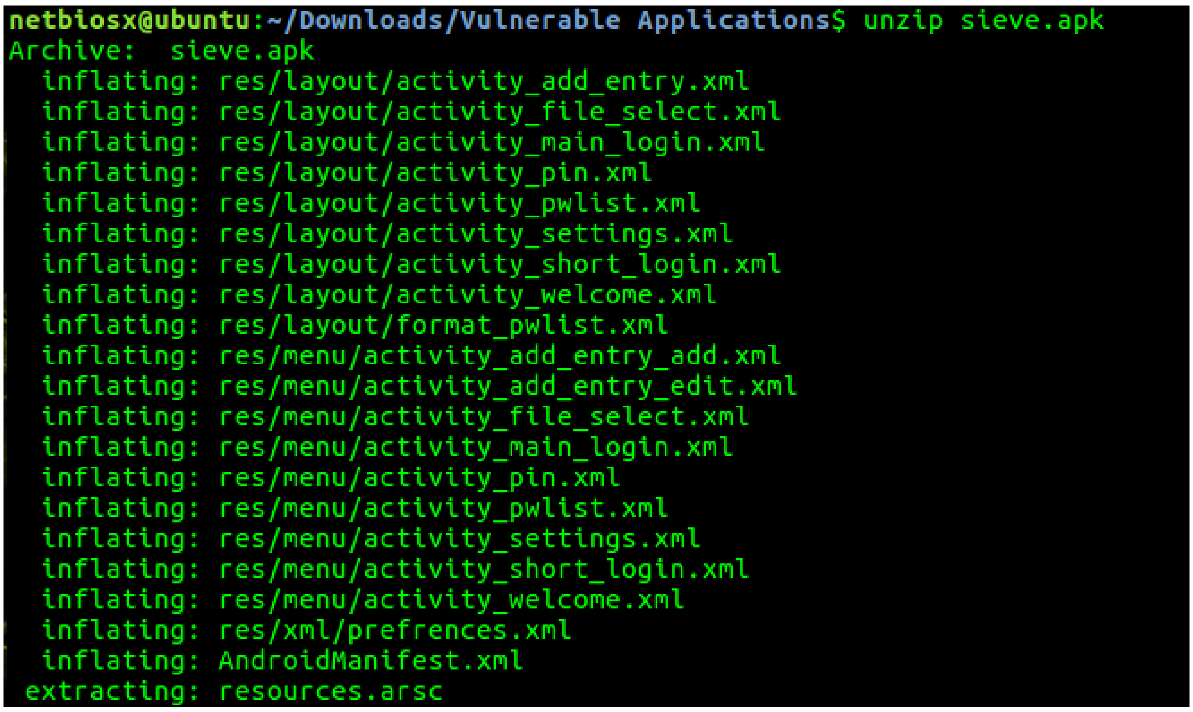
# Reverse Engineering Android Applications

pentestlab.blog/category/mobile-pentesting/page/4

February 6, 2017



```
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
```

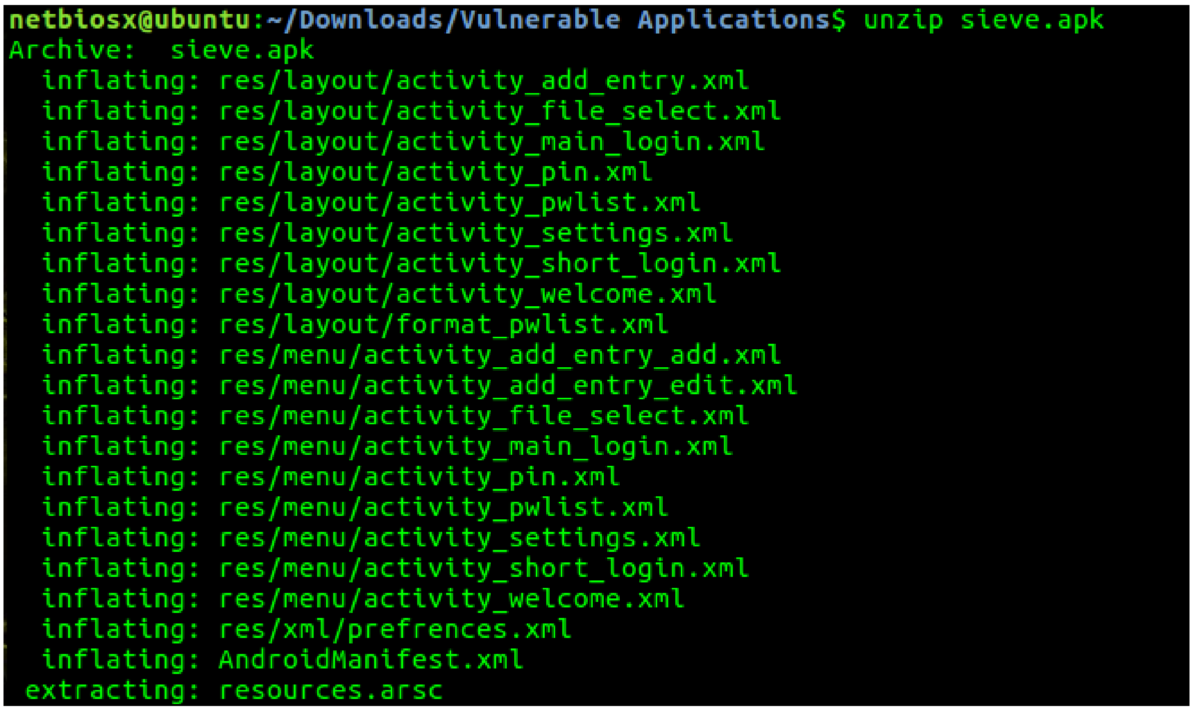
Extracting Data of an APK File

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.

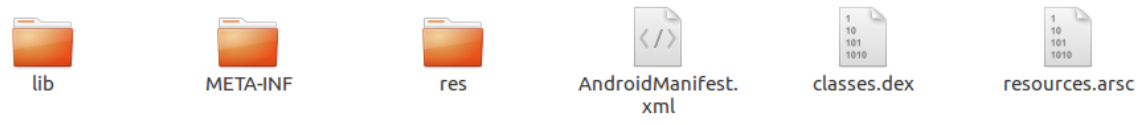


```
netbiosx@ubuntu:~/Downloads/Vulnerable Applications$ unzip sieve.apk
Archive:  sieve.apk
  inflating: res/layout/activity_add_entry.xml
  inflating: res/layout/activity_file_select.xml
  inflating: res/layout/activity_main_login.xml
  inflating: res/layout/activity_pin.xml
  inflating: res/layout/activity_pwlist.xml
  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

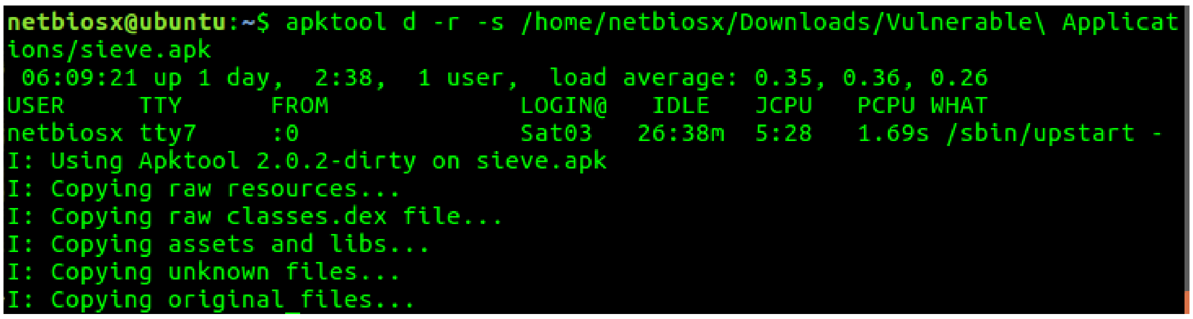
Every APK contains the following files:

- **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



*Contents of an APK File*

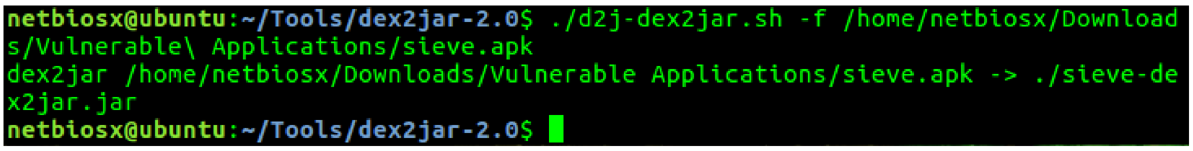
Alternatively other tools can be used as well to decompile an android application. The **d** parameter instructs the apktool to decompile the APK.



```
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...
```

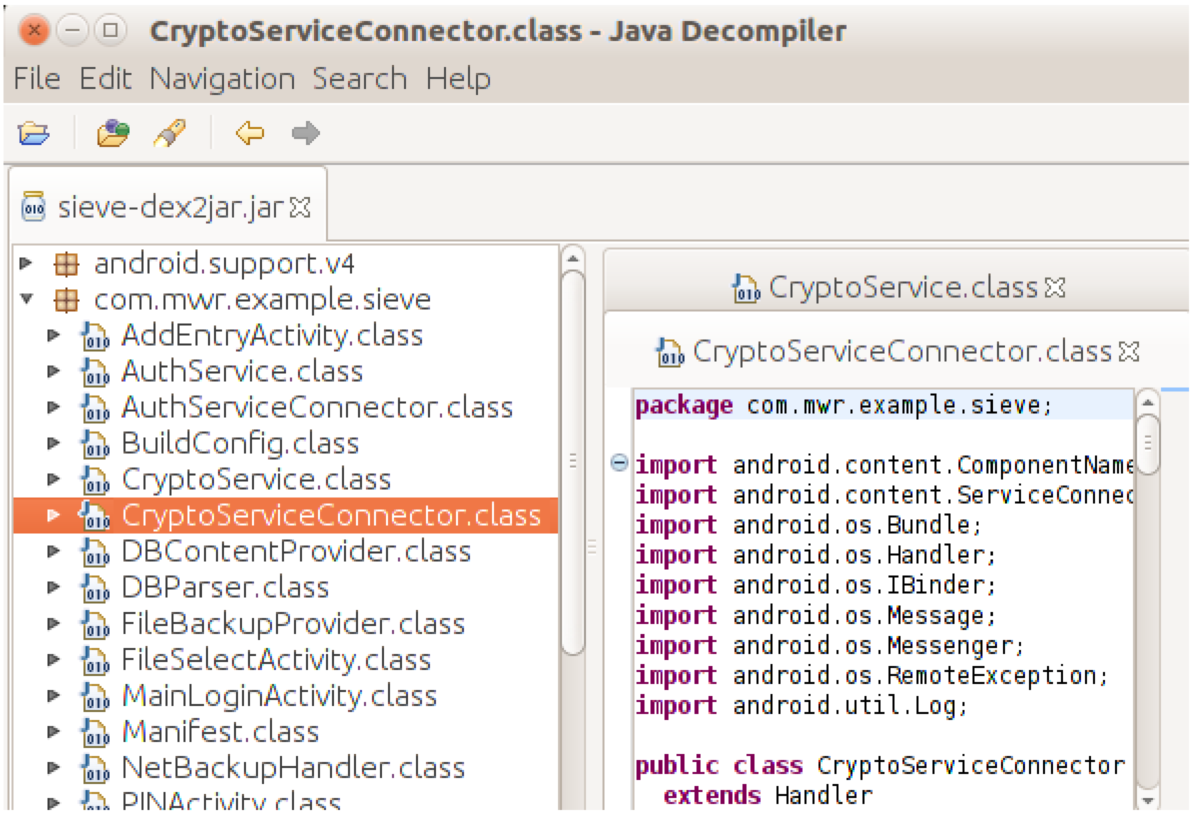
Decompile an APK via apktool

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



CryptoServiceConnector.class - Java Decompiler

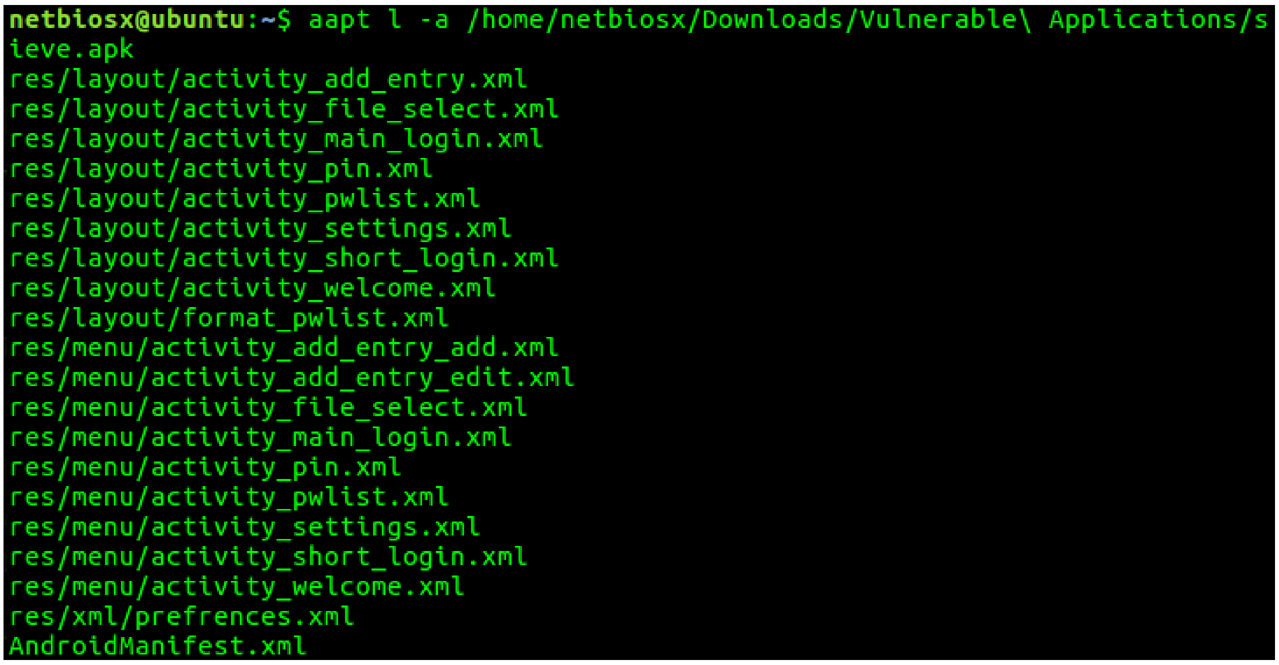
```
package com.mwr.example.sieve;

import android.content.ComponentName
import android.content.ServiceConnec
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

public class CryptoServiceConnector
  extends Handler
```

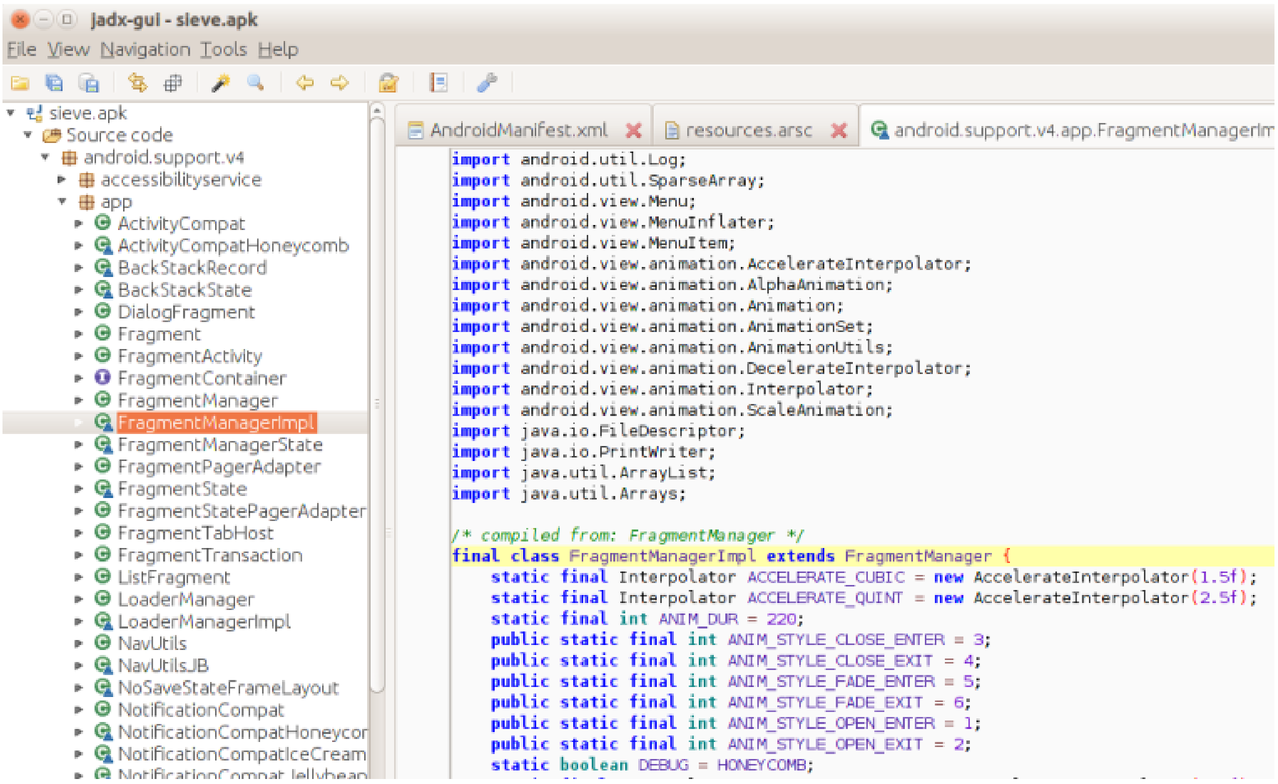
Decompile JAR File to native Java code

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



```
netbiosx@ubuntu:~$ aapt l -a /home/netbiosx/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.


jadx-gui - sieve.apk
File View Navigation Tools Help
AndroidManifest.xml
resources.arsc
android.support.v4.app.FragmentManagerIm

```
sieve.apk
  Source code
    android.support.v4
      accessibilityservice
      app
        ActivityCompat
        ActivityCompatHoneycomb
        BackStackRecord
        BackStackState
        DialogFragment
        Fragment
        FragmentActivity
        FragmentContainer
        FragmentManager
        FragmentManagerImpl
        FragmentManagerState
        FragmentPagerAdapter
        FragmentState
        FragmentStatePagerAdapter
        FragmentTabHost
        FragmentTransaction
        ListFragment
        LoaderManager
        LoaderManagerImpl
        NavUtils
        NavUtilsJB
        NoSaveStateFrameLayout
        NotificationCompat
        NotificationCompatHoneycor
        NotificationCompatIceCream
```

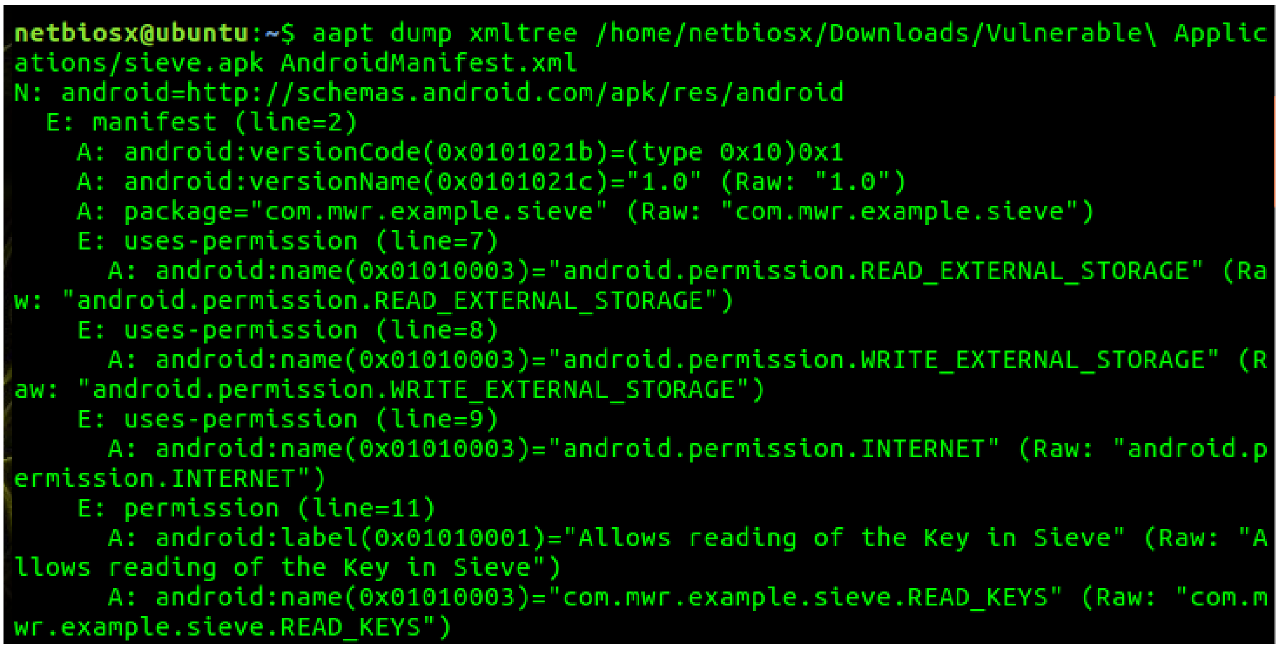
```
import android.util.Log;
import android.util.SparseArray;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.animation.AccelerateInterpolator;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.AnimationUtils;
import android.view.animation.DecelerateInterpolator;
import android.view.animation.Interpolator;
import android.view.animation.ScaleAnimation;
import java.io.FileDescriptor;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;

/* compiled from: FragmentManager */
final class FragmentManagerImpl extends FragmentManager {
    static final Interpolator ACCELERATE_CUBIC = new AccelerateInterpolator(1.5f);
    static final Interpolator ACCELERATE_QUINT = new AccelerateInterpolator(2.5f);
    static final int ANIM_DUR = 220;
    public static final int ANIM_STYLE_CLOSE_ENTER = 3;
    public static final int ANIM_STYLE_CLOSE_EXIT = 4;
    public static final int ANIM_STYLE_FADE_ENTER = 5;
    public static final int ANIM_STYLE_FADE_EXIT = 6;
    public static final int ANIM_STYLE_OPEN_ENTER = 1;
    public static final int ANIM_STYLE_OPEN_EXIT = 2;
    static boolean DEBUG = HONEYCOMB;
```

Decompile an APK via Jadx

## Android Manifest

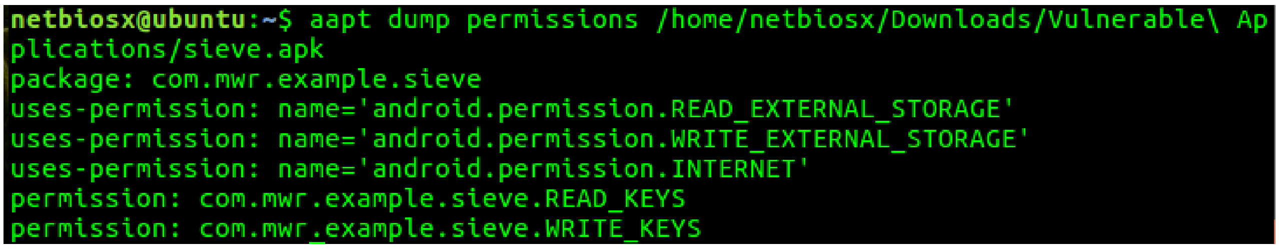
The android asset packaging tool can be used to obtain the manifest file of an APK application.



```
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

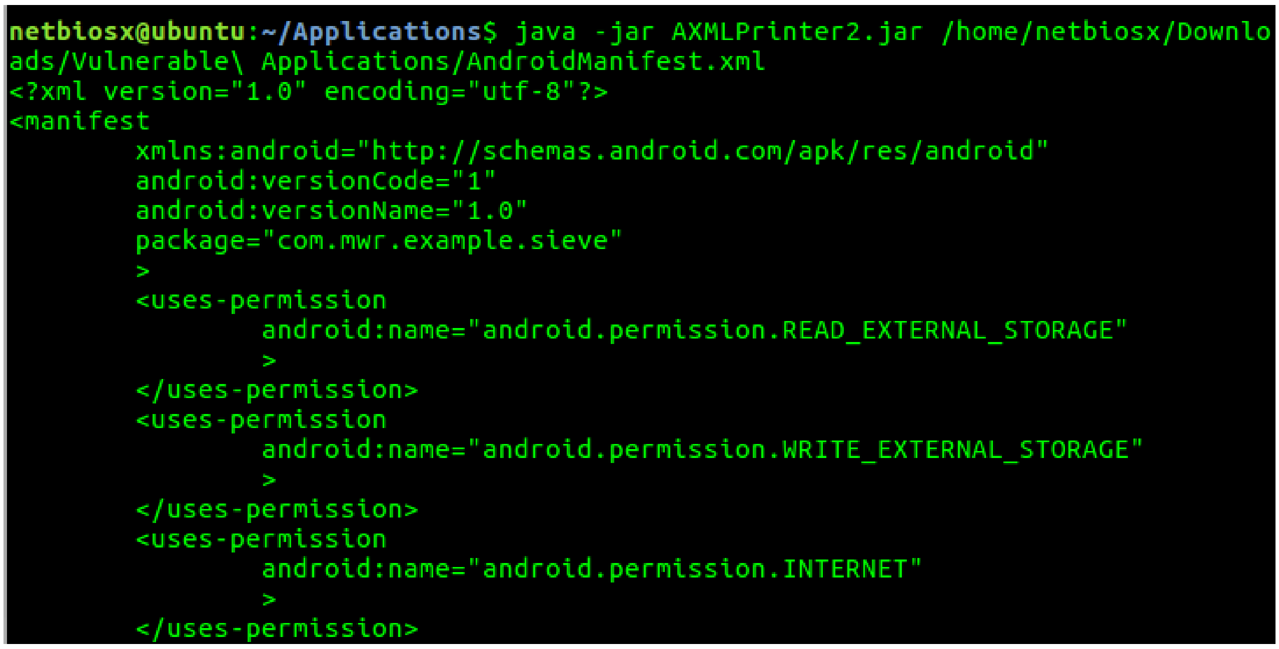
As the output above is not easy readable the following command can dump only the permissions of the application.



```
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

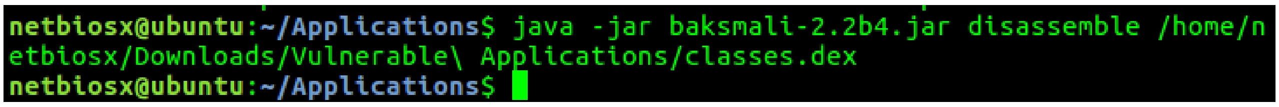
Drozer can also parse the manifest files of installed applications:

```
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.

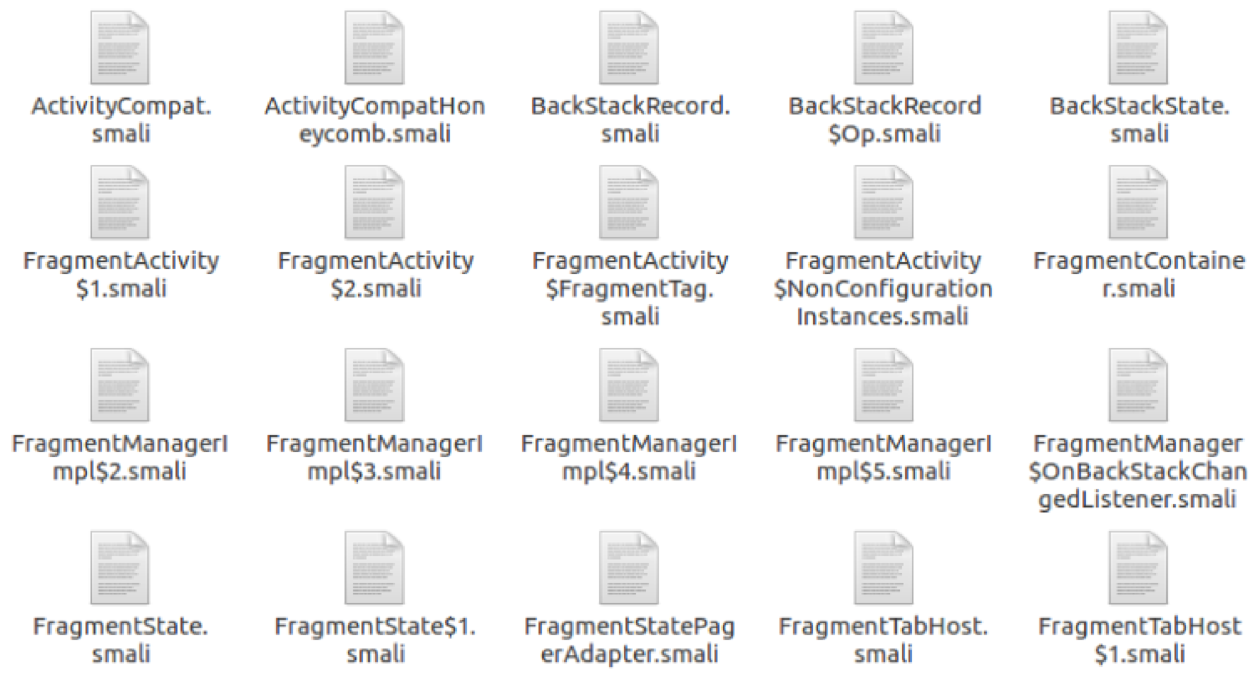
## Classes DEX

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$ 
```

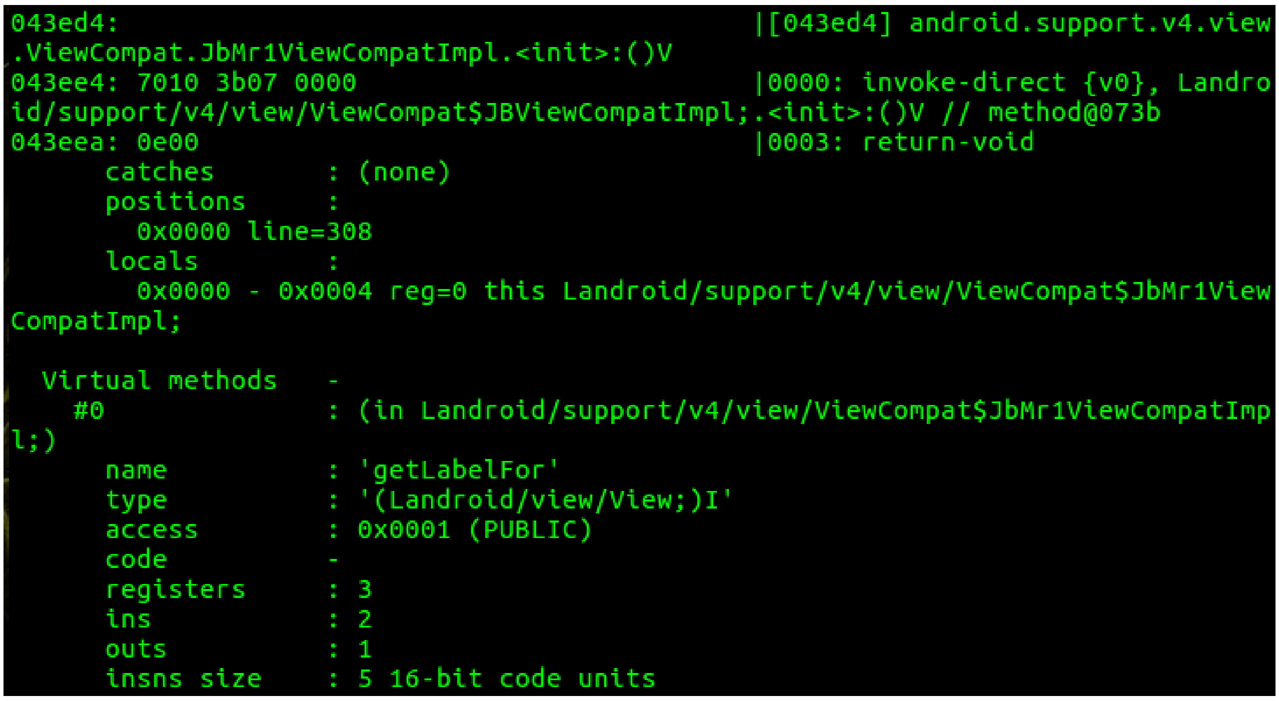
Decompiling DEX Files



```
ActivityCompat.smali
ActivityCompatHoneycomb.smali
BackStackRecord.smali
BackStackRecord$Op.smali
BackStackState.smali
FragmentActivity$1.smali
FragmentActivity$2.smali
FragmentActivity$FragmentTag.smali
FragmentActivity$NonConfigurationInstances.smali
FragmentContainer.smali
FragmentManagerImpl$2.smali
FragmentManagerImpl$3.smali
FragmentManagerImpl$4.smali
FragmentManagerImpl$5.smali
FragmentManager$OnBackStackChangedListener.smali
FragmentState.smali
FragmentState$1.smali
FragmentStatePagerAdapter.smali
FragmentTabHost.smali
FragmentTabHost$1.smali
```

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
```

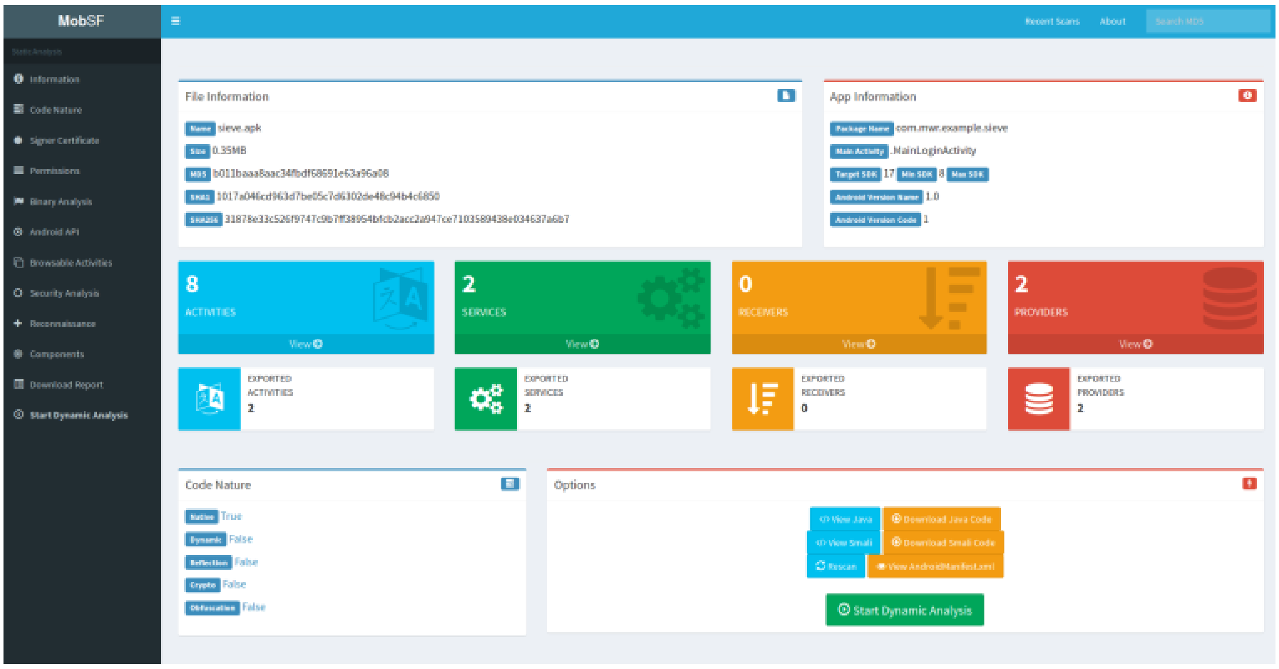
Dalvik Instructions

## 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:


File Information

Name sieve.apk
Size 0.35MB
MD5 b011baaa8aac34fbdf68691e63a96a08
SHA1 1017a046cd963d7be05c7d6302de48c94b4c6850
SHA256 31878e33c526f9747c9b7ff38954bfcb2acc2a947ce7103589438e034637a6b7

App Information

Package Name com.mwr.example.sieve
Main Activity .MainLoginActivity
Target SDK 17 Min SDK 8 Max SDK
Android Version Name 1.0
Android Version Code 1

8 ACTIVITIES
2 SERVICES
0 RECEIVERS
2 PROVIDERS

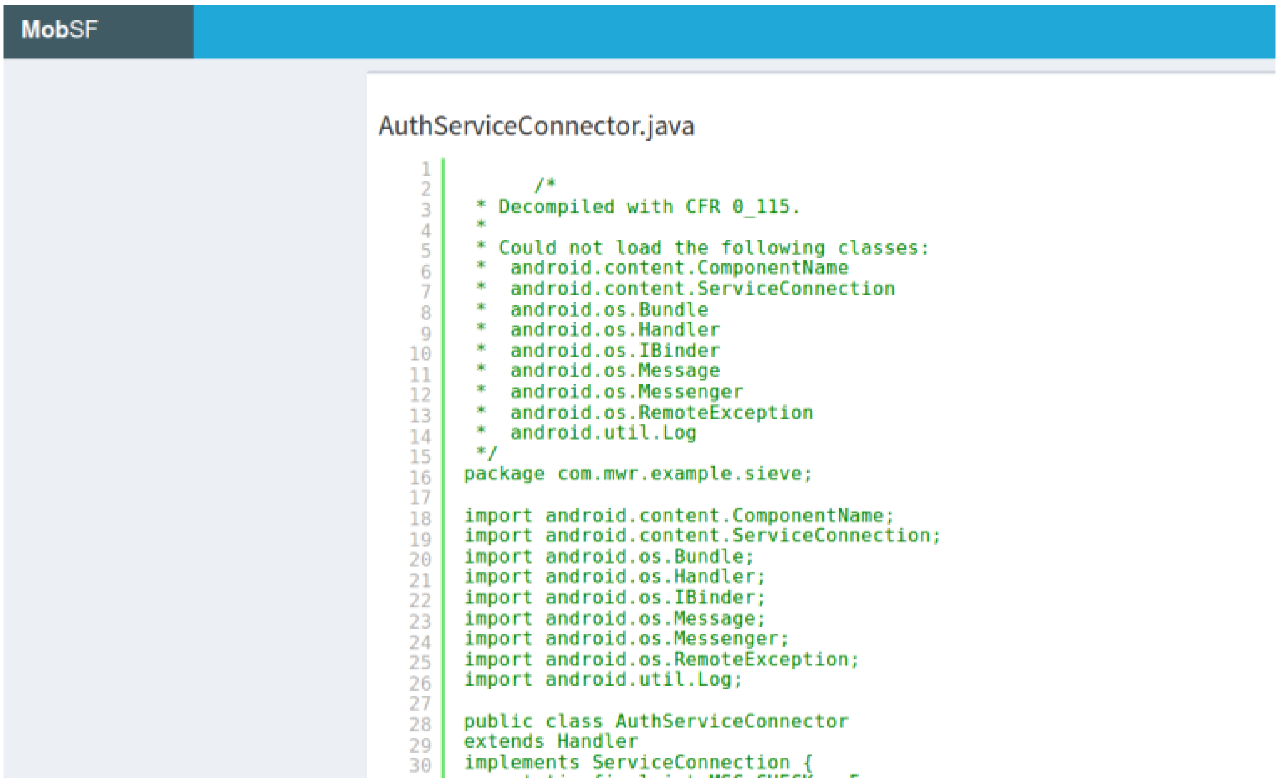
EXPORTED ACTIVITIES 2
EXPORTED SERVICES 2
EXPORTED RECEIVERS 0
EXPORTED PROVIDERS 2

Code Nature

Native True
Dynamic False
Reflection False
Crypto False
Obfuscation False

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.


AuthServiceConnector.java

```

      /*
 * Decompiled with CFR 0_115.
 *
 * Could not load the following classes:
 *  android.content.ComponentName
 *  android.content.ServiceConnection
 *  android.os.Bundle
 *  android.os.Handler
 *  android.os.IBinder
 *  android.os.Message
 *  android.os.Messenger
 *  android.os.RemoteException
 *  android.util.Log
 */
package com.mwr.example.sieve;

import android.content.ComponentName;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

public class AuthServiceConnector
extends Handler
implements ServiceConnection {
```

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.