Analysis of Malicious Apps in Android OS

Pooja Singh,
Dept. of Computer Science,
AMET University, Chennai, Tamilnadu,India

Dr. Santosh Singh
HOD, BSc IT & MSc IT Department,
Thakur College of Science & Commerce, Mumbai, India

Pankaj Tiwari
Dept. of Computer Science,
JJT University, Churu, Rajasthan, India

Abstract—Mobile operating system and mobile devices areplaying an important role in our day to day to life. Some latest examples of Mobile operating system such as iOS and Android have become a market leader. However the increasing use of theses mobile operating system has leads to several security issues also. There have been several approaches proposed on basis of permissions and malware detection, and source code analysis of android apps. However helping the users to know the security implications of mobile application is still ongoing research, in this paper we propose the method to analyze decompiled source code and permission of android apps to show the security threats of android apps and user information stored in mobile devices

Keywords— Malware detection, Permissions, Android apps.

I. INTRODUCTION

It has been constant increase in the use & development of the Android apps for business as well as personal use. The application run on mobile devices requires much permission such as contacts, emails, location and many more. These devices also have monetary risk such as phone calls, messages and mobile data usage can cost money for its usage. As we also know the increase in use of digital wallet applications such as paytm, freecharge etc. These apps require confidential information such as bank details. Due to growing use of theses apps it makes these mobile devices as an attractive target for malicious programs. Researchers have developed and proposed several methods to identify malicious behavior of application on their permission requirement. Some applications demands particular permission [1] but to complete that demand the applications also uses some permissions which are not listed by applications. However, permissions only provide a high-level and inaccurate observation of the performance of an application. An application may request permission without actually using the permission. One permission can control multiple other permissions via access control. For example allowing access to READ_PHONE_STATE will also use device IMEI via getDeviceId() which can be misused. The current caller is available via getCallerInfo(...) which has privacy implications, but this permission also grants access to more commonly used functions and intents such as the “android.intent.action.PHONE_STATE” intent to detect changes in the network connection type and similar changes to phone state. On other side we know that applications can be well studied by their decompiled source code which helps in detail analysis of applications. As Android system provide rich set of API which can be easily analyzed by decompiled source code. Therefore we saw that using decompiled source code the behavior of applications can provide more detail information about applications than the set of permission the applications actually uses.

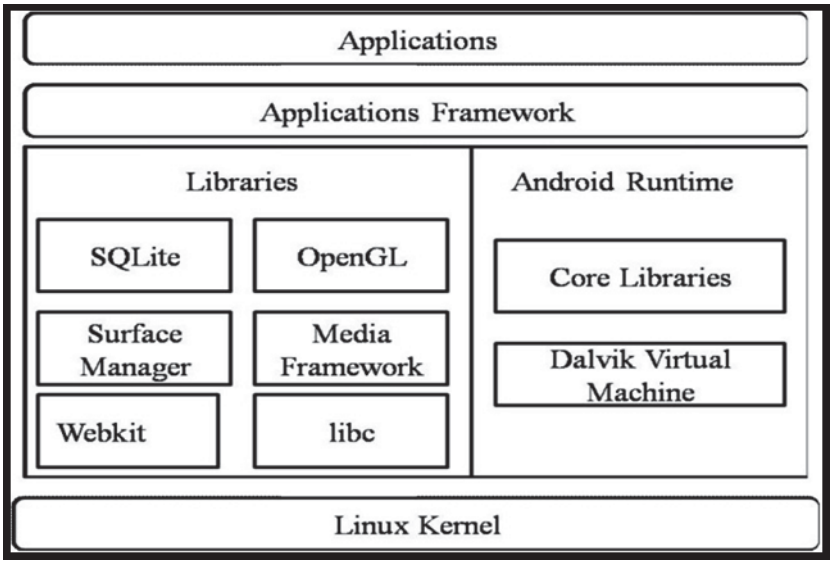


Fig 1. Android Architecture

The organization of the paper is as follows: Section 2 is concerning the Android fundamental framework and the security issues. Section 3 is the security analysis mechanism we proposed for android applications. Methodology is shown in Section 4. Section 5 concludes the whole paper counting the contributions.

II. ANDROID SECURITY ISSUES

2.1 Android basic architecture

Android most popularly known mobile operating system based on Linux kernel and it was developed by Google [2]. Android operating system is available in layered architecture, which includes the bottom layer as Linux Kernel layer, middle layer and top layer as application layer which provides different services to for the layer above. The main function of mobile phone are provided by middle layer, which is mostly implemented in java or C/C++.All the applications running on android operating system are written in java &after compilation it is converted into .class files which are again converted into .dex file by DX tool. Every Android applications has separate instance which runs in DVM [3] (Dalvik Virtual Machine) and each applications are assigned with unique identification number. Fig 1. will show the layered architecture of Android operating system. Android operating system is made of several components out of which DVM (Dalvik Virtual Machine) is important component of Android platform. It support all java applications which are converted into .dex format which Dalvik executable format which can be run on DVM .It is available in compressed format so that memory and processor speedlimitation can be overcome .DVM is also responsible for process separation and thread management..

2.2 Android security Model

Android operating system Security is same as Linux operating system. Android provides different mechanism to protect user's information. The main component of Android security is sandboxing mechanism, applications signature & permission model. The Android permission model restricts applications to access users confidential information such as phone number contacts, location, resources and Internet and GPS. If any application requires access to any resource it has to acquire the corresponding permissions. Android permission model is list of stated permission which is shown before installing the applications. However this mechanism seems to be simple but has several security issues that cannot protect user's information from misuse. Enck proposed Kirin [4], a detection tool to improve existing Android permission model. If any applications require permission it must be given before installation and cannot be modified after installation. The Android permission model has certain security threat also being user of mobile phone and application those who are not aware and least bothered with security mechanism they help the malware attacker to make misuse of these set of permission. Application can use combination of permission for stealing user's private and confidential information. To show the security implication of android apps

this paper proposes a malicious behavior analysis mechanism by combining static and dynamic analysis of android apps.

III. MALICUOIS ANALYSIS FRAMEWORK

There are basically two main approaches for malicious malware analysis for android app which includes static and dynamic. Static analysis is type of source code analysis of android apps. This is best way to know the source code and understand the behavior of application just by analyzing the programs source code. As it is totally based on the programs source code if we not able to generate the target code by reverse engineering or decompilation then it's very difficult ti analyze the source code. Dynamic analysis is keeping track on applications run time behavior when program is running. This method gives more accurate results compared to static. However dynamic method too has limitations in terms of covering the entire code. In this paper we present a combination of static and dynamic analysis that helps to overcome the limitations of each other. Fig 2. Will show the entire steps of execution.

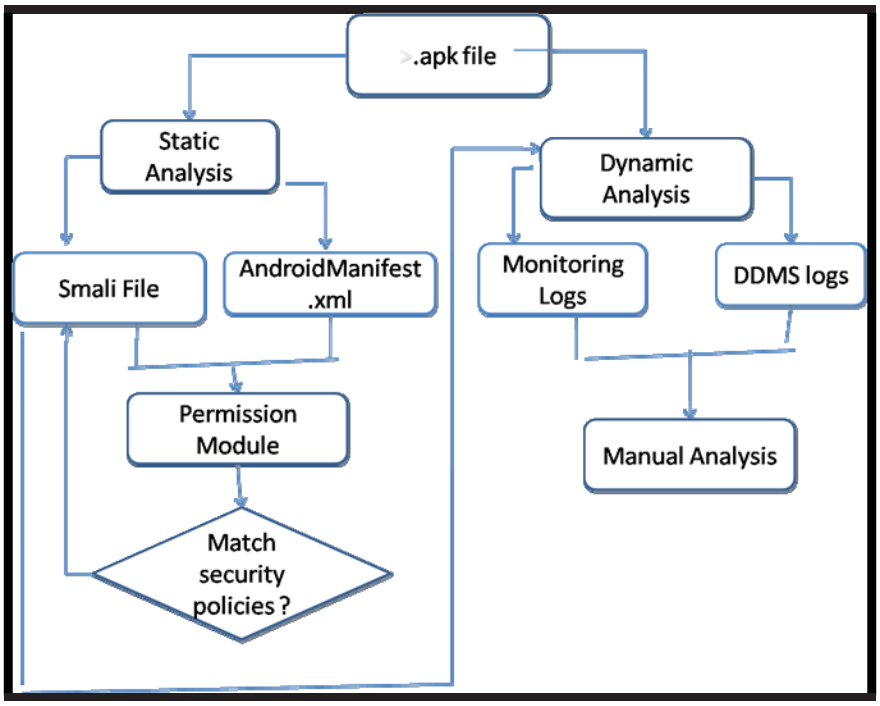


Fig 2. Android Malicious Behavior Analysis Framework

Prior to analyzing the android apps, we need to analyze android application package file by reverse engineering through static analysis to produce the configuration file i.e. smali code. The next configuration file is AndroidManifest.xml which is used for permission related purpose and the smali files is later on used for dynamic analysis. For doing this we selected the suspicious app which has more chances to leak user's private information. If program is found as suspicious then it taken to next stage of dynamic monitoring. In dynamic monitoring module the smali code and some tracking code is embedded and all these are repackaged and resign the .apk file. When apk is running we can dynamically view the behavior of apps privacy leakage and can notify the users about this malicious behavior. Theses alert are available in form of logs which can be further used for detailed analysis. After this we are going to discuss about three important component of this framework: apk de-compilation, permission module & dynamic module.

3.1 Decompilation

Before starting with permission and dynamic module we have extract the apps AndroidManifest.xml file and the equivalent smali [5] file for the given apk. The android apps are embed in into executable form with suffix .apk. It is same as .exe file available in our computer. This a.pk file after installation can be executed in any Android OS .apk file is compressed file which can be extracted with any .zip utility. After extracting the .zip file we can get resources,

permissions, the intermediate representation file called as smali file. In this paper we have used apktool [6] for decompilation.The File structure of any .apk file after extraction is shown below.

Table 1. File structure of .APK file

Directory/File	Description
Res	Application's resource file including pictures, sound video etc.
Smali	Dalvik register byte code files of apk
AndroidManifest.xml	The configuration file of apk including the package name, permission, referenced Libraries and other information
Apktool.yml	The configuration file of Apktool

3.2 Permission Module

There is some permission that may not create security risk but if they are combined with other set of permission can implement security risk. For example, application can demand for permission such as read phone state and sending messages it may cause security implication such as sending phones number or IMEI number out. In permission module we are implementing policies to find out whether an application uses risk permission in combination with other permission. In android there exist four different levels of permissions such as Normal, Dangerous, Signature, and SignatureOrSystem

Normal: It is least risk permission for any android app andit is granted automatically, without asking users.

Dangerous: It is high risk permission which can take userssensitive data. Dangerous permission is granted only by users of the application.

Signature; IT permission that is given by system only if theapplication is signed with same certificate as it declared in applications permission list.

SignatureOrSystem: It is granted only to those applicationsthat are in Android System image.

Out of these four permission types the main focus is on Dangerous permission as it allows accessing user's sensitive information. However there are two categories in which information can be leaked. First one is reading the privacy information such as android.permission.READ_PHONE_STATE, which allows reading SIM card number, phone number, IMEI number etc. The Second one is sending private information out. In this study we focus on two ways of leakage one is SMS and the other one is INERNET i.e. Android.permission.SEND_SMS and android.permission.INTERNET. Here we have used the security policies as combination of the two permissions i.e. READ permission and SEND permission. After decompilation we can get applications permission set from AndroidManifest.xml. Here to explain the permission module the permission matrix is prepared as combination of row and column where represents sending permission and column represents accessing the private data in this matrix we can explain how two permission can combine to send private information. Permission Model is explained in Table 2.In static analysis we used decompilation to extract the AndroidManifest.xml and with the help of this file we categorized the permission into read and send. We have assumed that value 1 means the apps is suspicious and at high risk. First row 1 indicates the app can send location information as SMS, second row "0" means that application does not send any information through calendar app neither send as SMS nor Over INTERNET. So there are no risk of this app as it does not combine with (READ_CALENDER, SEND_SMS) and (READ_CALENDER, SEND_INTERNET).

Table 2. Permission Model

Read Permission	Send Permission	
	SEND_SMS	INTERNET
ACCESS_FINE LOCATION	1	0
READ_CALENDER	0	0
READ_PHONE_STATE	0	1
READ_OWNER DATA	-	-
READ_SMS	-	-

3.3 Dynamic Module

In dynamic module we monitor the call messages f sensitive APIs in a.pk file. To do these we have used monitoring code which will be inserted into extracted source of android apps. The android app. is written in java after compilation it is converted into java bytecode. class file and it is executed by JVM(java Virtual Machine)and then it is converted in dalvik bytecode which is executed by DVM(Dalvik Virtual Machine).To monitor the suspicious code we have to again convert this dalvik byte code into java bytecode and rewrite the java code and finally convert this rewritten java byteccode into dalvik byte code. Sometime this approaches not work .to do this several tools are used such as dextojar[5] and ded[7] which convert dalvik byte code back to java bytecode. However this not error free conversion some information while converting into dalvik bytecode is missed. Sometimes thisconversion leads to invalid java byte code or invalid dalvik bytecode. There are assembler and disassemble known as smali and backsmali for the .dex file. i.e. classess.dex file. Smali code is intermediate e representation of dalvik bytecode. Smali file consist all related information about the .dex file i.e. (debug information, thread information etc.).As we have found that conversion from dalvik bytecode to smalicode is lossless so rather than JVM and DVM differences we have directly written into dalvik bytecode and insert the monitoring ode in smali code. The process of dynamic module is shown in Fig 3.

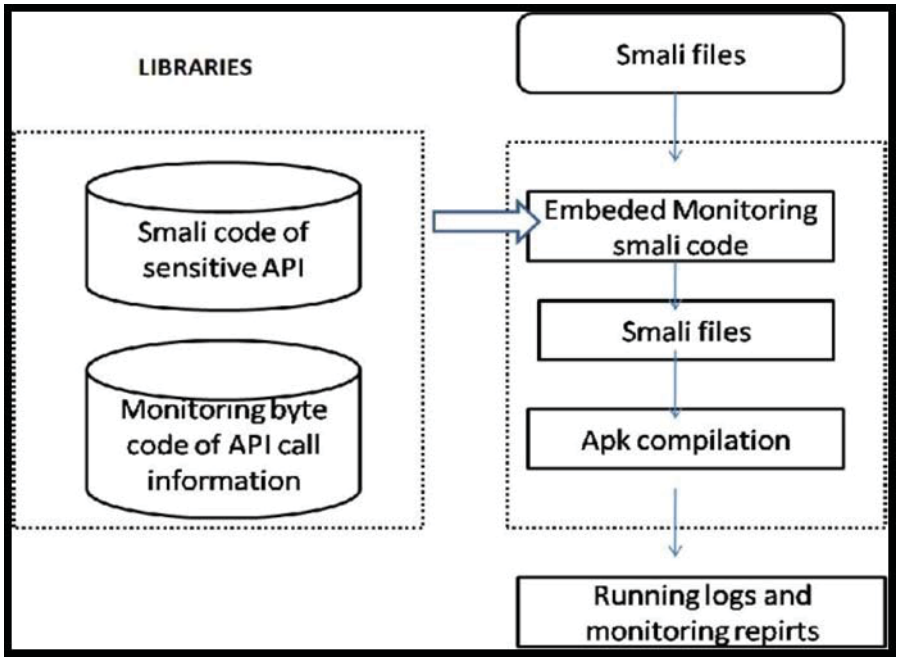


Fig 3. Dynamic Module Process

In Fig 3, we can obtain Smali files from the static decompiling. Then locate the concrete position of the sensitive API, and embed monitoring Smali bytecode to each different sensitive API. After that we use apktool to repackage the modified Smali bytecode to create a new APK and use the signature tool to re-sign it. Running the new APK on Android emulator, we are using log file to check the execution logs. It can generate a log on SD card which records the detailed call information of those sensitive APIs.

1) Smali bytecode [8]. Smali code is an intermediate code for dalvik bytecode.Smali is register based language. The smali language syntax is indicated in Table 3.
   Ljava/lang/String is equivalent to java.lang.String.Arrays can be written as int [] methods can be written as Lapackage/name/objectname:-
   >MethodName(III)Z which means Method name is name iof the method (III)Z is the prototype of the method.III indicates number of parameters and Z is return type of method.

2) Smali bytecode libraries for sensitive API.As smali bytecode keep sensitive API and their relevant smali bytecode. The aim for this library is to identify sensitive API in smali code after decompilation. In our study we have used some sensitive API and their description.

3) Monitoring bytecode: The monitoring bytecode library is used to store information about sensitive APIs call information when .apk is executing. Every API requires different ways of monitoring.

Table 3. Smali bytecode library for sensitive API

Class Name	**Function Name**	**Description**	**Smali bytecode**
Android.telephonysmsmanager	sendTextmessag E	Send messages	SmsManager; sendTextMessage[10]
android.location.LocationManager	getLastKnownLocation	GetLocation	LocationManager.getLast KnownLocation
Android.Telephony	getDeviceId()	Get ID,IMEI of phone	get- DeviceId()Ljava/ lang/String
android.location.	getSimSerialNu mber()	Get SIMserial Number	TelephonyMana ger get-SimSerialNumber
android.te lephony. Telephon yManager	getLine1Numbe r()	Get phone Number	Landroid/telephony/Telephon y Managerget-Line1Number()

IV. EXPERIMENT

Before experiment, some necessary tools such as Eclipse, jdk6, jre, android sdk other tools will be installed. apk decompiler and permission module were implemented in java. apk static compilation module uses apktool. Permission module mainly focuses security policies, extracting permission features from AndroidManifest.xml. The role of dynamic module is to scan and analyzing the smali code generated by static decompiler. Insert monitoring code in to smali, repackage and resign the apps smali code and then generate new .apk code for the same application. Then this apk is tested on Android emulator, while doing this generates the log file to show what has happened

4.1 Analysis of Android Markerts API

To show the problems faced by android users for information leakage we have selected some popular apps from known online android markets for our experiment. Some popular online android markets which we have used in our study are Google playstore, Aptoide,Amazon Appstore,Androlib.APK Samples collected from these popular app markets and these app have more than half android users. In this experiment we have focused mainly on users information such as LOCATION, SMS, CONATCTS, PHONE NUMBER, IMEI .Leakage meanly have sending message and INTERNET. We have used some random numbers of apps and collected permission detail of all those apps and found 26% apps have security implications of sending users private information.IN this showing users private information is considered as suspicious activity. After this we analyze the permission requested by .apk of applications. According to the security module proposed in the above section we count the number of apps equivalent to each type of privacy information. From the analysis, we observed that most security problem was with information leakage of IMEI number. As IMEI can find out phone type and device parameters and it can easily provide accurate users information for developer of apps and advertisers. The phone number, contacts & location theses information are used in illegal way .It causes huge loss to users.

Table 4. Analysis of suspicious Apps

Market	**App Number**	**Suspicious number**	**Ratio%**
shouji:com:cn	5	2	40
appchina:com	5	2	40
market:goapk:com	5	1	20
eoemarket:com	5	1	20
Total	20	8	40

.

4.2 Sensitive API monitoring

To check the efficiency and practicability of dynamic module we checked on Android emulator in Windows 7 operating system. The monitoring result generated after testing is a text file and the output is stored in SD card of Android emulator. To find different sensitive API we have to find out the set of registers the return value and then use a call to various log modules to find out record when applications are running. For finding out the send text message api the send text message smali code is shown in Fig 5. .From figure we can see that the send text message function has five parameters out of which only two parameter are important i.e. Receiver number and message. From smali code we can see that register v1 and v3 stores these two parameter (receiver number, message) and remaining three parameters are null. So we need to pass only v1 and v3 to log function to record the information related to message information when applications are executing .After inserting the monitoring code into smali file we used apktool to repack the modified smali file. The above experiments show that the dynamic module is successfully tested on smali file of apk file. The log file says detail information about sensitive api. After dynamic module once it is confirmed that app is suspicious we can use advanced tool such as DDMS to analyze

more deeply.

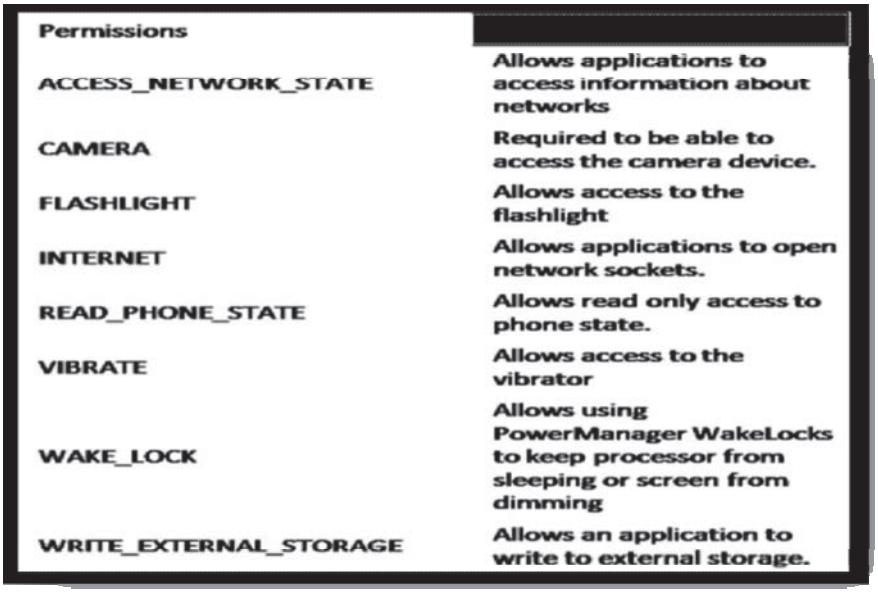
```
iget-object v1, p0,
Lcom/example/sendsms_example/SendSMS_ExampleMainActivi
ty;
->phoneNumber:Ljava/lang/String;
iget-object v3, p0,
Lcom/example/sendsms_example/SendSMS_ExampleMainActivi
ty;
->SMSContext:Ljava/lang/String;
move-object v4, v2
move-object v5, v2
invoke-virtual/range {v0 .. v5},
Landroid/telephony/SmsManager;
-
>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/la
ng/
String;Landroid/app/PendingIntent;Landroid/app/PendingIntent
;)V
```

Fig 5. Smali Code for Sendtextmessage

THERE IS EXAMPLE OF .APK FILE WHICH MALWARE APP SHOWSSUSPICIOUS ACTIVITY. THIS IS TESTED BY APKSCAN MALWAREANALYSIS TOOL. WHICH GIVES SOME INFORMATION ABOUT THIS APPSMISUSES THE PRIVATE DATA WITHOUT THE CONSENT OF USER.
NVISO ApkScan malware analysis report May 12, 2016 SuspiciousActivity Detected [9]

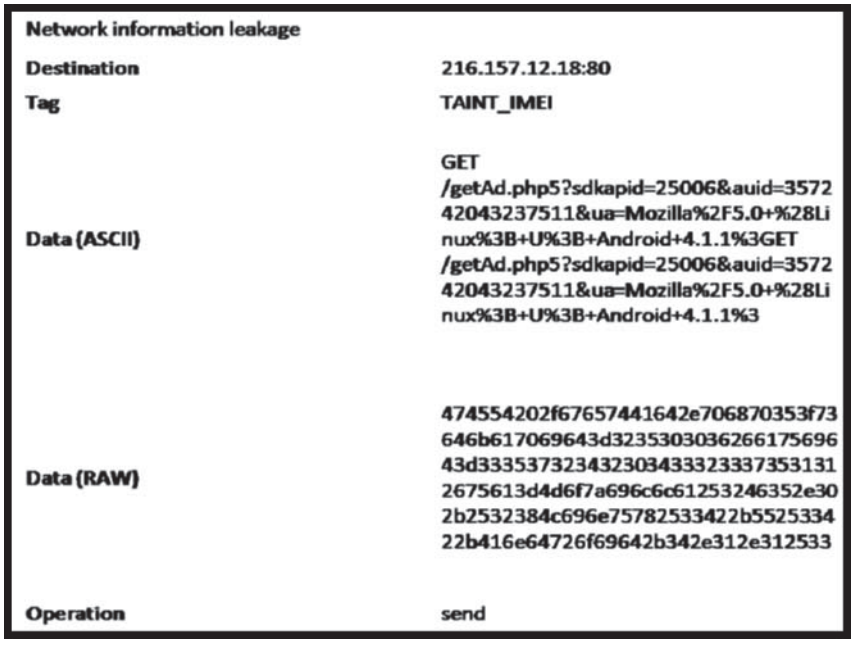
General information	
File name	Flashlight.apk
Other known file names	None
Origin	Manually uploaded by anonymous user [2016-02-20 17:28:39]
MD5 hash	dad9ddb623ed1f68de4be0434454ce6e
SHA256 hash	d5db441e1dc450a8307746915bd92577e1dff5535baf282e11e7eda6e1227ea6
File size	1628.21 KB
Worker	NVISO_API_KALI_01

Fig 6. Basic Information about app



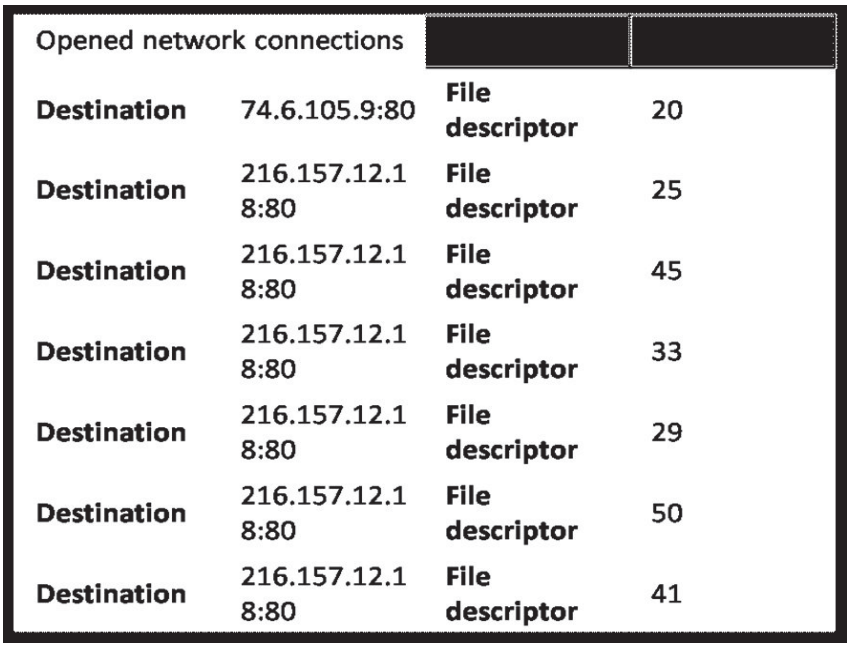
Permissions	
ACCESS_NETWORK_STATE	Allows applications to access information about networks
CAMERA	Required to be able to access the camera device.
FLASHLIGHT	Allows access to the flashlight
INTERNET	Allows applications to open network sockets.
READ_PHONE_STATE	Allows read only access to phone state.
VIBRATE	Allows access to the vibrator
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage.

Fig 7. Static malware analysis



Network information leakage	
Destination	216.157.12.18:80
Tag	TAINT_IMEI
Data (ASCII)	GET /getAd.php5?sdkapid=25006&auid=357242043237511&ua=Mozilla%2F5.0+%28Linux%3B+U%3B+Android+4.1.1%3GET /getAd.php5?sdkapid=25006&auid=357242043237511&ua=Mozilla%2F5.0+%28Linux%3B+U%3B+Android+4.1.1%3
Data (RAW)	474554202f67657441642e706870353f73646b617069643d32353030362661756964333537323432303433323337353131267561 3d4d6f7a696c6c61253246352e302b2532384c696e75782533422b552533422b416e64726f69642b342e312e312533
Operation	send

Fig 8. Network Activity



Opened network connections			
Destination	74.6.105.9:80	File descriptor	20
Destination	216.157.12.18:80	File descriptor	25
Destination	216.157.12.18:80	File descriptor	45
Destination	216.157.12.18:80	File descriptor	33
Destination	216.157.12.18:80	File descriptor	29
Destination	216.157.12.18:80	File descriptor	50
Destination	216.157.12.18:80	File descriptor	41

Fig 9. Opened network connections

V. CONCLUSION

In this paper malicious Android application detection system was Fig 7. Static malware analysis proposed. We have used the permission model to and combination of permission to identify the potential threats in applications. And the suspicious apps are forwarded to dynamic monitoring tool to keep track the call information of sensitive API when application s are running. To conclude this we have observed some beneficial approach

1) The smali byte code which the result of decompilation and it is intermediate code which help in knowing the obfuscation as it is not affected with any code obfuscation issues.

2) As it seems to be very simple by just inserting some monitoring smali bytecode without worrying of performance issues.

3) This method can also be used in remote conditions also at large scale and to implement monitoring service automatically.

The research moves ahead in direction where more sensitive api can be discussed and more real solutions to many apps can be given be developing real time apps to do this. Some more methods can be combined for better result.

REFERENCES

[1] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in Proc. 18th ACM Conf. Computer. Computer security, 2011, pp. 627–638.

[2] Google, Android Home Page, 2009. (http://www. android.com)

[3] D. Bornstein, Dalvik VM Internals, 2008. (https://sites.google.com/site/io/dalvik-minternals).

[4] W. Enck, M. Ongtang, and P. McDaniel, “Onlightweight mobile phone application certification," in Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235-245, Chicago, USA, Nov. 2009.

[5] Google, Dex2jar: Tools to Work with Android .dexand java .classses, 2013. (http://code. google. com/p dex2jar/)

[6] D. Reynaud, D. Song, T. Magrino, E. Wu, andR. Shin, “Freemarket:shopping for free in android applications," in 19th Annual Network & Distributed System Security Symposium, Hilton San Diego, USA,Feb. 2012.

[7] D. Octeau, W. Enck, and P. McDaniel, the DED Decompiler, 2011. (http://siis. cse. psu. edu/ ded/ papers/ NAS-TR-0140-2010.pdf)

[8] Google, Smali, July 11, 2015. (http://code. google.com/ p/ smali/)

[9] Android apk scan report. (https://apkscan.nviso.be)

[10] Android Malware and Analysis - Page 72 - Google Books result,”https://books.google.co.in/books?isbn=148225220.