Lab SSC week 7

The learning objective of this lab is for students to gain first-hand experience on analyzing Android applications, decompiling and understanding how code injection is applied on Android based applications.

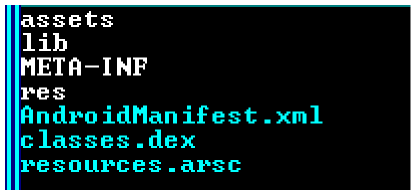
Lab structure

Alin-Mihai BARBATEI

Junior Malware Analyst

Introduction

Android application package (**APK**) is the package file format used to distribute and install application software and middleware onto Google's Android OS [1].



```
assets
lib
META-INF
res
AndroidManifest.xml
classes.dex
resources.arsc
```

An apk is essentially an archive file (derived from the zip format), which means it can contain anything or nothing.

Android being a Google asset, the latter has dictated that in order for an apk to be valid it must contain the following components:

- **`META-INF` directory: the directory where signature data is stored which is used to ensure the integrity of the .apk package and system security.**

- `lib`: the directory containing the compiled code that is specific to a software layer of a processor; the directory is split into several directories within it:
  - `armeabi`: compiled code for all ARM based processors only
  - `armeabi-v7a`: compiled code for all ARMv7 and above based processors only
  - `x86`: compiled code for x86 processors only
  - `mips`: compiled code for MIPS processors only

- `res`: the directory containing resources not compiled into resources.arsc (see below).

- `assets`: a directory containing applications assets, which can be retrieved by `AssetManager`.

- **`AndroidManifest.xml`: An additional Android manifest file, describing the name, version, access rights, referenced library files for the application in Android binary XML.**

- **`classes.dex`: The classes compiled in the dex file format understandable by the Dalvik Virtual machine.**

- `resources.arsc`: a file containing precompiled resources, such as binary XML for example.

From a reverse-engineering point of view, the highlighted are more relevant.

`classes.dex`

- This file contains the application's code structured in the dex file format [2] understood and interpreted by the Dalvik Virtual machine [3].
- Any modification to the underlying application must be committed to this file.

`AndroidManifest.xml`:

- The manifest contains information about the capabilities of the application [4] defined by its permissions [5] and its components: activities, receivers, services and providers [6].
- Contains the applications original EP (Entry Point).

- Found in an unreadable form (compressed).
- Any modification to the behavior, that requires further permissions, or component change to the application must be committed to this file.

`META-INF` directory:

- Android applications are required to be signed in order to be valid for publication on Google Play Market and other Android application markets.
- When using Google's Android emulator the apk also needs to be digital signed. This was imposed by Google but it is not required for the application to actually run on third party devices.
- The application is signed with a digital key that is unique to the developer and stands as a certificate of guarantee of the origin and integrity of the application.

Tools

apktool

- Is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications.
- For this laboratory version 2.0.0-rc2 is used.
- Homepage:
  - http://connortumbleson.com/2014/10/05/apktool-2-0-0-rc2-released/
- Installation guide:
  - https://code.google.com/p/android-apktool/wiki/Install
- Download Link:
  - https://bitbucket.org/iBotPeaches/apktool/downloads/apktool_2.0.0rc2.jar

Java JDK

- JDK version depends on what version of apktool is used. If version 2.0.0+ is used then JDK 1.7+ is needed, else JDK 1.6.

Eclipse ADT Bundle

- For the purpose of testing our apps and developing code.
- Download:
  - http://developer.android.com/sdk/index.html

Jadx – or any other (good) java/Dalvik bytecode decompiler

- Currently one of the best Dalvik bytecode decompilers.
- For this laboratory it is only used as a means to debug
- Git:
  - https://github.com/skylot/jadx
- Download:
  - https://github.com/skylot/jadx/releases

Telnet

- Windows distributions come with telnet deactivated, you must first activate it
- How-to (works on Windows 8 as well):
    - http://windows.microsoft.com/en-us/windows/telnet-faq#1TC=windows-7

Wireshark - OPTIONAL

- When starting Google's Android Emulator with the option `-tcpdump`, the corresponding TCP dump can be analyzed with wireshark.
- Wireshark GUI can analyze the dump only after the emulator is closed. This is because some format specific information is buffered and only written by the emulator to the dump file on close.

Very Nice App

Task

Consider the following scenario:

Very Nice App is a very nice app which is found on a respectable market. You, as a concerned citizen, want to point out to the whole world how dangerous the market has become and how easy it can be for ANYONE to modify an existing, and very nice, application into a malicious application that can steal your money.

For this, you have downloaded the application (in /Desktop/work), have set up a malicious lab (installed apktool, jadx, adt-bundle) and are now ready to conduct your “evil” work.

Your tasks:

1. Write malicious code in Eclipse.
2. Decompile your code to smali programming language using apktool.
3. Decompile Very Nice App.
4. Modify your code so it can be injected into the target application.
5. Inject your code.
6. Recompile the application using apktool with the necessary option.
7. Sign you malefic application.
8. Let it wreak havoc upon the Wild Wilde Web.

Sending SMS to/Calling the emulator

When you were working with the Android Emulator you found out that you can send SMS messages to it and call it using telnet. This helped you in verifying that your code injection was successful.

The emulator starts using a specific port. You can see the port number in the emulator window or by typing `adb devices`. Using that port number to connect with telnet, you can send SMS to/Call the emulator using the following commands:

#connect to the emulator
```
telnet localhost 5554
```

#send a SMS message with the body "your sms goes here" from the displayed number "+1234567"
#to the emulator
```
sms send +1234567 your sms goes here
```

#call the emulator with the posing number of "+1234567"
```
gsm call +1234567
```

More detailed information [7]

Seeing if the emulator has sent a SMS

You have also observed the logcat information from the adb and realized that you can use it to see if the emulator has sent a SMS message.

Use command:
```
adb logcat –b radio > radio.log
```
then parse the correspoing log for "^Z" which indicates that a SMS message was sent.

Signing your Application

In order to achieve this you need to:

1. Generate a private key using keytool. For example:

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

2. Sign your app with your private key using jarsigner:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore my-release-key.keystore my_application.apk alias_name
```

You can also verify that your APK is signed. For example:

```
$ jarsigner -verify -verbose -certs my_application.apk
```

3. Finally you need to align the APK package using zipalign.

```
$ zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

The entire process is detailed at [8]

Disclaimer
The work carried out in this laboratory and the sample analyzed was written for the Computer Security laboratory of 2014. It is not intended and is not to be used to commit unlawful actions; it serves only to highlight the danger of 3rd party markets, the extent of what malware writing has become and **how to prevent it.**

Resources

[1] http://en.wikipedia.org/wiki/Android_application_package
[2] https://source.android.com/devices/tech/dalvik/dex-format.html
[3] http://youtu.be/ptjedOZEXPM
[4] http://developer.android.com/guide/topics/manifest/manifest-intro.html
[5] http://developer.android.com/reference/android/Manifest.permission.html
[6] http://developer.android.com/guide/components/fundamentals.html
[7] http://www.tutorialspoint.com/android/android_emulator.htm
[8] http://developer.android.com/tools/publishing/app-signing.html