ISSN No:-2456-2165

Analysis of Android Applications by Using Reverse Engineering Techniques

Soe Myint Myat
Myanmar Aerospace Engineering University

May Thu Kyaw
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.*

I. 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 [7]. The world wide smart phone market reach a total 355.2 million units shipped in 2018 and Android will dominate the market with an 89.0% share in 2019.

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 [8]. 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 [9].

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

Table 1:- The Five Biggest Andorid Malware Attack in 2017

The five biggest android malware attacks in 2017 [20] are shown in Table I. The first one, Marcher is found on third-party markets and other malware attacks are discovered from Google Play store. Expensive Wall sent fake messages and charged without users' permission. The second one, Marcher would disable security, removes its icon, sent all device's information to C&C when the users open an app from it list of targets. It could steal login credentials from retail, social media and banking apps. The third one, Xavier can quietly store personal and financial data from users by hiding inside the several types of apps such as ringtone changers, photo manipulators, call recorders and so on. Another one, Dvmap could inject code into system library and eliminate root detection software by hiding inside puzzle game, Colourblock. And another attack, Bankbot created fake overlay screens which looked like the login pages of popular banking apps by injecting inside a game, Jawels Star Classic. And then the data was passed onto cybercriminals when they entered their login credential.

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. Reverse engineering methodology and tools will be discussed in Section IV. The implementation and analysis results will be explained in Section V and this paper will conclude in Section VI.

II. RELATED WORK

There are many research 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. [2] 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. [3] 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 are relationship between API and manager classes in malicious apps.

Y. Cuixia et al. [1] 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. [4] 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 the technique of reverse engineering, which can effectively patch the advertising code, even obfuscated by other tools. 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 [2] 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 [1], [3], and [4], 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.

III. BACKGROUND THEORY

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

A. 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 [13]. 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.

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.

Table 2:- Android Application Architecture [13]

B. 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 Kernal level and Application Framework level.

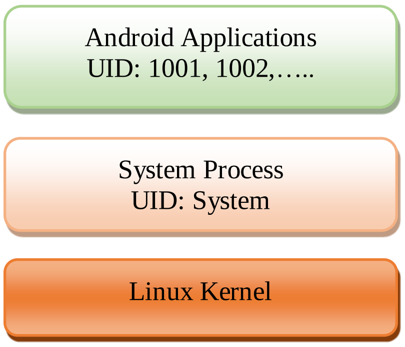


Fig 1:- Android security model

➢ *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. Permissions are used for protecting the access to the system resources. The third-party applications developers may also use custom permissions to guard the access to the components of their applications.

➢ *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.

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.

Table 3:- Android Application Permission Level

C. Android Malware

Mobile malware is malicious software that targets mobile phones by causing the crash of the system and stealing the confidential information. The first known mobile virus, "Timofonica", originated in Spain and was identified by antivirus labs in Russia and Finland in June 2000 [10].

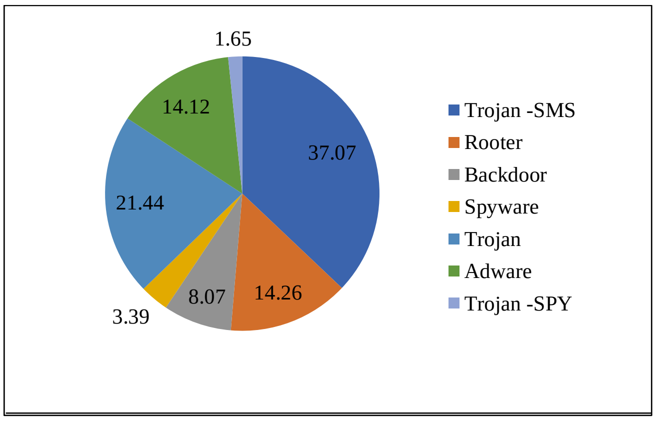


Fig 2:- The growing threat of android mobile malware [10]

The top android malware families are shown in Fig. 2. Trojan is the most spread types in android malware. All types of Trojan malware are totally 60.16% all of malware [11]. The second more attack type is the Advertising Malware (Adware).

The behavior of different malware families is provided in subsequent sections.

➢ *Trojans*

Trojans appear to a user as benign application but it is actually steal the user's confidential information without the user's knowledge. Such applications can easily get access to the browsing history, messages, contacts and device IMEI numbers [8]. Mobile banking Trojans can run together with Win-32 Trojans to bypass the two-factor authentication and the theft of banking verification codes that banks send their customers in SMS messages. These trojans attack a limited number of bank customers and it can invent new techniques to allow them for expanding the number and the geography of potential victims.

➢ *Rooter*

Originally, the word "root" is used to refer to the root account on Linux, that is to say, the system administrator, who has all the rights on the device and can modify all OS elements as it sees fit, including sensitive files. The rooted phone or tablet means that the users get the system administrator level and can control every resources on the devices. The root may include a phone blocking risk.

➢ *Adware*

Adware is a software that contains advertisements embedded in the application. Adware targeted to the users who do not wish to pay the software cost. There are many adware ad-supported programs, games or utilities that are distributed as adware [19].

➢ *Backdoor*

A backdoor is a hidden program to bypass the security mechanisms. Sometimes, the developer may install back door for troubleshootional purposes. Backdoors can utilize the root exploits to grant root privileges for malwares and help them to hide from antivirus.

➢ *Spyware*

There are many reports the spyware as the serious threat for mobile users Spyware threats are also highly persistent according to security company and 0.24% of Android devices that they scanned in the U.S. had surveillance-ware installed intended to target a specific individual [18].

➢ *Botnet*

Botnet is a network of compromised Android devices which is running one or more bots. Botmaster, is also called a remote server controls the botnet through the Command and Control Server (C&C) network. The botnet tendencies to actually hijack and control infected devices.

D. *Reverse Engineering*

Reverse engineering is called back engineering. Reverse engineering can also be the process of extracting knowledge or design information from a product that can be hardware or software. Fig. 3 shows the general reverse engineering process. To make the source code translation, it is needed to use the automated tools that can convert one language to another. Source code translation is a process of converting from a language to another. This may be machine bytecodes to original source codes. It is needed to translate the original program to required human readable format. After that, it is needed to note the program structure as the documentation. Most of the programs are too large, it is necessary to pass through program modularization process. Program modularization is a process of subdividing a program into separate sub-programs. After getting the modularized programs, it is easy to analyze the whole program. Reengineering of data components of existing system can be done with the help of methods and software tools. It extends the life of existing systems by standardizing data definition and facilitating source code simplification. It is also called data reengineering process.

Reverse engineering can reproduce the original one or reproducing anything based on the extracted information. In android application, there are many reasons for using reverse engineering. It can be trying to hack or inject malicious code into an application. Repacakging is a methodology to modify an application with a particular layout or animation by using the tools that could access the XML resource files of interest.

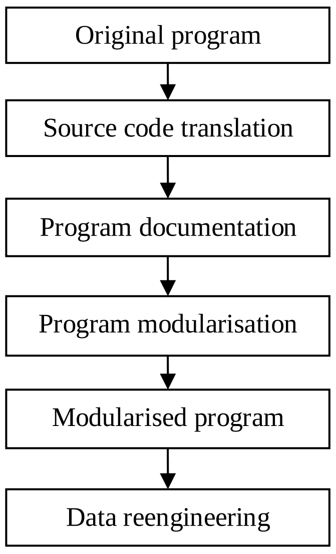


Fig 3:- Reverse eingineering process

Reverse engineering techniques can also be used to inject the modified code in the original one and it is also called repackaging application. Therefore, it is always a good practice to check the developers who develop the application for security reasons. It is needed to check the code or the resources that have been effectively obfuscated or to be sure that unwanted files have not been packaged into the final release APK, including the information like API keys, authentication tokens or unused resources [13]. On the other way, reverse engineering techniques or tools can be used to detect not only repackaged apps but also malicious apps.

There are many tools for reverse engineering for android applications and the following are some of popular tools.

➢ *SMALI/BAKSMALI*

This tool is an assembler or dis-assembler for the dex format that is used by dalvik bytecode.

➢ *ANDBUG*

This tool is also a debugger program for dalvik bytecode and it uses the same interface as Android's Eclipse debugging plugin.

➢ *ANDROGUARD*

This tool is a full python tool to perform with android files such as dex, apk, xml and bytecode resources.

➢ *APKTOOL*

It is the most useful tool for android reverse engineering. It can be used for both decompiling and re-compiling the android apps. It can generate smail, xml and resources file.

➢ *DEX2JAR*

This tool can work with android.dex and java bytecode *.class files. It can convert android(dex) file to bytecode package (jar) file.

➢ *JD-GUI*

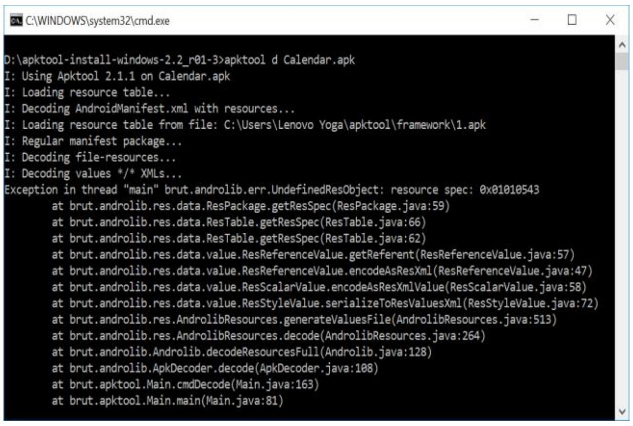
It is a graphical user interface tool to display java source code from java bytecode (class) file.

IV. METHODOLOGY

This paper will focus on the malware analysis for Android apps by using reverse engineering tools and static analysis. There are mainly two steps to extract the required files for static analysis. The first one, it needs to extract permission information from AndroidManifest.xml of the apps. It needs to use apktool for extracting the permission file. The second one to needed to analyse the java source codes for extracting the malicious codes. For this step, dex2jar can be used to convert android (dex) file to java bytecode (jar) file. But, it is still needed to translate these java bytecodes to java source codes. For this process, jdgui can translate the java (class) file to source code (java) file.

A. Reverse Engineering Tools

There are many tools for reverse engineering for android applications. Among them, the following tools will be used for the malware analysis.



```
C:\WINDOWS\system32\cmd.exe

D:\apktool-install-windows-2.2_r01-3>apktool d Calendar.apk
I: Using Apktool 2.1.1 on Calendar.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Lenovo Yoga\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
Exception in thread "main" brut.androlib.err.UndefinedResObject: resource spec: 0x01010543
        at brut.androlib.res.data.ResPackage.getResSpec(ResPackage.java:59)
        at brut.androlib.res.data.ResTable.getResSpec(ResTable.java:66)
        at brut.androlib.res.data.ResTable.getResSpec(ResTable.java:62)
        at brut.androlib.res.data.value.ResReferenceValue.getReferent(ResReferenceValue.java:57)
        at brut.androlib.res.data.value.ResReferenceValue.encodeAsResXml(ResReferenceValue.java:47)
        at brut.androlib.res.data.value.ResScalarValue.encodeAsResXmlValue(ResScalarValue.java:58)
        at brut.androlib.res.data.value.ResStyleValue.serializeToResValuesXml(ResStyleValue.java:72)
        at brut.androlib.res.AndrolibResources.generateValuesFile(AndrolibResources.java:513)
        at brut.androlib.res.AndrolibResources.decode(AndrolibResources.java:264)
        at brut.androlib.Androlib.decodeResourcesFull(Androlib.java:128)
        at brut.androlib.ApkDecoder.decode(ApkDecoder.java:108)
        at brut.apktool.Main.cmdDecode(Main.java:163)
        at brut.apktool.Main.main(Main.java:81)
```

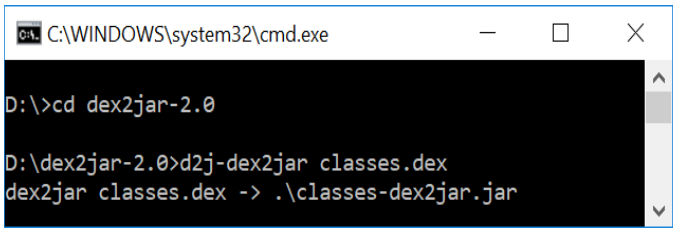
Fig 3:- Decoding of android app using apktool

➢ *APKTOOL*

A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and it can rebuild them after making some modifications [15]. It also makes working with an app easier because of project-like file structure and automation of some repetitive tasks like building apk, etc. It can extract the original source as smali code and the important permission file. Decoding process of android app using apktool is shown in Fig. 4.

➢ *DEX2JAR*

This tools is to work with android.dex and java .class files [16]. It can convert the classes.dex file to classes-dex2jar.jar file. This jar file is the combination of original source class files. The process of this tool is shown in Fig. 4. It is useful for extracting the original sources as java byte code.

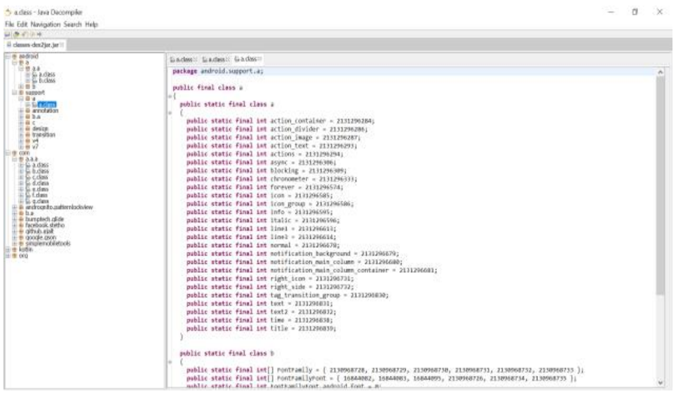


```
C:\WINDOWS\system32\cmd.exe

D:\>cd dex2jar-2.0

D:\dex2jar-2.0>d2j-dex2jar classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
```

Fig 4:- Regenerating of bytecode using dex2jar

➢ *JD-GUI*

JD-GUI is a standalone graphical utility and it can display Java source codes from java object code ".class" files. It can be browsed the reconstructed source code with this tool for instant access to methods and fields [17]. This tool can translate *.jar file to *.java code. It is useful to check the source codes as java based language. Fig. 5 shows the reconstruction of java code from dex file using dex2jar.



```
package android.support.a;

public final class a
{
  public static final class a
  {
    public static final int action_container = 2131296284;
    public static final int action_divider = 2131296286;
    public static final int action_image = 2131296287;
    public static final int action_text = 2131296293;
    public static final int actions = 2131296294;
    public static final int async = 2131296306;
    public static final int blocking = 2131296309;
    public static final int chronometer = 2131296333;
    public static final int forever = 2131296574;
    public static final int icon = 2131296585;
    public static final int icon_group = 2131296586;
    public static final int info = 2131296595;
    public static final int italic = 2131296596;
    public static final int line1 = 2131296613;
    public static final int line3 = 2131296614;
    public static final int normal = 2131296678;
    public static final int notification_background = 2131296679;
    public static final int notification_main_column = 2131296680;
    public static final int notification_main_column_container = 2131296681;
    public static final int right_icon = 2131296731;
    public static final int right_side = 2131296732;
    public static final int tag_transition_group = 2131296830;
    public static final int text = 2131296831;
    public static final int text2 = 2131296832;
    public static final int time = 2131296838;
    public static final int title = 2131296839;
  }

  public static final class b
  {
    public static final int[] FontFamily = { 2130968728, 2130968729, 2130968730, 2130968731, 2130968732, 2130968733 };
    public static final int[] FontFamilyFont = { 16844082, 16844083, 16844095, 2130968726, 2130968734, 2130968735 };
    public static final int FontFamilyFont_android_font = 0;
```

Fig 5:- Reconstruction of java code using dex2jar

B. Android Malware Analysis

In order to analyses a malware there are two methods, namely, static analysis and dynamic analysis. This work will only use static approach for malware detection because static analysis is more effective than dynamic analysis. And, the cost of computing cost is low and low cost consuming in static analysis. It can classify as java source code analysis and permission based analysis. For these analysis, it is necessary to use reverse engineering methods or tools.

Static analysis consists of executing a selected sample in a controlled environment to monitor its analysed and it determine whether it is malicious, and what the changes or modifications are in the system. Static analysis is a commonly used tool in malware detection. For Java applications, static analysis works directly on the bytecode and can perform various analyses such as reconstruct the class hierarchy. This analysis find method invocations and extract control-flow and data-flow information [6] from them. In a static analysis, it analyze the apk file which has a common characteristic with jar file for detection the malicious application. After the resources such as files and folders are extracted, the static analysis will be mainly focused on two components such as AndroidManifest.xml and classes.dex [18]. This xml file is one of main feature because all the processes need to set the permission in this file. Therefore, it is needed to check whether unnecessary permission will be used or not.

V. IMPLEMENTATION

There are some reverse engineering tools that are used to analysis and check the applications for mobile security. The apktool is used to extract the permission file. Dex2jar is used to re-convert the *.jar file from original apps and Jdgui is used for viewing the java code from *.jar file. The proposed flow of malware analysis architecture is shown in Fig. 6.

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 smili language. This analysis will only use permission file because smali codes are difficult to analysis and it will need several process.

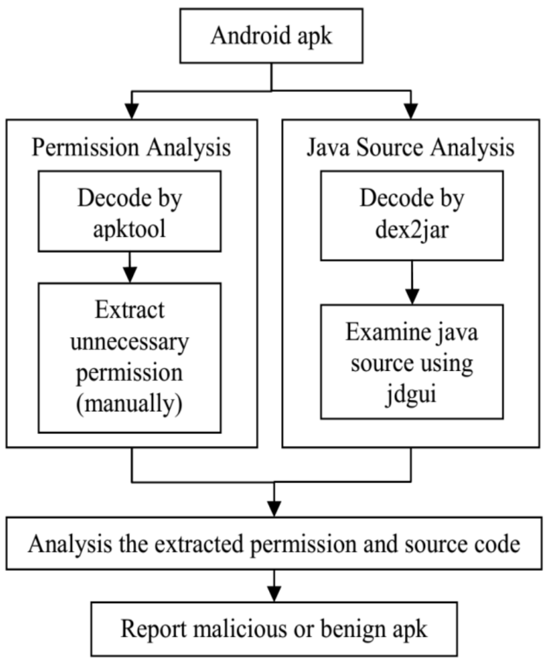

Android apk
Permission Analysis
Decode by apktool
Extract unnecessary permission (manually)
Java Source Analysis
Decode by dex2jar
Examine java source using jdgui
Analysis the extracted permission and source code
Report malicious or benign apk

Fig 6:- Flow of the Analysis

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 Android Minifest.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.

Apk	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 4:- Analysis Result of Calendat Apks

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.

Manifest Permissions	Malicious Codes
INTERNET, ACCESS_COARSE_LOCATI ON, RESTART_PACKAGES, RECEIVE_SMS, SEND_SMS, SET_WALLPAPER	SmsManager.getDefault ().sendTextMessage("10 66185829", null, "921X1", PendingIntent.getBroadc ast(this, 0, new Intent(), 0), null);

Table 5:- Analysis Result of iCalendar.apk

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 access 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 permission and including unnecessary codes in the packages. Other tested applications also include several permission 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 permission 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.

Manifest Permissions	Malicious Codes
WRITE_SMS, RECEIVE_BOOT_COMPLETED, VIBRATE, SEND_SMS, READ_SMS, RECEIVE_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);

Table 6:- Analysis Result of qqgame.apk

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.

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("0646112264", null, str1, null, null);

Table 7:- Analysis Result of SuiConFo.apk

There are mainly used the reverse engineering tools such apktool, dex2jar and jdgui for these analysis. 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 this 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.

VI. CONCLUSIONS

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.

ACKNOWLEDGEMENT

The authors are grateful for the supports provided by Myanmar Aerospace Engineering University.

REFERENCES

[1]. 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.

[2]. 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.

[3]. 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.

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

[5]. T. K. Barsiya1, M. Gyanchandani, R. Wadhwani, "Android Malware Analysis: A Survey Paper", International Journal of Control, Automation, Communication and Systems (IJCACS), 2014.

[6]. Y. J. Ham, H. W. Lee, "Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events", International Journal of Computer and Communication Engineering, Vol. 3, No. 2, March 2014.

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

[8]. "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

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

[10]. "Mobile Malware", [Online], http://en.wikipedia.org/wiki/Mobile_malware

[11]. "The Growing Threat of Mobile Malware", [Online], http://blogarchive.quickheal.com/wp/the-growing-threat-of-mobile-malware-top-android-malware-families-of-2012/

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

[13]. D. Altomare, "Android Reverse Engineering", November 2016. [Online], http://www.fasteque.com/android-reverse-engineering-101-part-4/

[14]. "Smali/Baksmali", [Online], https://github.com/JesusFreke/smali

[15]. "Apktool", [Online],https://ibotpeaches.github.io/Apktool/

[16]. "Dex2jar", [Online], https://sourceforge.net/projects/dex2jar/

[17]. "Java Decompiler", [Online], http://jd.benow.ca/

[18]. J. Kirschner, [Online], "Moblie Security Apps Perform Dismally Against Spyware", https://www.techlicious.com/review/mobile-security-apps-perform-dismally-against-spyware/

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

[20]. A. Elise, "5 types of Android Malware that made headlines in 2017", December, 2017. [Online], http://www.kcci.com/article/5-types-of-android-malware-that-made-headlines-in-2017/14508001