



7th International Conference on Communication, Computing and Virtualization 2016

Analysis of Malicious Behavior of Android Apps

Pooja Singh, Pankaj Tiwari, Dr. Santosh Singh

*AMET Univeristy, Chennai
JIT University ,Rajasthan*

Abstract

As increasing in number of Android phones there is simultaneous increase in mobile malware apps which performs malicious activities such as misusing user's private information as sending messages i.e. SMS, reading users contact information and can harm user by exploiting the user's confidential data which is stored in mobile. Malware are speeded not only infecting the user's data but also harming several organizations in term of stealing of private and confidential data. Hence Malware classification and identification is a critical issue. Android users are unaware about several apps which they are using whether they are malware infected or not. Android applications require the concept of permission mechanism to show that apps are using certain permissions to get access to information from your device. Android apps which are installed in the smart phones get access to all the required permission during installation of apps. Google assure their customer in terms of security about the apps which are available to download from there play store. Android operating system is open system and it allows users to install any applications downloaded from any unsafe site. However permission mechanism is still very diminutive defense mechanism to assure that the applications can harm to user. Therefore in this paper we propose the Malware characterization from manifest file and allows user to improve the efficiency of Android permission to inform user about the risk of Android permission and apps.

© 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Peer-review under responsibility of the Organizing Committee of ICCCV 2016

Keywords: Andoid Security,permissions,maware apps

Introduction

As increase in development of internet, the mobile internet is need of hour. As with the popularity and the fast development of Android OS, the security of Android OS is major challenge. As Android is very popular mobile OS hence its demand is increasing day by day with the increase in the challenge to safeguard an apps installed on users device which is connected to world of internet most of the time. These devices are more prone to the attack by the apps installed on their device called as malware. Report says that Android OS is first choice for the malicious software designers to attack Android OS. The main purpose of this paper is to analyze the Android apps properly and using static and dynamic analysis methods to show the behavior of apps leaking users private information.

Privacy leakage in this paper means the stealing of user's confidential information by getting the unauthorized access to resources during installation such as Device ID, contacts, call records, location information etc. and send these information through messages or network. As if now the methods of finding users data privacy and leakage of information in Smartphone has mainly two types, static and dynamic. Static analysis means to focus on control flow, data flow and structural analysis [3]. As Androids major part is written in java programming language which has large number of function calls static analysis is not as effective as dynamic. As compared to static, dynamic analysis is concerned about sandboxing and dynamic taint tracking method. Sandboxing is the way of isolating android OS. Some background researches on Android security are mentioned as follows:

1. Kui Luo proposed a byte code converter, converting DVM code into Java [6].
2. Leonid Batyuk projected a method by decompiling sample applications not touching the core function, Although International Journal of Network Security, Vol.18, No.1, PP.182-192, Jan. 2016 183 this method can analyze the sample malware code effectively, it is unsatisfactory when the target program has been obfuscated.
3. Enck implemented a Dalvik decompiler, DED, by using the static analysis package tool. The tool use Fortify SCA to analyze the application's control flow, dataflow, structure and semantics [7].
4. Qian et al also depends on Dalvik decompiling and gives a basic two-step framework for Android malware behavior monitoring [8].
5. ComDroid analyzes the DEX byte code disassembled by Dedexer, and checks the Intent creation and transmission to identify the program broadcast hijacking vulnerabilities [9].
6. ScanDroid extracts the security specification from configuration files of Android application and checks the consistency between the application dataflow and the specification [10]. ScanDroid is based on the WALA analysis framework, can only evaluate the open source applications.

1.1. Android basic Architecture

Android operating system is designed on basis of Linux kernel and is developed by the Google [4]. Android has a layered architecture, including the Linux kernel layer, middle layer and application layer, which can provide consistent services for the upper layer, masks the differences of the current layer and lower layer [5]. The middle layer of android performs center functions which can be implemented by programming languages like JAVA/C/C++. Most of the applications running on Android are written in Java programming language, and then these multiple java class files are converted to dex format by the Android dextojar tool. Each Android application is an independent instance to run in DVM, and has a unique identification number known as PID. Figure 1 gives brief information about architecture of an Android operating system. Dalvik Virtual Machine (DVM) [4] is the main component of Android platform as compared to other components of the Android. It fully supports all Java applications which are converted to dex (Dalvik Executable) format. The dex format is compressed format of Dalvik executable code, which is suitable for memory and processor speed. Dalvik code is accountable for process segregation and thread management. Each Android application corresponds to a separate instance of Dalvik virtual machine, and can be executed in virtual machine.

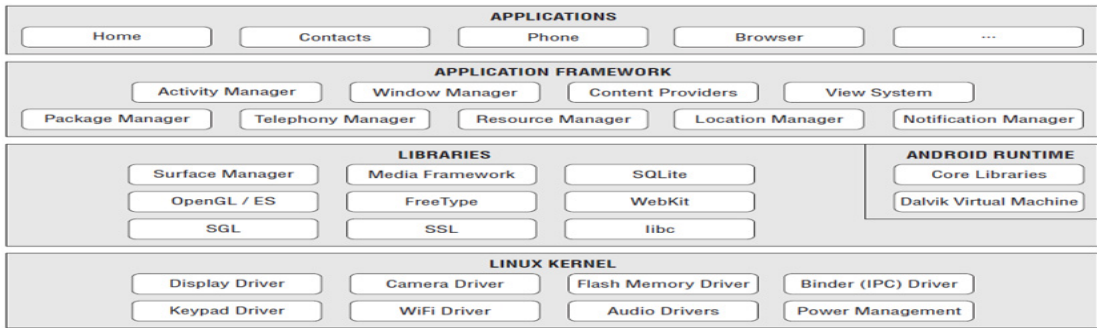


Fig. 1 Android Architecture

1.2. Android security issue

Android security model is similar to Linux as it is designed on Linux kernel [4]. The main part of Android security model mainly includes the sandbox, application signature and permission mechanism. The permission mechanism limits applications to access user's private data (i.e. telephone numbers, contacts etc.), resources (i.e. log files) and system interface (i.e. Internet, GPS etc.). In permission mechanism, the phone's resources are organized by different categories, and each category corresponds to one kind of accessed resource.

If an application requires access to certain resources, it needs to have the corresponding permissions. Although this mechanism is simple, it also has some defects that cannot protect the user's private information adequately. Some researchers, Ontang et al questioned Android security model, and pointed out that the current Android permissions model cannot meet certain security requirements. Enck proposed Kirin [10], a detection tool, to enhance existing Android permissions model.

Based on a set of policy, Kirin has used to determine whether to grant the requested permissions to applications and through the analysis of the Android application's Manifest file to ensure the granted permission in accordance with system strategy. Android permissions mechanism is coarse-grained [4].

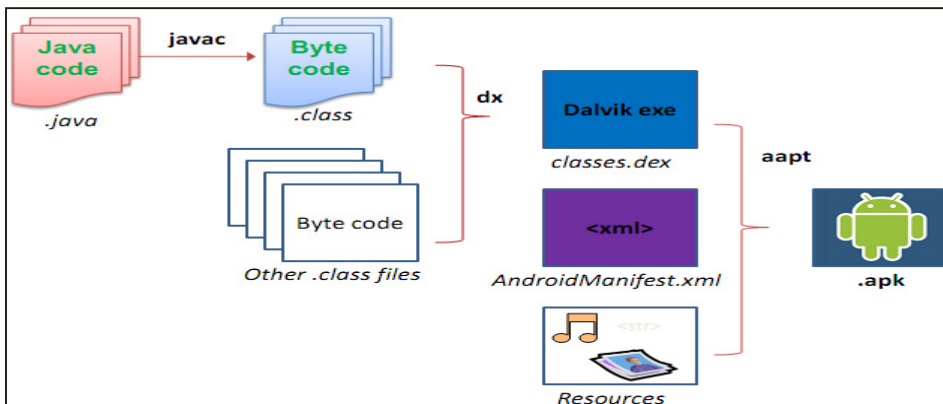


Fig 2. APK conversion steps

The application required permissions must be granted all before installed and cannot be changed after installation. This permission model leads to certain potential security threats. On the one hand, permissions to access private data will be decided by users. For those non-security awareness users, the permission granting process is casual and blind. During the installation phase, if the program obtains permissions to access privacy information, then can be arbitrary abuse of user's privacy and sensitive data at any time. On the other hand, the mechanism cannot effectively prevent permission escalation attacks. Applications can take advantage of a combination of permissions to steal the user's sensitive data. In order to reveal Android apps leaking user privacy information behavior, according to the

Android OS security mechanism, this paper proposed a malicious behavior analysis model combining the dynamic and static method, which will be discussed in detail in the next sections.

2. Android Applications Malicious Behavior Analysis

2.1. Analyzing Behaviors of Android apps

Different methods for doing malware analysis are: static and dynamic approach. Static analysis is based on program's source code. It has maximum advantage of exposure and it helps to analyze the source code broadly. However static method is based on source code. And if we cannot get the target source code, through decompiling or reverse engineering, it is hard to analyze the program accurately, especially in the occasion that the target program has been malicious File.

Dynamic analysis points to the exploration of run-time performance by running the program. This type of method is precise for finding the actual malicious code behavior. Meanwhile, the dynamic method has its own disadvantages because of its inadequate execution exposure, that is to say we cannot guarantee all of the running paths have been triggered during the test. In this paper, we present a combination of static and dynamic security analysis model that can make up for their shortcomings with each other, performs the analysis of malicious performance more widely and correctly. Fig. 2 explains the whole steps. Before analyzing the Android application, APK (android application package) needs to be decompiled to get the corresponding configuration and Smali [11] files. Among them, the configuration file with the format of AndroidManifest.xml is mainly used for permissions filtering stage, and the Smali files are mainly applied to dynamic monitoring module. First of all, we choose those suspicious applications with great potential to leak user's privacy. Then if a program is suspicious, enter into the dynamic monitoring module, where input the target Smali codes, embed some tracking code, repackage and re-sign the APK. In future, once the APK is running, we can dynamically monitor the behavior of privacy leakage and give immediate alarm for users. And those alerts or logs can be used for further detailed analysis manually or automatically. Next, we will discuss the three core components of the framework: APK Static Decompiler, Permission Filtering Module and Dynamic Monitoring Module.

2.2 APK Static De-compilation

Before permission filtering and dynamic monitoring, we need to extract the Android application's AndroidManifest.xml file and smali files equivalent to the target APK. The Android application is an installation package ended with suffix.apk (an acronym for Android Package). APK is same as to .exe (executable) file in computer, after installed can be executed in Android OS immediately. APK is actually a compressed file compliance with the ZIP format, which can be extracted by popular .zip compatible decompression tools. In addition, it must be noted that most applications are code-obfuscated, and the unzipped file is not able to analyze directly. It should be decompiled to extract its resource, permissions, and the intermediate representation files. In this paper the apktool[12] is used for decompiling. The file structure of .APK file in Android apps after decompilation is shown in Table 1.

Table 1: The file structure after APK decompiled

Directory/File	Description
res	Application's resource file, including pictures, sound, video and etc.
smali	Dalvik register bytecode files of APK
AndroidManifest.xml	The global configuration file of APK including the package name, permissions, referenced libraries and other related information of the application.
Apktool.yml	The configuration file of Apktool

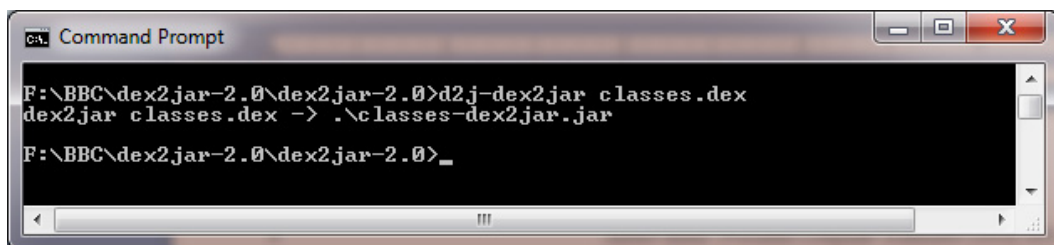
Fig 2. APK to .Class conversion steps

2.3 Permission Module

There is some permission that may not exist risks by itself, but combination of permissions may exist a security risk. For example, an application applies for permissions to read phone state and sending messages, and there may exist risk of transfer the phone number or IMEI out. Permissions module is based on a set of security policies to find out whether an application has some special risk permission combinations. For all the Android permissions, there are four types of security levels. Those are Normal, Dangerous, Signature and SignatureOrSystem.

2.4 Dynamic Monitoring Module

We apply real-time monitoring by inserting monitoring code to the decompiled APK. The Android developers write the application in Java, compiles it into Java byte code, and finally transfers to the Dalvik byte code which can be executed in DVM. So it easy to do reverse engineering by converting the Dalvik byte code to Java byte code, then rewrite the Java byte code, and finally convert the rewritten Java byte code back to Dalvik byte code. Still, this type of method does not work all the time. There are quite a few significant differences between Java Virtual Machine and Dalvik Virtual Machine. JVM is based on stack while DVM is based on register. A number of tools are available, such as dex2jar [11] and ded [7], which will convert Dalvik byte code to Java byte code. Still this is not a lossless converting; some information from the Java byte code is lost when being converted to Dalvik. These tools try to gather the lost details based on the context, but at times the inference is defective. Even while these errors does not prevent static analysis on the converted Java byte code, in our experience they often lead to invalid Java byte code or later invalid Dalvik bytecode. After we convert an application's Dalvik byte code to Java byte code (e.g. dex2jar) and then back to Dalvik bytecode, the resulting application does not run properly. So the possible way is to directly use the Dalvik byte code. Smali and baksmali are two assembler and disassembler respectively for the dex format used by the DVM. Smali is an intermediary depiction of Dalvik byte code. Smali can fully realized all the features of dex format (annotations, debug information, thread information, etc.). Moreover, dex and Smali can convert lossless between each other. In this paper we try to directly rewrite Dalvik byte code, insert the monitoring Smali bytecode into the decompiled Smali files. The process of dynamic monitoring method is shown in Figure 4. In Figure 4, we can obtain Smali files from the static decompiling. Then locate the concrete position of the sensitive API, and insert monitoring Smali byte code to each dissimilar susceptible API. After that we use apktool to repackage the modified Smali byte code to create a new .APK and use the signature tool to re-sign it. Running the new APK on Android emulator, we can use logcat to view the runtime logs. It can generate a log on SD card which records the detailed call information.

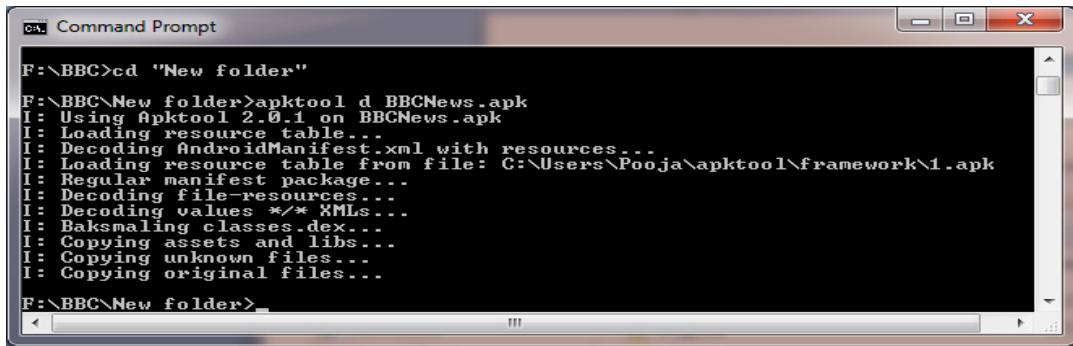


```

C:\> Command Prompt
F:\BBC\dex2jar-2.0\dex2jar-2.0>d2j-dex2jar classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
F:\BBC\dex2jar-2.0\dex2jar-2.0>_

```

Fig 3. Converting .dex to .class of Android app BBCNews.Apk



```

C:\> Command Prompt
F:\BBC>cd "New folder"
F:\BBC\New folder>apktool d BBCNews.apk
I: Using Apktool 2.0.1 on BBCNews.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Pooja\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
F:\BBC\New folder>

```

Fig 4. Decoding smali code and AndroidManifest.xml of Android app BBCNews.apk

3. Conclusion

In this paper we proposed a two-step analysis of Android apps: static and dynamic. At the initial stage the permission related issues of applications are highlighted. The permission mechanism of Android OS provides the facility to access all the required details during installation and once accepted the permission can't be modified. To stop this we use two mechanism i.e. static and dynamic methods to analyze the behavior of to find apps are malicious or not. We use reverse engineering tools through which the .APK file is converted to .Class file and the code is analyzed and changes are made and then it repackaged again using Smali byte code which is an intermediate code of APK file. This smali code is inserted into process for monitoring purpose. This method can be used at wide scale for monitoring service automatically. Further research will continue for monitoring the sensitive API in Android and explore the vulnerability of the apps. At the same time to provide more dynamic analysis methods to conduct more research.

References

1. D. Bornstein, Dalvik VM Internals, 2008.(<https://sites.google.com/site/io/dalvik-vminternals>)
2. http://onlinepresent.org/proceedings/vol35_2013/13.pdf
3. P. Hornyack, S. Han, J. Jung, S. schechter, and D.Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in Proceedings of the 18th ACM Conference on Computer and Communications
4. Google, Android Security and Permissions, 2013.(<http://d.android.com/guide/topics/security.html>)
5. Google, Android Home Page, 2009. (<http://www.android.com>)
6. K. Luo, "Using static analysis on android applications to identify private information leaks," Master Dissertation of Kansas State University, 2011.
7. D. Oceau, W. Enck, and P. McDaniel, the DED Decompiler, 2011. (<http://siis.cse.psu.edu/ded/papers/NAS-TR-0140-2010.pdf>)
8. Q. Qian, J. Cai, and R. Zhang, "Android malicious behavior detection based on sensitive api monitoring," in 2nd International Workshop on Security, pp. 54{57, Nov. 2013.
9. E. Chin, A. P. Felt, K. Greenwood, and D. Wagner," Analyzing inter-application communication in android," in Proceedings of the 9th International Conference on Mobile Systems, Applications and Service, pp. 239{252, Washington, USA, June 2011.
10. W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235{245, Chicago, USA, Nov. 2009.
11. Google, Dex2jar: Tools to Work with Android .dex and java .classes, 2013. (<http://code.google.com/p/dex2jar/>)
12. D. Reynaud, D. Song, T. Magrino, E. Wu, and R. Shin, "Freemarket: shopping for free in android applications," in 19th Annual Network & Distributed System Security Symposium, Hilton San Diego, USA, Feb. 2012.