

You have 1 free story left this month. Upgrade for unlimited access.

Morty Sherlocked: Android Application Based CTF Challenge Walkthrough



Saurabh Jain

Sep 27 · 5 min read ★



Morty Sherlocked is a beginner level Android application CTF challenge. It walks us through the basic concepts of Android application security, giving us an amazing experience of analyzing an APK.

The aim of this CTF challenge is to concentrate on the basic flaws which are being founded while performing a security assessment of a mobile application.

Let's take a minute to thank [Moksh](#) for creating this challenge. If someone wants to try and solve the challenge, the link for the CTF can be found [\[here\]](#) and the application can be downloaded from [\[here\]](#).

Tools Used :

adb : command line tool that lets you communicate with device

apktool : command line tool for reverse engineering android applications

jadx-gui : tool for producing Java source code from Android Dex and APK files

Android Studio : official Integrated Development Environment (IDE) for Android app development

Device : Android Device/Android Studio Emulator/Genymotion Emulator

Connecting the device with a USB cable and entering command for checking proper connectivity.

```
adb devices
```

The above command will list down all the connected devices/emulators.

```
$ adb devices
List of devices attached
52032295e8f96377    device
```

The above exhibit shows the list of devices connected to the system

Note : Make sure to connect the android phone with debugging mode enabled for initiating the application installation process.

After downloading the application from the above given link, the application can be installed in device/emulator by a very simple command.

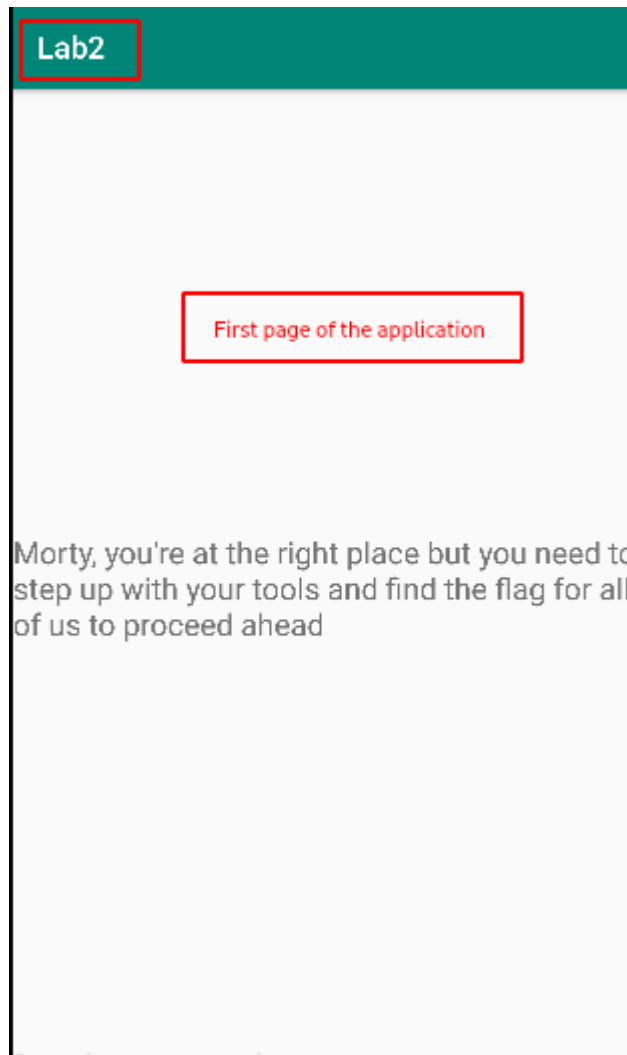
```
adb install <apk-name>
```

```
$ adb install com.cybergym.lab2.apk
```

```
Performing Push Install  
com.cybergym.lab2.apk: 1 file pushed. 6.1 MB/s (1029738 bytes in 0.160s)  
pkg: /data/local/tmp/com.cybergym.lab2.apk  
Success
```

The above exhibit shows that the application has been successfully installed in the device

We can run the application in the device/emulator

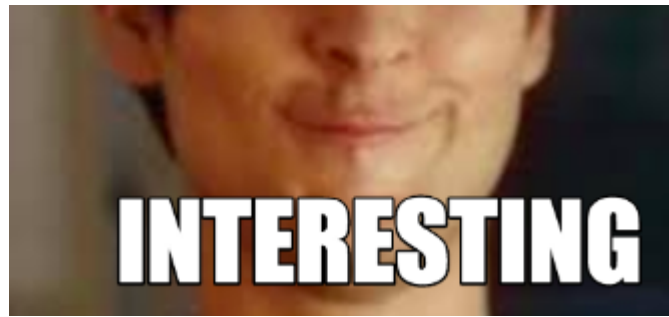


The above exhibit shows that first page while running the application

So it's written...

Morty, you are at the right place but you need to step up with your tools and find the flag for all of us to proceed ahead.





The message is a hint indicating that we need to use APK analysing tools to find the flag. Great!

We have *jadx-gui* in our bucket as an APK analysing tool. Let's reverse engineer the APK.

```
jadx-gui <apk-name>
```

After reverse engineering the APK using *jadx-gui*, we can read the source code of the application and grab the flag.

Now start reading the source code from MainActivity.java

```
package com.moksh.lab2;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        new ja();
        int a = jb.a(this);
        String string = getResources().getString(R.string.key1);
        String string2 = getResources().getString(R.string.key2);
        if (a <= 0) {
            Toast.makeText(this, "Application Tampered", 1).show();
            finishAffinity();
        }
        ja.a(string2, string);
        Toast.makeText(this, "Decrypted: Nothing specified", 1).show();
    }
}
```

The above exhibit shows that the application starts executes code from **MainActivity.java**

*After observing the code we can see there are keys stored named as i.e **string** and **string2**.*

```

public class MainActivity extends AppCompatActivity {
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        new ja();
        int a = jb.a(this);
        String string = getResources().getString(R.string.key1);
        String string2 = getResources().getString(R.string.key2);
        if (a <= 0) {
            Toast.makeText(this, "Application Tampered", 1).show();
            finishAffinity();
        }
        ja.a(string2, string);
        Toast.makeText(this, "Decrypted: Nothing specified", 1).show();
    }
}

```

The above exhibit shows that the two java string objects are being created and value is fed from resource directory

Tracing down the code we were able to find that the keys that are being stored in *res/values/strings.xml*

```

<string name="abc_shareactionprovider_share_with_application">Share with %s</string>
<string name="abc_toolbar_collapse_description">Collapse</string>
<string name="app_name">Lab2</string>
<string name="key1">VCKqrs6ZBX9GUjDYfCnqtoTvMXEyy3eW9tzT7b5hWsA=</string>
<string name="key2">119235873858382</string>
<string name="search_menu_title">Search</string>
<string name="status_bar_notification_info_overflow">999+</string>

```

The above exhibit shows that the two strings are hardcoded and stroed in resource file

Following the code, a function *ja.a(string2,string)* is being observed with those two strings we found in *res/values/strings.xml*.

```

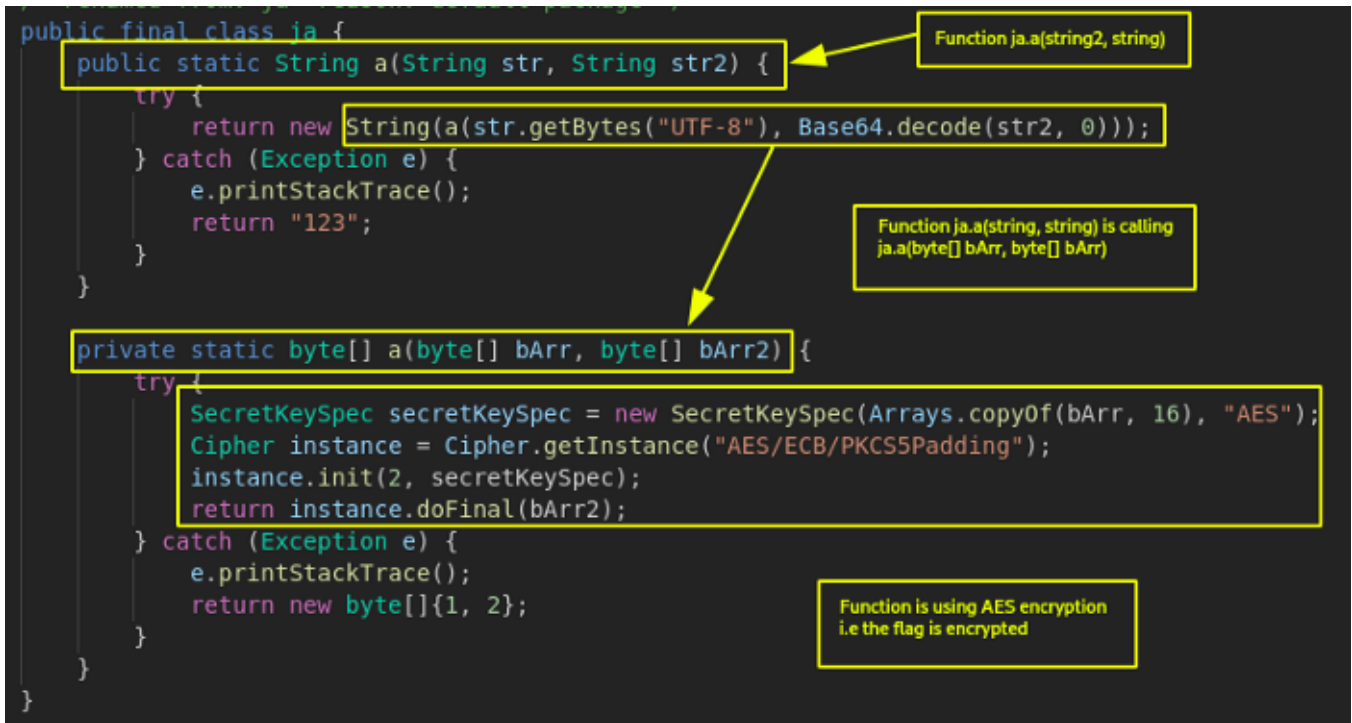
public class MainActivity extends AppCompatActivity {
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        new ja();
        int a = jb.a(this);
        String string = getResources().getString(R.string.key1);
        String string2 = getResources().getString(R.string.key2);
        if (a <= 0) {
            Toast.makeText(this, "Application Tampered", 1).show();
            finishAffinity();
        }
        ja.a(string2, string);
        Toast.makeText(this, "Decrypted: Nothing specified", 1).show();
    }
}

```



The above exhibit shows the function call in the source code

Let's see function `ja.a(string2, string)` declaration.



The above exhibit shows the declaration of the called function in application source code

We can see in `private static byte[] a(byte[] bArr, byte[] bArr2)` function, they are using encryption.

Now, We need to break the encryption...





Fasten your seat belt guys.. The encryption is about to break...

We can replicate these two functions in android studio :

- public static String a(String str, String str2)
- private static byte[] a(byte[] bArr, byte[] bArr2)

```
public class MainActivity extends AppCompatActivity {

    String key1 = "VCKqrs6ZBX9GUjDYfCnqtoTvMXEyy3eW9tzT7b5hWsA=";
    String key2 = "119235873858382";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String flag = a(key2, key1);
        System.out.println(" Flag : " + flag);
    }

    public static String a(String str, String str2) {
        try {
            return new String(a(str.getBytes( charsetName: "UTF-8"), Base64.decode(str2, flags: 0)));
        } catch (Exception e) {
            e.printStackTrace();
            return "123";
        }
    }

    private static byte[] a(byte[] bArr, byte[] bArr2) {
        try {
            SecretKeySpec secretKeySpec = new SecretKeySpec(Arrays.copyOf(bArr, newLength: 16), algorithm: "AES");
            Cipher instance = Cipher.getInstance("AES/ECB/PKCS5Padding");
            instance.init( opmode: 2, secretKeySpec);
            return instance.doFinal(bArr2);
        } catch (Exception e) {
            e.printStackTrace();
            return new byte[]{1, 2};
        }
    }
}
```

The above exhibit is taken from android studio where the code is being replicated from the application

After replicating the above two functions in android studio we tried printing the flag value in application logcat.

Output in Logcat

```
? W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter androidx.vec
com.example.cybergym2 I/art: Rejecting re-init on previously-failed class java.lang.
com.example.cybergym2 I/art: Rejecting re-init on previously-failed class java.lang.
com.example.cybergym2 I/System.out: Flag : cygym3{False_flag}
com.example.cybergym2 D/OpenGLESRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
```

```
com.example.cybergymLab2 D/libEGL: loaded /system/lib/egl/libEGL_emulation.so  
com.example.cybergymLab2 D/libEGL: loaded /system/lib/egl/libGLESv1_CM_emulation.so  
com.example.cybergymLab2 D/libEGL: loaded /system/lib/egl/libGLESv2_emulation.so  
com.example.cybergymLab2 I/OpenGLRenderer: Initialized EGL, version 1.4
```

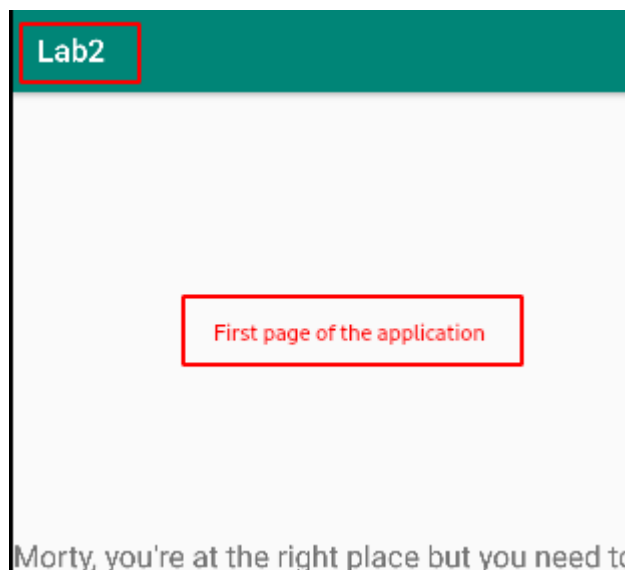
The above exhibit shows the output in application logs

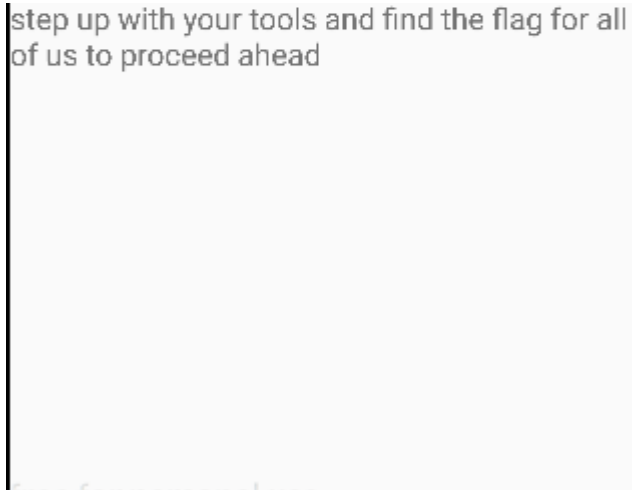
FALSE FLAG... ? WHAAAT !



Congratulations Guys !! We all were part of the Rabbit hole.

Let's go to the beginning of the application and read the hint again...





The above exhibit shows the beginning of the application

Is there any other APK analysing tool ? Wait! we have *apktool* as well.

*The basic difference between **apktool** and **jadx-gui** is that, we get smali code while analysing APK using **apktool** and on the other side we get java source code using **jadx-gui**.*

Let's reverse engineer the application using *apktool* and try to analyse application source code again.

We can use a very simple command for that

```
apktool d <apk-name>
```

OR

```
apktool d <apk-name> -o <new-folder-name>
```

Note : Here “d” stands for decompiling the application.

```
Lab2-MortySherlocked$ apktool d
com.cybergym.lab2.apk -o MortySherlocked
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.4.1 on com.cybergym.lab2.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/saurabhjain/.local/share/apktool/
framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

The above exhibit shows that the application is reverse engineered using apktool

After reverse engineering the APK, analysing all the XML files in the application source code.

```
-----
    <item>35</item>
  </string-array>
  <string-array name="gender">
    <item>Gender</item>
    <item>Male</item>
    <item>Female</item>
    <item>cygym3{You_did_it_again_morty}</item>
  </string-array>
</resources>
```

The above exhibit shows that the flag was stored in resource directory

So the flag was hidden in the resource directory **/res/values/arrays.xml**

BINGO!! Morty! You did it again..

Takeaways

Learned how to reverse engineer android application.

Learned how to read the application source code.

*Learned breaking encryption and never to use weak encryption algorithm such as
“AES/ECB/PKCS5Padding”*

Never hard code data in application source code.

Android Ctf Writeup Reverse Engineering Security Pentesting

About Help Legal

Get the Medium app



