February 6, 2017

Reverse Engineering Android Applications

netbiosX | Mobile Pentesting | Android, APK, Drozer, Mobile, MobSF, Reverse Engineering | 1 Comment

Mobile application penetration tests go beyond the standard discovery of vulnerabilities through Burp Suite. It is vital to know how to decompile the application for the examination of vulnerabilities into the application code. The purpose of this article is to demonstrate various techniques and tools of how to reverse engineer an android application.

In order to start the reversing process the APK file of the target application is needed. Usually the client is responsible to provide this file to the penetration tester. However if for whatever the reason this is not possible then this article explains various methods of how to retrieve the APK file from Google Play Store and from the actual device.

The APK File

Android Application Package (APK) files are the files which are used by the Android operating system for distribution and installation of mobile applications. Typically an APK file is just a zip file which has been renamed as an APK in order the Android operating system to recognize it as an executable. The unzip utility can be used to extract files that are stored inside the APK.

```
netbiosx@ubuntu:~/Downloads/Vulnerable Applications$ unzip sieve.apk
Archive:  sieve.apk
  inflating: res/layout/activity_add_entry.xml
  inflating: res/layout/activity_file_select.xml
  inflating: res/layout/activity_main_login.xml
  inflating: res/layout/activity_pin.xml
  inflating: res/layout/activity_pwlist.xml
  inflating: res/layout/activity_settings.xml
  inflating: res/layout/activity_short_login.xml
  inflating: res/layout/activity_welcome.xml
  inflating: res/layout/format_pwlist.xml
  inflating: res/menu/activity_add_entry_add.xml
  inflating: res/menu/activity_add_entry_edit.xml
  inflating: res/menu/activity_file_select.xml
  inflating: res/menu/activity_main_login.xml
  inflating: res/menu/activity_pin.xml
  inflating: res/menu/activity_pwlist.xml
  inflating: res/menu/activity_settings.xml
  inflating: res/menu/activity_short_login.xml
  inflating: res/menu/activity_welcome.xml
  inflating: res/xml/prefrences.xml
  inflating: AndroidManifest.xml
 extracting: resources.arsc
```

Extracting Data of an APK File

Every APK contains the following files:

- **AndroidManifest.xml** // Defines the permissions of the application
- **classes.dex** // Contains all the java class files
- **resources.arsc** // Contains all the meta-information about the resources and nodes

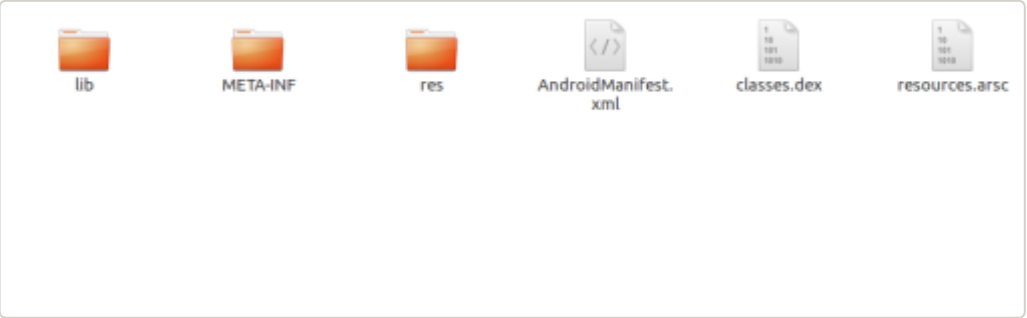
Recent Posts

- AppLocker Bypass – CMSTP
- PDF – NTLM Hashes
- NBNS Spoofing
- Lateral Movement – RDP
- DCShadow

Categories

- **Coding** (10)
- **Defense Evasion** (20)
- **Exploitation Techniques** (19)
- **External Submissions** (3)
- **General Lab Notes** (21)
- **Information Gathering** (12)
- **Infrastructure** (2)
- **Maintaining Access** (4)
- **Mobile Pentesting** (7)
- **Network Mapping** (1)
- **Post Exploitation** (11)
- **Privilege Escalation** (14)
- **Red Team** (24)
- **Social Engineering** (11)
- **Tools** (7)
- **VoIP** (4)
- **Web Application** (14)
- **Wireless** (2)

Archives



Contents of an APK File

Alternatively other tools can be used as well to decompile an android application. The **d** parameter instructs the apktool to decompile the APK.

```
netbiosx@ubuntu:~$ apktool d -r -s /home/netbiosx/Downloads/Vulnerable\ Applicat
ions/sieve.apk
 06:09:21 up 1 day,  2:38,  1 user,  load average: 0.35, 0.36, 0.26
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
netbiosx tty7     :0               Sat03   26:38m  5:28   1.69s /sbin/upstart -
I: Using Apktool 2.0.2-dirty on sieve.apk
I: Copying raw resources...
I: Copying raw classes.dex file...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

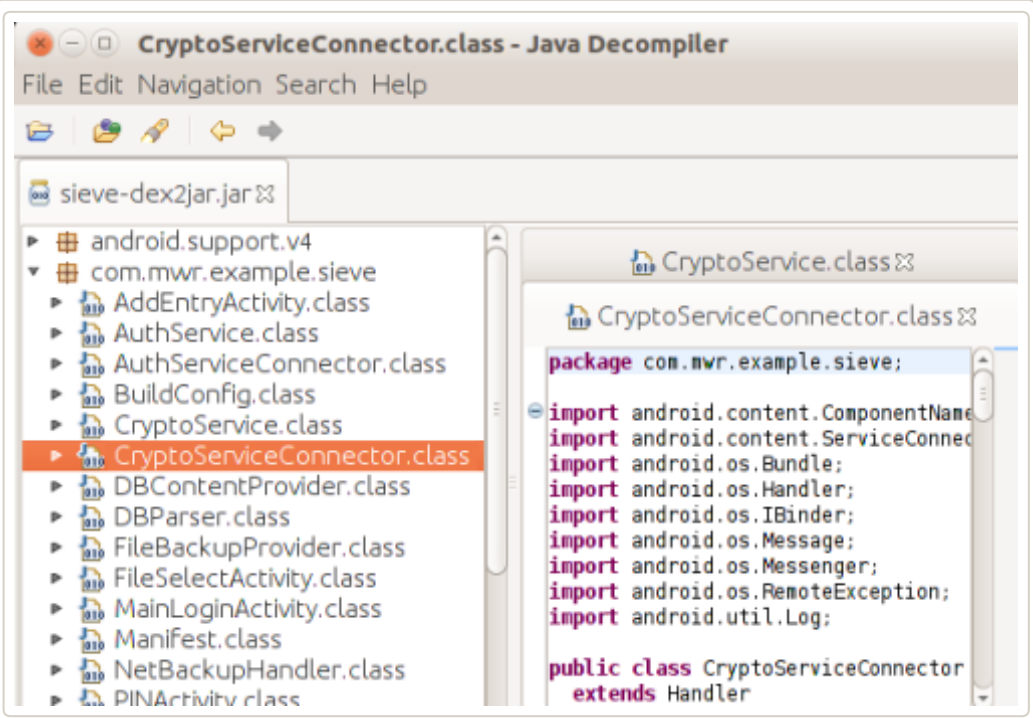
Decompile an APK via apktool

Another method is to try to convert the actual .APK file into .JAR and then use any tool that can decompile java code.

```
netbiosx@ubuntu:~/Tools/dex2jar-2.0$ ./d2j-dex2jar.sh -f /home/netbiosx/Download
s/Vulnerable\ Applications/sieve.apk
dex2jar /home/netbiosx/Downloads/Vulnerable Applications/sieve.apk -> ./sieve-de
x2jar.jar
netbiosx@ubuntu:~/Tools/dex2jar-2.0$
```

Convert APK files to JAR

- May 2018
- April 2018
- January 2018
- December 2017
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017
- March 2017
- February 2017
- January 2017
- November 2016
- September 2016
- February 2015
- January 2015
- July 2014
- April 2014
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012



Decompile JAR File to native Java code

Extraction of APK files can be done also with the Android Asset Packaging Tool.

- August 2012
- July 2012
- June 2012
- April 2012
- March 2012
- February 2012

@ Twitter

- @n0dec Thanks for sharing this. Good work! I guess another rule can be created easily for cmstp execution from this… **twitter.com/i/web/status/9… 23 hours ago**
- RT @n0dec: @netbiosX Note that .inf extension it's not necessary to execute the file 🤔. I uploaded a gist with one **#malwless** set to test ou… **1 day ago**
- Microsoft Word Document Upload to Stored XSS: A Case Study **coalfire.com/Solutions/Coal… 1 day ago**
- RT @bohops: Another great post! **twitter.com/netbiosX/statu… 1 day ago**
- Files for DLL and SCT Execution via CMSTP **gist.github.com/netbiosX/15c96… 1 day ago**

Follow @netbiosX

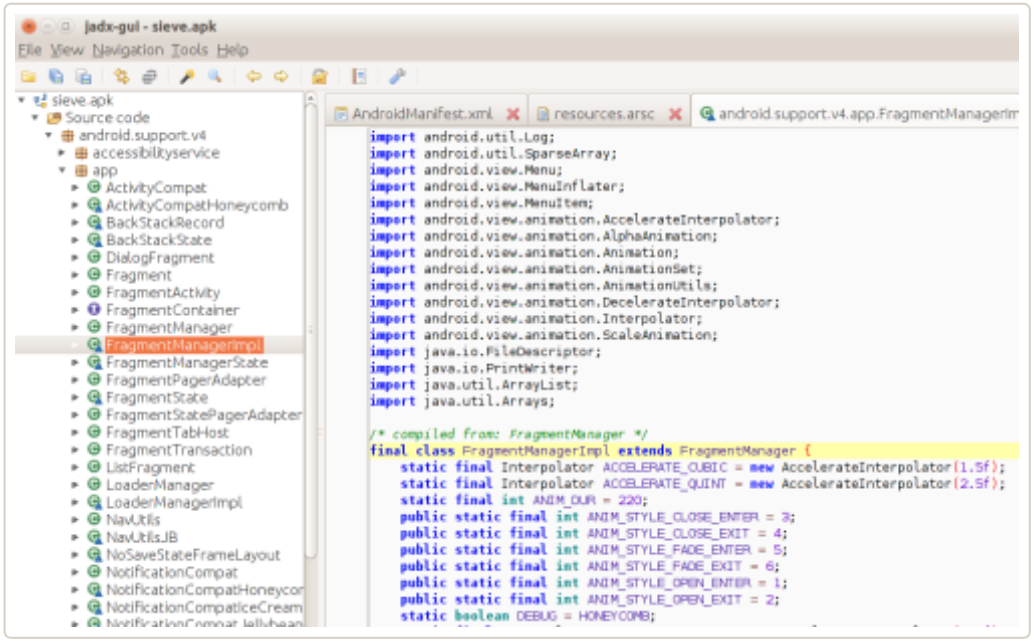
Pen Test Lab Stats

- 2,958,985 hits

Blogroll

```
netblosx@ubuntu:~$ aapt l -a /home/netblosx/Downloads/Vulnerable\ Applications/s
ieve.apk
res/layout/activity_add_entry.xml
res/layout/activity_file_select.xml
res/layout/activity_main_login.xml
res/layout/activity_pin.xml
res/layout/activity_pwlist.xml
res/layout/activity_settings.xml
res/layout/activity_short_login.xml
res/layout/activity_welcome.xml
res/layout/format_pwlist.xml
res/menu/activity_add_entry_add.xml
res/menu/activity_add_entry_edit.xml
res/menu/activity_file_select.xml
res/menu/activity_main_login.xml
res/menu/activity_pin.xml
res/menu/activity_pwlist.xml
res/menu/activity_settings.xml
res/menu/activity_short_login.xml
res/menu/activity_welcome.xml
res/xml/prefrences.xml
AndroidManifest.xml
```

Jadx is another tool which can produce Java source code from Android APK and DEX files.



Decompile an APK via Jadx

Android Manifest

The android asset packaging tool can be used to obtain the manifest file of an APK application.

Professional

- **The Official Social Engineering Portal** Information about the Social Engineering Framework,Podcasts and Resources 0

Next Conference

Security B-Sides London
April 29th, 2014

The big day is here.

Facebook Page

Penetrati…
9.9K likes

Like Page

Be the first of your friends to like this

```
netbiosx@ubuntu:~$ aapt dump xmltree /home/netbiosx/Downloads/Vulnerable\ Applic
ations/sieve.apk AndroidManifest.xml
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.mwr.example.sieve" (Raw: "com.mwr.example.sieve")
    E: uses-permission (line=7)
      A: android:name(0x01010003)="android.permission.READ_EXTERNAL_STORAGE" (Ra
w: "android.permission.READ_EXTERNAL_STORAGE")
    E: uses-permission (line=8)
      A: android:name(0x01010003)="android.permission.WRITE_EXTERNAL_STORAGE" (R
aw: "android.permission.WRITE_EXTERNAL_STORAGE")
    E: uses-permission (line=9)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw: "android.p
ermission.INTERNET")
    E: permission (line=11)
      A: android:label(0x01010001)="Allows reading of the Key in Sieve" (Raw: "A
llows reading of the Key in Sieve")
      A: android:name(0x01010003)="com.mwr.example.sieve.READ_KEYS" (Raw: "com.m
wr.example.sieve.READ_KEYS")
```

Retrieving the Manifest File from aapt

As the output above is not easy readable the following command can dump only the permissions of the application.

```
netbiosx@ubuntu:~$ aapt dump permissions /home/netbiosx/Downloads/Vulnerable\ Ap
plications/sieve.apk
package: com.mwr.example.sieve
uses-permission: name='android.permission.READ_EXTERNAL_STORAGE'
uses-permission: name='android.permission.WRITE_EXTERNAL_STORAGE'
uses-permission: name='android.permission.INTERNET'
permission: com.mwr.example.sieve.READ_KEYS
permission: com.mwr.example.sieve.WRITE_KEYS
```

Retrieving Permissions from the Manifest

Alternatively if the contents of the APK file are already extracted then a more specialized tool like AXMLPrinter can be used to read the XML file in a more elegant way.

```
netbiosx@ubuntu:~/Applications$ java -jar AXMLPrinter2.jar /home/netbiosx/Downlo
ads/Vulnerable\ Applications/AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:versionCode="1"
        android:versionName="1.0"
        package="com.mwr.example.sieve"
        >
        <uses-permission
                android:name="android.permission.READ_EXTERNAL_STORAGE"
                >
        </uses-permission>
        <uses-permission
                android:name="android.permission.WRITE_EXTERNAL_STORAGE"
                >
        </uses-permission>
        <uses-permission
                android:name="android.permission.INTERNET"
                >
        </uses-permission>
```

Viewing the Android Manifest File

Drozer can also parse the manifest files of installed applications:

```
dz> run app.package.manifest com.mwr.dz
<manifest versionCode="5"
versionName="2.3.4"
package="com.mwr.dz">
<uses-sdk minSdkVersion="7"
targetSdkVersion="18">
</uses-sdk>
<uses-permission name="android.permission.INTERNET">
</uses-permission>
<application theme="@2131165185"
label="@2131099648"
icon="@2130837513"
debuggable="true"
```

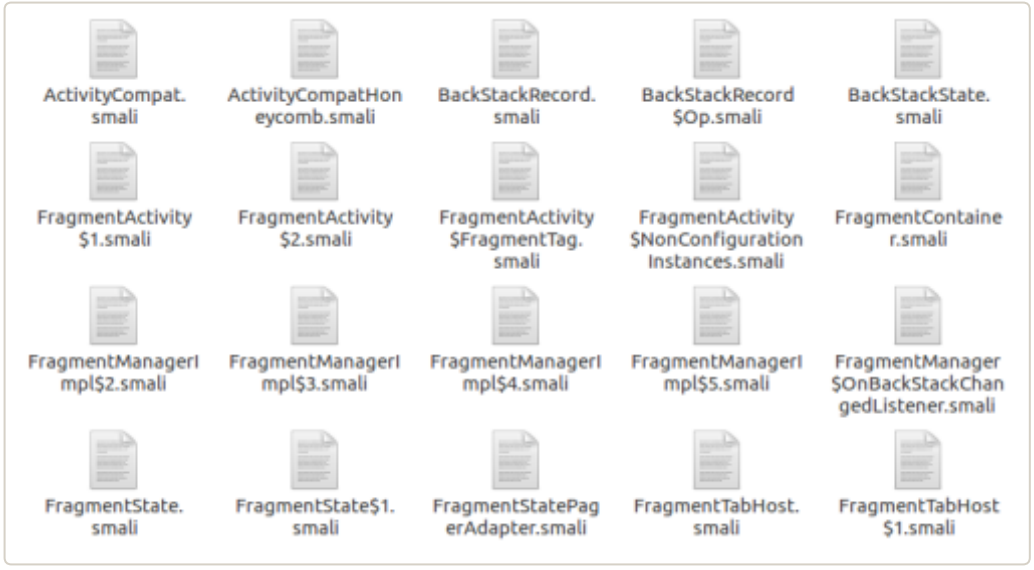
Extensive information of how to assess Android Manifest files can be found in this article.

Classes DEX

The classes.dex file contains all the java classes of the application and it can be disassembled with baksmali tool to retrieve the java source code.

```
netbiosx@ubuntu:~/Applications$ java -jar baksmali-2.2b4.jar disassemble /home/n
etbiosx/Downloads/Vulnerable\ Applications/classes.dex
netbiosx@ubuntu:~/Applications$
```

Decompiling DEX Files



Disassemble DEX Files

DEX files can be also disassembled into Dalvik instructions:

```
043ed4:                                          |[043ed4] android.support.v4.view
.ViewCompat.JbMr1ViewCompatImpl.<init>:()V
043ee4: 7010 3b07 0000                           |0000: invoke-direct {v0}, Landro
id/support/v4/view/ViewCompat$JBViewCompatImpl;.<init>:()V // method@073b
043eea: 0e00                                     |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=308
      locals        :
        0x0000 - 0x0004 reg=0 this Landroid/support/v4/view/ViewCompat$JbMr1View
CompatImpl;

  Virtual methods   -
    #0              : (in Landroid/support/v4/view/ViewCompat$JbMr1ViewCompatImp
l;)
      name          : 'getLabelFor'
      type          : '(Landroid/view/View;)I'
      access        : 0x0001 (PUBLIC)
      code          -
      registers     : 3
      ins           : 2
      outs          : 1
      insns size    : 5 16-bit code units
```

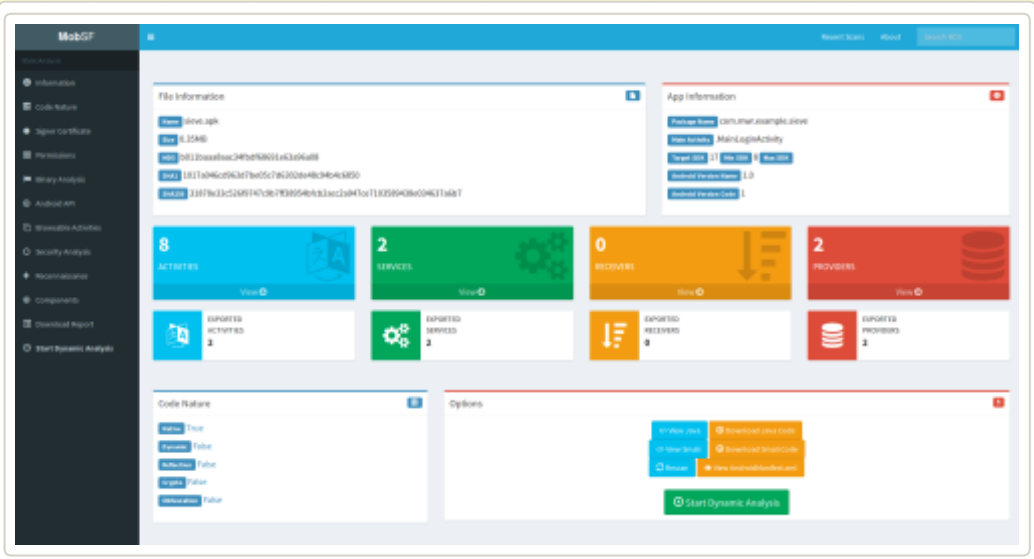
Dalvik Instructions

MobSF

The mobile security framework is all in one suite that can be used to perform static and dynamic code analysis for Android, iOS and Windows phone applications. MobSF automates the process that is has been described in this article as it can decompile the APK, read the manifest file, identify issues in the source code and in the Manifest file, extract the certificate of the application etc.

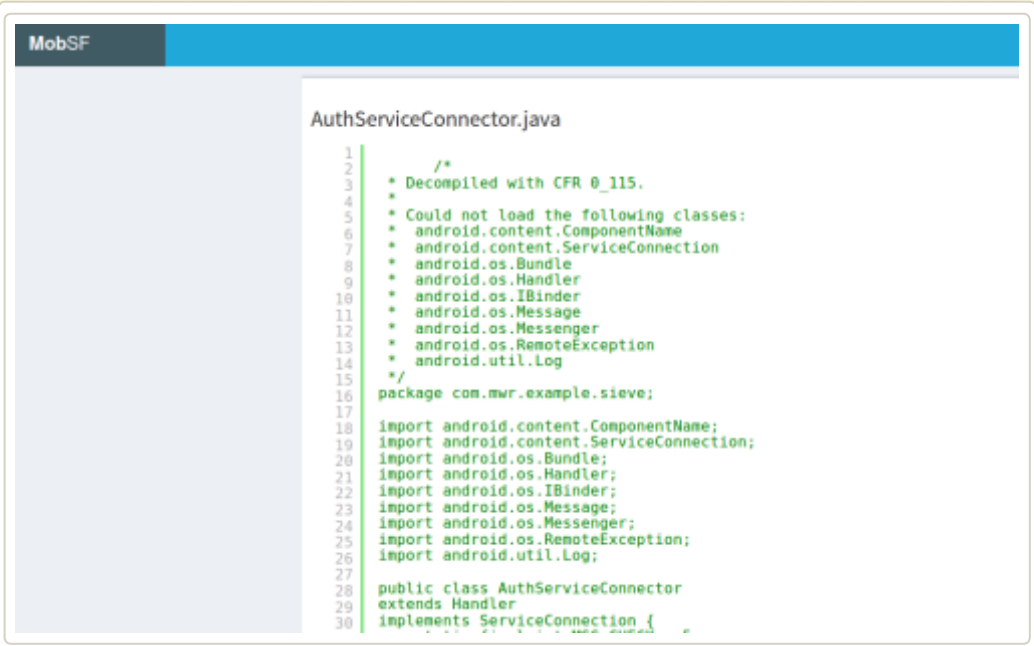
MobSF – Main Page

The image below demonstrates the analysis of an APK file via the mobile security framework:



MobSF – APK File Analysis

From the moment that the APK is uploaded the framework decompiles the application automatically so it eliminates the need for further tools.



Decompiled Java Code

The official GitHub repository of the tool is: https://github.com/MobSF/Mobile-Security-Framework-MobSF

Summary

Reverse engineering an android application can give an understanding of how the application really works in the background and how it interacts with the actual phone. This knowledge would assist in the process of discovery vulnerabilities that exist in the code and are not obvious.