Droid

Problem: Can you find the flag in this file? You might need this tool.

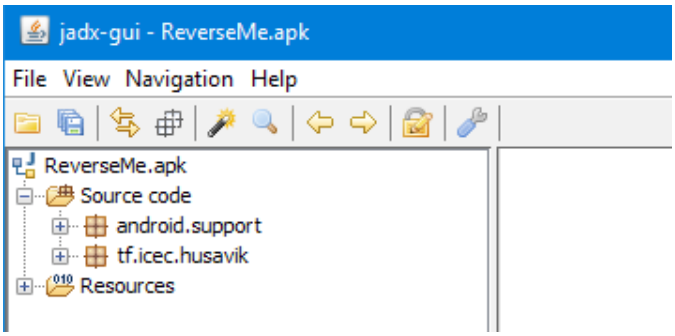
Given: ReverseMe.apk

Hint: Tools are helpful

Introduction: Droid is designed to introduce students to the basics of reverse engineering a very simple Android application. This challenge centers mostly around basic decompilation of .apk files using a tool known as jadx followed by reading the decompiled Java code.

Steps:
(1) Download both the provided tools and the ReverseMe.apk file provided in the challenge package. We will use Jadx due to the simple GUI interface provided. APKTool works on decompilation of an .apk file with a command line interface.

(2) So first things first here, go ahead and launch Jadx - Gui. Decompile the provided .apk by going to file->openfile->reverseme.apk. After we specify the apk to decompile, Jadx will list the decompiled packages in the lefthand side of the interface seen below.

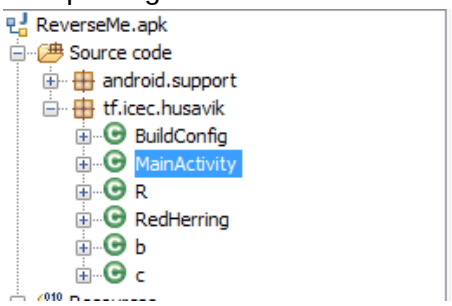


(2) We see two folders named, “Source code” and “Resources”. The first stop we should make when reversing any apk is to take a look at a file called “AndroidManifest.xml”, located under the “Resources” folder. The manifest file contains important information required by the Android system (refer to https://developer.android.com/guide/topics/manifest/manifest-intro.html). A screenshot of this application's manifest file is below:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:"http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="tf.icec.husavik" platfo
  <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="22" />
  <uses-permission android:name="android.permission.INTERNET" />
  <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@drawable/icon" android:allowBackup="true">
    <activity android:label="@string/app_name" android:name="tf.icec.husavik.MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:label="@string/title_activity_red_herring" android:name="tf.icec.husavik.RedHerring" />
  </application>
</manifest>
```

(3) By examining the manifest file, we can learn several key pieces of information. This application's main package is "tf.icec.husavik", this application requires internet privileges, and finally this application's main activity while running is contained in "tf.icec.husavik.MainActivity". Let's shift gears now and see if we can find any interesting information contained in the main package "tf.icec.husavik". This package is contained under the "Source code" tab go ahead and open it up to list the contents of the package.



(4) We can see from the contents of the main package that this is a relatively simple program. Think of the MainActivity class as you would a main() function in a c++ program. While every other class file could be seen similarly to their c++ counterpart if that makes sense. Our next step should be to take a look at the code executed in MainActivity for clues.

```
package tf.icec.husavik;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.StrictMode;
import android.os.StrictMode.ThreadPolicy.Builder;
import android.view.View;
import java.net.ServerSocket;

public class MainActivity extends Activity {
  Thread f0a;
  Thread f1b;
  ServerSocket f2c;

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    StrictMode.setThreadPolicy(new Builder().permitAll().build());
    this.f0a = new Thread(new c(this));
    this.f0a.start();
    this.f1b = new Thread(new b(this));
    this.f1b.start();
  }

  public void red(View view) {
    startActivity(new Intent(this, RedHerring.class));
  }
}
```

Above is a screenshot of the MainActivity class. There's a few things to be learned by examining this code, I'll list them out from top to bottom. First, notice that two Thread variables are created, as well as a ServerSocket variable. The declaration of those variables could indicate that at some point this application could execute additional behavior in a secondary thread while the main application continues to run. Additionally the ServerSocket variable could be used for some form of internet communication.

(5) Let's analyse the behavior of class 'c' the code is below. The only real functionality class 'c' provides is opening a socket on port 6464 and accepting incoming connections while the thread remains active.

```
package tf.icec.husavik;

import java.io.IOException;
import java.net.ServerSocket;

public class c extends Thread {
  final MainActivity f4a;

  public c(MainActivity mainActivity) {
    this.f4a = mainActivity;
  }

  public void run() {
    try {
      this.f4a.f2c = new ServerSocket(6464);
      while (true) {
        this.f4a.f2c.accept();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

(5) Let's take a look at class 'b' now and hope we find something worthwhile, code snippet below. At first glance it seems class 'b' provides similar functionality to that of class 'c'. Class 'b' opens a socket on port 6464 using the localhost ip address 127.0.0.1. After the socket is created the application uses an objectOutputStream to write a base64 encoded string object to the socket.

```
package tf.icec.husavik;

import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class b extends Thread {
  final MainActivity f3a;

  public b(MainActivity mainActivity) {
    this.f3a = mainActivity;
  }

  public void run() {
    try {
      Socket socket = new Socket("127.0.0.1", 6464);
      ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.getOutputStream());
      objectOutputStream.writeObject("ZmxhZ193YWl0X3dhc250X2l0X2RhbHZpawo=");
      objectOutputStream.close();
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

(6) Let's use an online base64 decoder to find out. Decoding the base64 string reveals...