effective in well-known attacks because of the rise of the malware attacks and the budding of malware natures such as ADB.Miner, a copycat from Marai which is IoT botnet.

3. Background Theory

This section will discuss about android application architecture, security and malware that are populated on recent years.

3.1. Android Application Architecture

The APK bundle is the format used to package the android apps that can be got from Google Play Store or any third-party markets [9]. An APK file is basically a ZIP file, it can be renamed and can be extracted their contents. Table II shows the basic architecture of the android application.

Table II: Android Application Architecture [10]

Entry	Notes
AndroidManifest.xml	The manifest file in binary XML format to set the resources permission.
classes.dex	The application code compiled in the dex format.
resources.arsc	This file contains precompiled application resources, in binary XML.
res/	This folder contains resources not compiled into resources.arsc
assets/	This folder contains applications assets, which can be retrieved by Asset Manager.
lib/	This folder contains compiled code, native code libraries.
META-INF/	This folder stores meta data about the contents of the JAR. The signature of the APK is also stored in this folder.

3.2. Android Security

Android apps run in separate processes under distinct Unix user identifiers (UIDs) each with distinct permissions as shown in Fig. 1. Programs can't either read or write each other's data or code of apps, and applications must be done explicitly for sharing data. There are two levels for android security such as Linux Kernel level and Application Framework level.

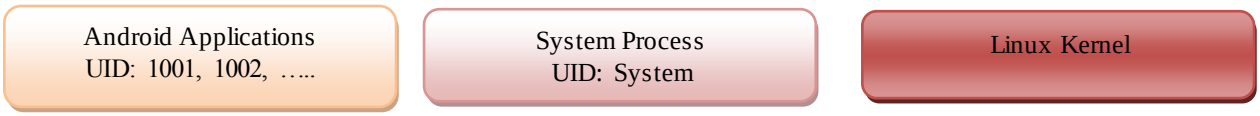

Fig. 1: Android security model.

- Linux Kernel Level Security: Android relies on Linux both of the process, memory and file system management. It is also one of the most important components in the Android security architecture. And, it is responsible for provisioning Application Sandboxing and enforcement of some permission.
- Application Framework Level Security: Android applications consist of different components and there is no central entry point unlike Java programs with the main method. Therefore, it is needed to declare the resources permission by the developer of an application in the AndroidManifest.xml file.
- Android Permission: The Android operating system uses a permission-based model not only to limit the behavior of an application but also to inform the user of the application's potential behavior. An application is needed to declared the required permissions in AndroidManifest.xml file. The user can decide to grant the list of permissions, an application requests when it is to be installed. The user gets to make the choice whether or not to install the application based on the list of permissions. Once an application is installed, the permissions that it has remains static. The android permission classified into four different levels is shown in Table III.

Table III: Android Application Permission Level

Permission Level	Notes
Normal	These cannot impart real harm to the user (e.g. change the wallpaper)
Dangerous	These can impart real harm (e.g. call numbers, open Internet connections, etc)
Signature	These are automatically granted to requesting app if that app is signed by the same certificate.
Signature/ System	Same as Signature, expect that the system image gets the permissions automatically as well and it is designed only to use by device manufacturers.

4. Methodology

There are some reverse engineering tools that are used to analysis and check the applications for mobile security. The apktool [10] is used to extract the permission file. Dex2jar [11] is used to re-convert the *.jar file from original apps and Jdgui [12] is used for viewing the java code from *.jar file. The proposed flow of malware analysis architecture is shown in Fig. 2.

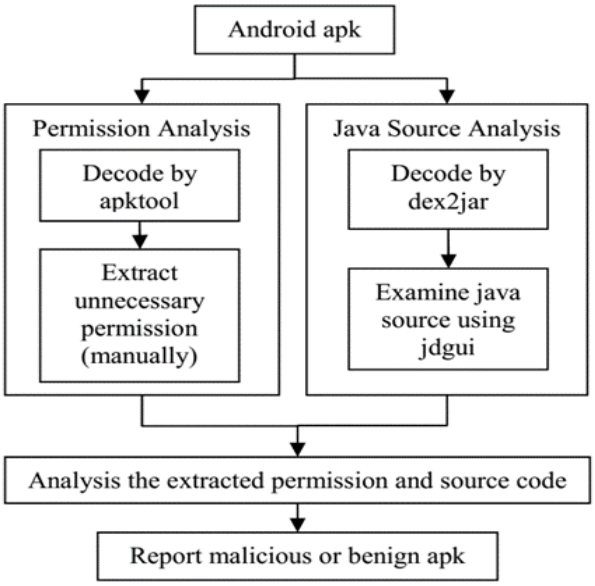

Fig. 2: Flow of the analysis

The analysis has basically two parts, permission and source code analysis. For the permission analysis, it will use apktool that it can extract the AndroidMinifest.xml and original bytecodes. These codes are based on the machine code which is implemented in smali language. This analysis will only use permission file because smali codes are difficult to analysis and it will need several processes. The permission file is important on android apps because it is needed to set the permission for several resources that are implemented in source codes. Most of the malicious apps use the unnecessary permission that isn't related with their apps, and it is needed to extract from AndroidMinifest.xml.

For source code analysis, dex2jar and jdgui tools are used. The first tool can convert the android apk to *.jar file based on the java byte codes. But these are also difficult to analysis due to the implementation of java based machine language. So, the second tool will be used to decompile *.jar file and it can generate the original java sources. In the proposed analysis, the suspected android apps are carefully analyzed the source code if these apps will use unnecessary permissions. Finally, it will report the selectd app (apk) is malicious or benign app.

Table IV: Analysis Result of Calendar apks

Apks	Manifest Permission
iCalendar.apk	INTERNET, ACCESS_COARSE_LOCATION, RESTART_PACKAGES, RECEIVE_SMS, SEND_SMS, SET_WALLPAPER
Kalendar Indonesia.apk	INTERNET, ACCESS_NETWORK_STATE
Calendar.apk	RECEIVE_BOOT_COMPLETED, WRITE_EXTERNAL_STORAGE, READ_CONTACTS, VIBRATE, READ_CALENDAR, WRITE_CALENDAR, WAKE_LOCK

Table IV shows the analysis result of three types of calendar apps such as iCalender.apk, Kalendar Indonesia.apk and Claendar.apk. Among them, iCalender.apk is one of the malicious app and the other apps are benign apps. In Kalendar Indonesia.apk, it only used two permissions including internet access permission. This permission used to implement for adding the ads in the application but it didn't add any dangerous malicious code in the app. In Calendar.apk, there are seven access permissions but these

permissions are used only for giving the calendar facilities. It is not using any malicious code and it is also a benign app. But, some unnecessary permissions (sms permissions) are used in iCalendar.apk and it can be malicious app. The detail analysis of this app will be shown in Table V.

Table V: Analysis Result of Icalendar.apk

Manifest Permissions	Malicious Codes
INTERNET, ACCESS_COARSE_LOCATION, RESTART_PACKAGES, RECEIVE_SMS, SEND_SMS, SET_WALLPAPER	SmsManager.getDefault().sendTextMessage("1066185829",null,"921X1",PendingIntent.getBroadcast(this, 0, new Intent(), 0), null);

Some of the analysis results of the android apps (apk) that malware apps are shown in Table V, VI and VII. In these tables, the left column is the manifest permission list of the analyzed app and the right column is the malicious code in the app. The existing several applications like Skype that needs many accesses to various data on the phone; but there are a few applications like Wallpaper, Calendar, etc that require very few permissions. In these tables, the analysis results are for a calendar application and two simple games. But these applications used several permissions and including unnecessary codes in the packages. Other tested applications also include several permissions and some malicious codes such as SMS receive/send, read content lists, location access and so on.

Table V is showed the analysis result of calendar application that is called iCalendar.apk. It is only simple application that it only needs to show the date of the years, but it used several unnecessary permissions such as internet and sms. After checking the source code, it also used the malicious code that is used for sending information to premium number (1066185829). So, the analysis can determine that this app is a malicious application.

Table VI: Analysis Result of Qqgame.apk

Manifest Permissions	Malicious Codes
WRITE_SMS, RECEIVE_BOOT_COMPLETED, VIBRATE, READ_SMS, RECEIVE_SMS, SEND_SMS, READ_PHONE_STATE, DISABLE_KEYGUARD, READ_CONTACTS, WRITE_CONTACTS, INTERNET, ACCESS_NETWORK_STATE, READ_PHONE_STATE, CALL_PHONE, WAKE_LOCK, RESTART_PACKAGES, WRITE_APN_SETTINGS	String str = paramIntent. getStringExtra("ObjNG0Zw5A"); Intent localIntent = new Intent("android.intent.action.CALL ", Uri.parse("tel:" + str)); paramContext.startActivity(localIntent)

Other analysis are based on the android games, qqgame.apk and suiconfo.apk. These results are shown in Table VI and VII. These apps also used the unnecessary permissions that are not related with the game features. In qqgame, the usage of sms and contact permissions are not related with this game feature. These can utilize to keep and watch on the users of their calls. And, it is trying to use the intent.action class by passing the specific number from the predefined string (ObjNG0Zw5A).

In suiconfo, it also used the unnecessary permissions such as location access, sms send and contact read. And it used the malicious codes which are implemented to send the personal information to the premium number (0646112264) as in the background process. The users can only know that they only play the game but their personal information is stolen in the background by the malicious developer.

Most of malware apps include the malicious code that can read contact data to be used to send span messages of just keep track of the user's personal data. Some of apps can be finding GPS location. The permission of android apps can enable an application to track the collect information regarding the user who does not comfortable providing. The internet access permission is also the most command and dangerous permissions. This internet access permission is requested by all application that supports advertisements, video games, etc. But, most of the freeware apps used internet access permissions to use the advertisement purpose.

Table VII: Analysis Result of Suiconfo.apk

Manifest Permissions	Malicious Codes
INSTALL_PACKAGES, USE_CREDENTIALS, INTERNET, BLUETOOTH_ADMIN, DEVICE_POWER, READ_CONTACTS, SEND_SMS, ACCESS_LOCATION, ACCESS_GPS	Object[] arrayOfObject = (Object[])paramIntent. getExtras().get("pdus"); SmsMessage[] arrayOfSmsMessage = new SmsMessage [arrayOfObject.length]; String str1 = arrayOfSmsMessage[0].getMessageBody(); SmsManager.getDefault().sendTextMessage("06 46112264", null, str1, null, null);

There are mainly used the reverse engineering tools such apktool, dex2jar and jdgui for these analyses. Actually, apktool can generate the permission file, source codes and resource files. It is useful for static analysis to extract the unnecessary permission usages and malicious code. But these extracted source codes are implemented with smali, bytecode format. It is difficult to understand and it can't easily extract malicious features. Therefore, dex2jar and jdgui tools are needed to use for extracting the malicious features. Dex2jar can convert the android bytecodes (dex format) to java bytecodes (jar format). This jar format is a package of java (.class) files combination. Jdgui can translate these files (.class) to (.java). After that, it can easily extract the malicious features. If it is possible to analyze the smali codes, apktool can only be used for static analysis.

5. Conclusion

This analysis has to break apart the application or malware using the reverse engineering tools and techniques. For this analysis, the results are based on the manually checking mechanism after converting to the original source codes by using the reverse engineering tools. As the results, some apps consist of the unnecessary permissions which are used to inject the malicious code for stealing the information. For this reason, the user should need to check the usage the permission of the apps when it will be installed in the mobile devices. As the future work, the automatic detection mechanism will be proposed for checking the malicious features.

6. Acknowledgements

We thank Yan Naung Soe, University of Computer Studies, who provided insight and expertise that greatly assisted the research, and Nang Saing Moon Kham, professor, University of Computer Studies, for comments that greatly improved the paper.

7. References

[1] "Smartphone OS Market Share, 2016 Q2", [Online], http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] "Cumulative Number of Android Malware in 2015", [Online], https://www.itvoice.in/index.php/it-voice-news/android-malw are-doublyed-in-2015-vs-2014-reports-trend-micro-2015-threat-report

[3] "Continued Rise in Mobile Threats for 2016", [Online], http://blog.treandmicro.com/continued-rise-in-mobile-threats-for-2016 "Mobile Malware", [Online],

[4] V. Beal, "Adware", [Online], https://www.webopedia.com/TERM/A/adware.html

[5] C.Y. Huang, Y.T. Tsai, and C.H. Hsu, "Performance evaluation on permission-based detection for android malware", Adv. Intell. Syst. Appl. - Vol. 2, vol. 21, pp. 111–120, 2013.

[6] S .M. A. Ghani, M. F. Abdollah, R. Yusof, M. Z. Mas'ud, "Recognizing API Features for MalwareDetection Using Static Analysis", Journal of Wireless Networking and Communications, 2015.

[7] Y. Cuixia, Z. Chaoshun, G. Shanqing, H. Chengyu, C. Lizhen, "UI ripping in android: reverse engineering of graphical user interfaces and its application", IEEE Conference on Collaboration and Internet Computing, 2015.

[8] J. Y. Pan, S. H. Ma, "Advertisement Removal of Android Applications by Reverse Engineering", Workshop on Computing, Networking and Communications (CNC), 2017.

[9] "Backdoor"http://searchsecurity.techtarget.com/definition/back-door

[10] “Smali/Baksmali”, [Online], https://github.com/JesusFreke/smali

[11] “Apktool”, [Online],https://ibotpeaches.github.io/Apktool/

[12] “Dex2jar”, [Online], https://sourceforge.net/projects/dex2jar/