Hacking your Droid

ADITYA GUPTA

adityagupta1991 [at] gmail [dot] com

facebook[dot]com/aditya1391

Twitter : @adi1391

INTRODUCTION

After the recent developments in the smart phones, they are no longer used just to make calls and send sms's. They have been endowed with much more features than what used to be in them 5 years back. They can now take pictures, make video calls, track locations, do banking, and many more things.

With all these features, there also comes a need for security for the mobile phones. No one wants their personal information to be seen by anyone else.

Just like the Operating Systems for computers, the smartphones these days also come with an Operating System to perform advance tasks.

In this paper, I would be focussing on the Android OS, and discussing about its security, and existing malwares on this platform.

According to Google, Android is a software stack consisting of an OS, middleware and software applications. It is an open source platform developed and maintained by Google and the OHA(Open Handset Alliance) which is a consortium of over 50 different international electronics and telecom companies.

It has also got a huge user and developer community base, mainly because of its open nature. Any developer can publish his/her app on the Android Market with a self signed certificate. This makes it easier for the developers as there is no intermediate CA involved. The official application store for Android is the Android Market, which consists of over 4 billion apps.

Android is supported by a Linux Kernel 2.6.x at its core. The native libraries are in C/C++ whereas the applications are written exclusively in Java, with the layout designed in either XML or Java.

Shown above is the Android Architecture. At the bottom core, is the linux kernel, which has been modified specially for the mobile platform. It is required for the core system services such as security, process management, memory management etc.

Above it, we have the libraries, which includes WebKit, SQLite (a powerful and lightweight database for the mobile), and OpenGL and so on. The libraries are written in C and C++ and are used by various components of the Android System.

The Android Runtime includes DVM (Dalvik Virtual Machine) and Core Libraries. The DVM is a virtual machine to run the applications. We will be looking more about the Dalvik Virtual Machine in the coming pages.

Above this, we have the Application Framework and Applications Layer. The Application Framework allows the developers to take advantage of various inbuilt functionalities of the

Android Platform. All the developers have access to the same framework API that is used to build the default applications. So, developers can access each and every feature supported by the phone.

The Applications layer consists of the default applications present in-built into the phone. These include Phone, Browser, Contacts, Home and so on.

Android comes with a Android SDK which is a Development kit made for developers to create applications for the android platform. The Android also consists of an emulator and the other tools, along with the libraries necessary for the development. Emulator is a virtual phone, which you can run on your computer, to test the developed applications.

ANDROID APPLICATIONS

Android Applications or Android apk's are basically an archive file containing all the necessary files and folders in an application. An Android application is made up of many components namely Activities, Intents, Services, Broadcast Receivers and Content Providers. We'll be discussing each of them one by one in this paper.

ACTIVITY

Activities are any visual screens you see and interact with in an android application. It can consist of views such as the Button View, Text view, Table View etc.

Each application has a “main” activity which is the first screen presented to the user as soon as he starts the application.

The Android activities follow an activity lifecycle. So, whenever an activity of an application goes to the background, it doesn’t exist, instead it just goes to an *OnPause* state.

Also, one interesting thing to note here is that, Activity of one application can start the activity of another application, if it has the appropriate permissions.

SERVICES

Apart from the components with which you interact, there are also components in an application, which work in the background, for example a song playing in the background. For those things which need to continue even when they are not in focus, we need services. They are the components which work in the background and keep performing the necessary operations without any user interaction.

Suppose, you launch a music player application. The first screen, with which you are interacting with, is an activity. But as soon as you select a song to play, and move on to some other application, the activity goes to a pause state, but the service (the song which is playing) keeps running in the background.

Some more examples of services are network operations, file input/output etc.

CONTENT PROVIDERS

Every application needs to store its data, be it in the SQLite database or some place in the SD card or in the phone itself. For storing and retrieving the data, we need to use Content Providers.

BROADCAST RECEIVERS

As the name suggests, they receive broadcast signals made by other applications or by the phone itself and perform the action whatever they are assigned.

INTENTS

Intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService (Intent) or bindService (Intent,

ServiceConnection, int) to communicate with a background Service

Intents are used by the application to bring all the other components together. It is used by an application to interact with the phone's hardware components and other applications, or to start a service or activity with the help of broadcast receiver. Intents can also be used to call the activities of another application from one application.

Android Manifest

All the android applications must contain a file named as AndroidManifest.xml. The xml file should be containing the following necessary things:

1. All the hardware and software permissions needed by the application.
2. The external API's if required by the application (ex – Google Maps API)
3. The minimum Android Version needed to run the program

So, if we want to access any of the features of the phone, we have to clearly state it in the Android Manifest file. These permissions will be shown to the user, when he/she would be installing the application.

A sample AndroidManifest.xml file :

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.aditya.com"

    android:versionCode="1"
```

```
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".aditya"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
<uses-permission android:name="android.permission.SEND_SMS"></uses-permission>

    </application>
</manifest>
```

In the above application, one can clearly see that, I'm trying to access the Send SMS feature of the phone which is stated as the permission android.permission.SEND_SMS.

The AndroidManifest file also helps a user in determining whether an application is a legitimate one or it is of malicious intentions. For example, a game application may need INTERNET permission to upload the high scores to the online server, but in no case, it should need permissions such as SEND_SMS, READ_CONTACTS or anything like that.

ANDROID SECURITY MODEL

Android puts the user in control of everything. The Android Security model is based on the fact that each application is to be run within its own DVM, which acts as a sandbox between two applications. So that, even if one of the application crashes, it won't affect the other application or the phone. This makes sure that the two applications are independent of each other; also the data of one application cannot be accessed by another application without sufficient permissions.

Dalvik Virtual Machine was created by Dan Borstein, specifically for the android platform, so that applications could run smoothly under low power processors with graphics and hardware acceleration suitable for the mobile device.

The DVM is *register based* instead of the standard Stack Based JVM (Java Virtual Machine). Due to this, java files of the android applications, are compiled to a Dalvik byte codes or dex (Dalvik Executables) before being packaged into the apk.

Also, if there are more than one java files, all are compiled into a single dex file to save space and memory.

When installed, each application is assigned a unique UID and GID, just like what is there in the Linux file system. Each app's UID has to be different from another application. Also, applications from a single developer could share the same UID, so that it is easy to push updates for the application, without having the user to uninstall the old version and install the new one.

REVERSE ENGINEERING

Reverse Engineering is a process in which we decompile an application to understand its working and functionality better, by analyzing the codes and debugging it.

The reverse engineer of an Android Application, is not much different from the reverse engineering of computer software.

Before reversing, let's understand, how an APK is made

Now, in real world scenario, we have the apk file in our phone. So, first of all we would be depackaging it, using any standard, unzip file, such as Winrar or 7-zip.

If we're on Linux, a standard command "unzip app2test.apk" would be uncompressing the apk to get the following files and folders.

1. Meta-Inf
2. Res
3. classes.dex
4. AndroidManifest.xml
5. Resources.arsc

At this point of time, our main point of concern would be the *classes.dex* file, which is the compiled java classes and contains all the codes of the things to be performed by the application.

So, our main aim would be to somehow decompile the classes.dex file into human readable codes.

There are two ways we can proceed now. The first one being, converting the classes.dex into smali format, which is similar to java file format and the other method, is to decompile it to JAR file and then opening it using a jar decompiler such as JD-GUI.

Method I : Converting it to smali format

Smali is a file format having a structure similar to Jasmine.

We can convert *classes.dex* to *smali* formats using a tool name as *Baksmali.*

Or we can use another tool name as *APKTool* made by __ which is an overall tool, which could be used to both decompile and compile an android application.

Lets in our case, use APKTool, as it contains the feature of converting the classes.dex file to Smali format codes, along with some other nice features.

According to *Brut.all*, the creator of APKTool, it is a tool to reengineer closed binary android apps. It also makes us easier to analyse/debug the smali codes step by step and read the AndroidManifest.xml file.

```
apktool d Settings.apk
```

Here the d stands for decoding in debug mode.

```
r00t@hax0r:~$ apktool d Malware.apk

I: Baksmaling...

I: Loading resource table...

I: Loaded.

I: Loading resource table from file: /home/r00t/apktool/framework/1.apk

I: Loaded.

I: Decoding file-resources...

I: Decoding values*/* XMLs...

I: Done.

I: Copying assets and libs...
```

We just decompiled an APK into readable smali format for debugging

The debugged files and folders would be saved to a folder named as the name of the apk, in this case, in a folder named "Malware"

If you wish to save it in a desired folder, use

```
r00t@hax0r:~$ apktool d Malware.apk Outputfolder/
```

The decompiled smali files would be saved to a folder named as "Smali" .

The whole list of commands can be brought up by

```
apktool
```

After we've made necessary modifications with the apk, If we have to recompile the apk, we would be using the build functionality of apktool

```
r00t@hax0r:~$ sudo apktool b Malware/
```

The newly generated apk would be located in a folder named as "build".

2. Converting it to Java files :

Apart from converting to smali format and analysing the codes, we could also convert the classes.dex file to the original java classes. However the point to note here is that, by converting to java classes, we would be having a little bit modified code and in some cases, we may even not be getting some of the parts of the codes.

The first step in converting the file to java format is to first convert it to JAR file. To do this, we have a wonderful free utility named as Dex2Jar.

First of all, we need to have the classes.dex file of the apk. To get it, just extract the apk.

```
r00t@hax0r:~$ unzip malware.apk
```

The extracted files would be having our required classes.dex file.

Also, in some of the cases, the AndroidManifest.xml file upon extracting won't be readable. In that case, you could use the aapt tool present in the Android SDK.

```
r00t@hax0r:~$ aapt xmltree d AndroidManifest.xml malware.apk
```

To convert to Java, first we have to convert it to JAR format using Dex2jar.

From the JAR file, we could get the original java files using a tool named as JD-GUI.

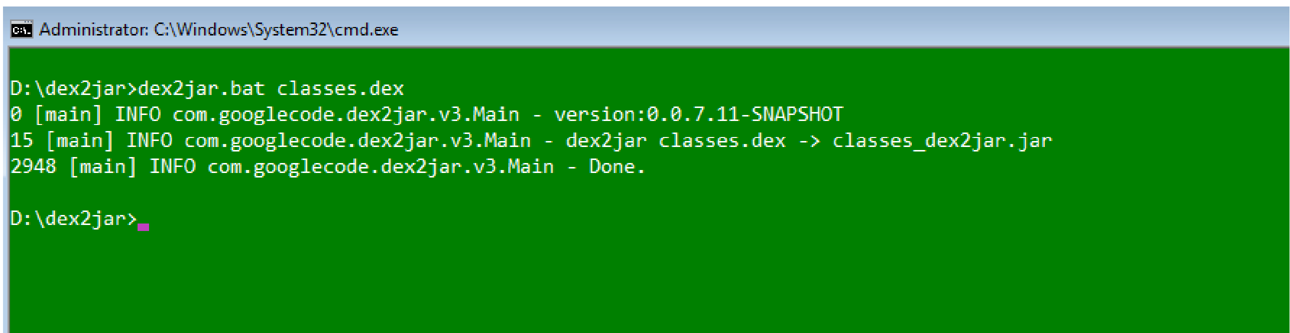
r00t@hax0r:~/dex2jar$ ./dex2jar.sh

Before using the above command, you've to set the class path if you're using Linux.

If you're on windows, you can use the command directly.

Like,

dex2jar.bat classes.dex

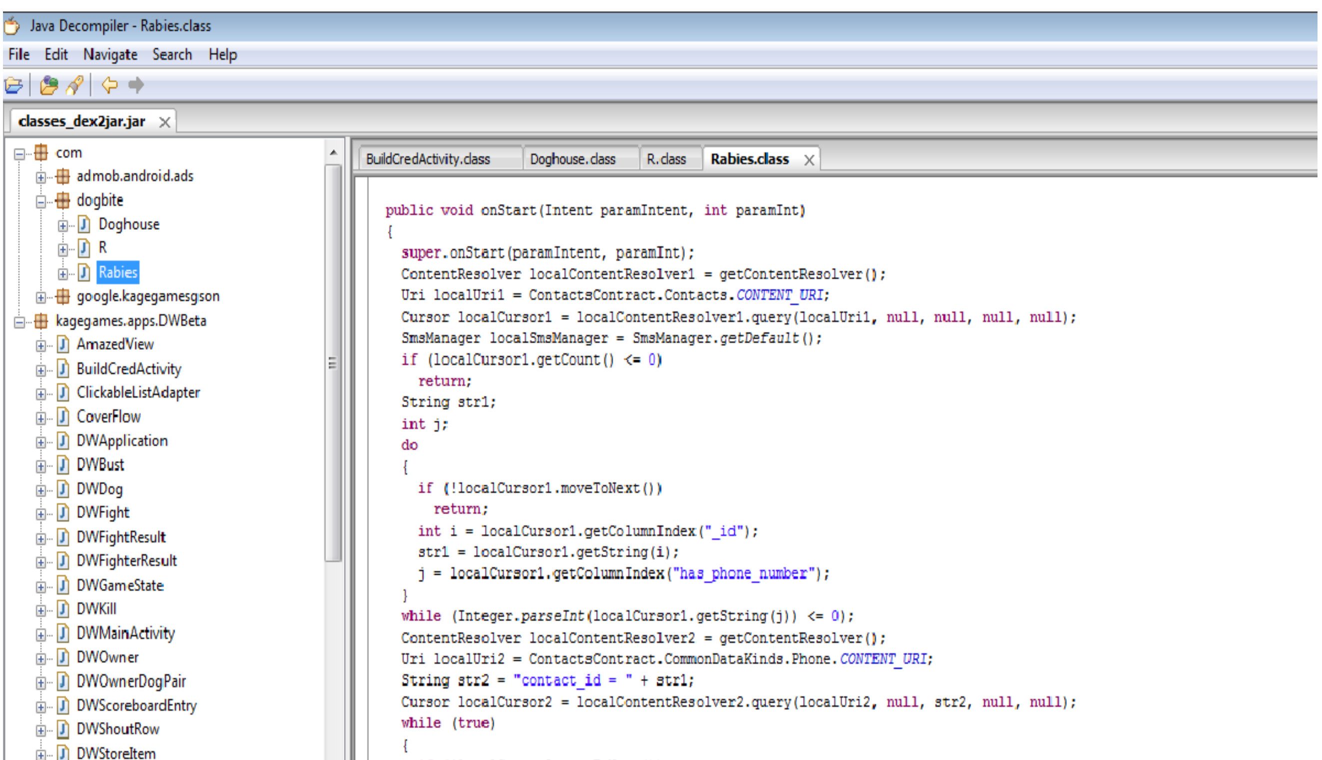


After conversion, you'll be having a jar file named as classes_dex2jar.jar

Now, the final step is to convert the jar file to readable java file format.

To do this, you can use a Java decompiler such as JD-GUI.

Just opening the jar file in the JD-GUI, we would be having the java readable code with us.



We would be having a screen like the above with the java codes. Using JD-GUI, we could also save all the files and modify it later, and then recompile it to make a modified application. This is how most of the mobile malwares for this platform are made, by inserting the malicious codes into the legitimate app and then repackaging it to make a new APK looking exactly similar to the original one.

Analysing the java/smali files could also be useful from a penetration testing point of view. However mobile application penetration testing consists of lot more things than just analysing the codes. One also needs to capture the network data, and then see if, any information is being sent by an application to a remote server of the attacker. Also, one need to trace the system calls and the changes made to the Dalvik Virtual Machine, after installing the application on it.

Thank You.

You could always reach to me for any kind of projects or discussions.

About the Author

Aditya Gupta is a Cyber Security Consultant and Ethical Hacker. He loves researching on Web Application Security and Android Malwares. He also gives workshops and talks on Cyber and Mobile Security.

References

Android Developers (http://developer.android.com/)

APKTool by Brut.All (http://code.google.com/p/android-apktool/)

Java Decompiler JD-GUI (http://java.decompiler.free.fr/?q=jdgui)

Dex2Jar converter (http://code.google.com/p/dex2jar/)