



## CAPTURE THE FLAG – BASIC ANDROID RE1

Autor: ETR00M

Github: <https://github.com/ETR00M/>

Linkedin: <https://www.linkedin.com/in/ls-anderson/>

Link da Challenge: <https://ctflearn.com/challenge/962>

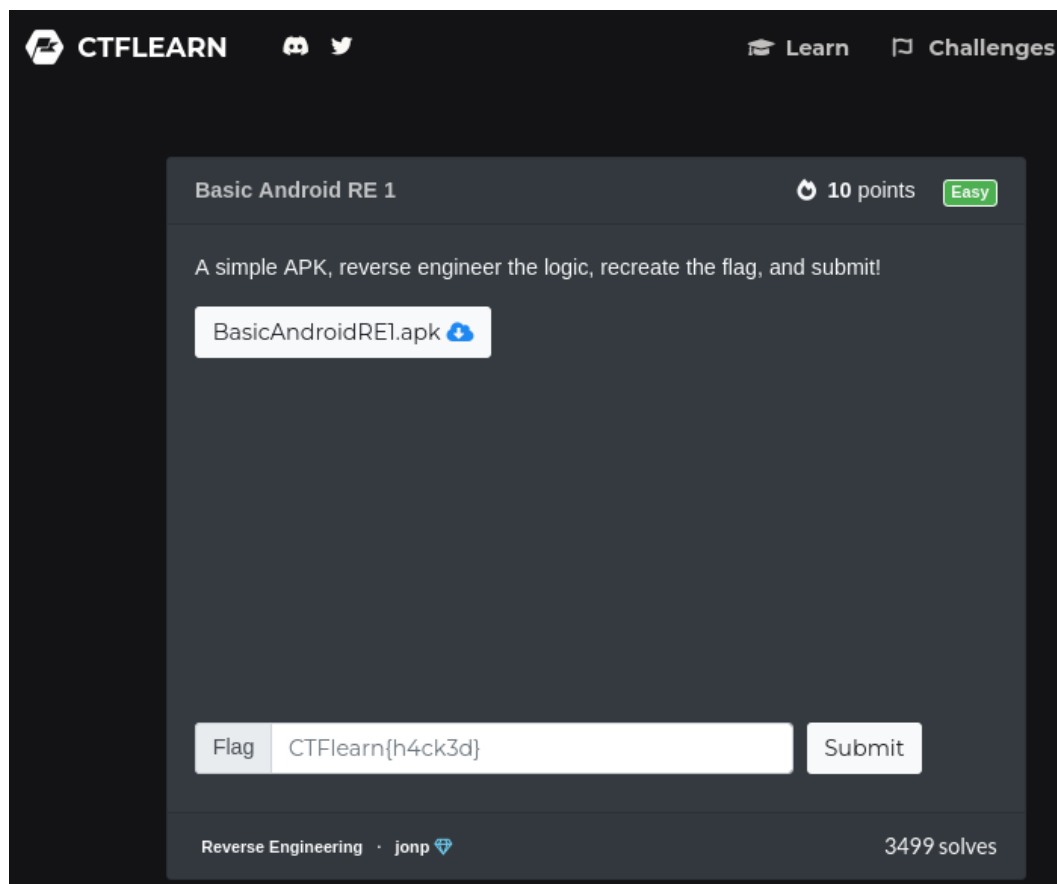
Nível: fácil;

Categoria: Reverse Engineering;

Tag: comandos Linux (apt, apt-get), ferramenta (jadx), desenvolvimento de software (textbox, if, concatenar), Hash (MD5), pensamento linear.



Neste desafio do **CTFLearn** precisamos analisar um aplicativo Android (**.apk**) disponibilizado para download pelo autor da *challenge*, entender a lógica de seu funcionamento e recriar a *flag* a partir do seu código fonte.





Primeiramente precisamos visualizar as propriedades do arquivo, para verificar que tipo de arquivo estamos lidando, neste caso trata-se de um aplicativo Android “**APK – Android Package**”:

```
(kali㉿kali)-[~/Downloads]
$ ls -la
total 1592
drwxr-xr-x  2 kali kali    4096 Feb 22 19:02 .
drwx----- 25 kali kali    4096 Feb 22 18:49 ..
-rw-r--r--  1 kali kali 1621059 Feb 22 19:00 BasicAndroidRE1.apk

(kali㉿kali)-[~/Downloads]
$ file BasicAndroidRE1.apk
BasicAndroidRE1.apk: Android package (APK), with AndroidManifest.xml, with
APK Signing Block
```

Como sugerido pelo próprio autor da *challenge* precisaremos realizar uma engenharia reversa no aplicativo móvel, para isso utilizarei a ferramenta *jadx* para “descompilar” o software e obter seu código fonte, a ferramenta está disponível tanto para Windows quanto para Linux podendo ser baixada a partir do repositório no Github ou via gerenciador de pacotes no Linux: (<https://github.com/skylot/jadx>).

Caso você não possua conhecimentos referente a instalação de pacotes via terminal Linux, teoria básica de funcionamento de aplicações Android, lógica de programação ou conhecimento básico de qualquer linguagem (if, textbox), recomendo o estudo destes assuntos antes de seguir com o *Writeup*.

Dicas de materiais para estudo:

- [https://www.youtube.com/watch?v=rRH\\_UzPkEh4&ab\\_channel=Inform%C3%A1ticaNaPedra](https://www.youtube.com/watch?v=rRH_UzPkEh4&ab_channel=Inform%C3%A1ticaNaPedra)
- [https://www.youtube.com/watch?v=XXDYw6387Vo&ab\\_channel=Certifica%C3%A7%C3%A3oLinux](https://www.youtube.com/watch?v=XXDYw6387Vo&ab_channel=Certifica%C3%A7%C3%A3oLinux)
- [https://www.youtube.com/watch?v=5wIP1\\_iwxHg&ab\\_channel=RedTeamBrasil](https://www.youtube.com/watch?v=5wIP1_iwxHg&ab_channel=RedTeamBrasil)
- [https://www.youtube.com/watch?v=gdgfl5evAUI&ab\\_channel=WhiteboardCrypto-Portugu%C3%AAs](https://www.youtube.com/watch?v=gdgfl5evAUI&ab_channel=WhiteboardCrypto-Portugu%C3%AAs)

Comando: **apt install jadx**

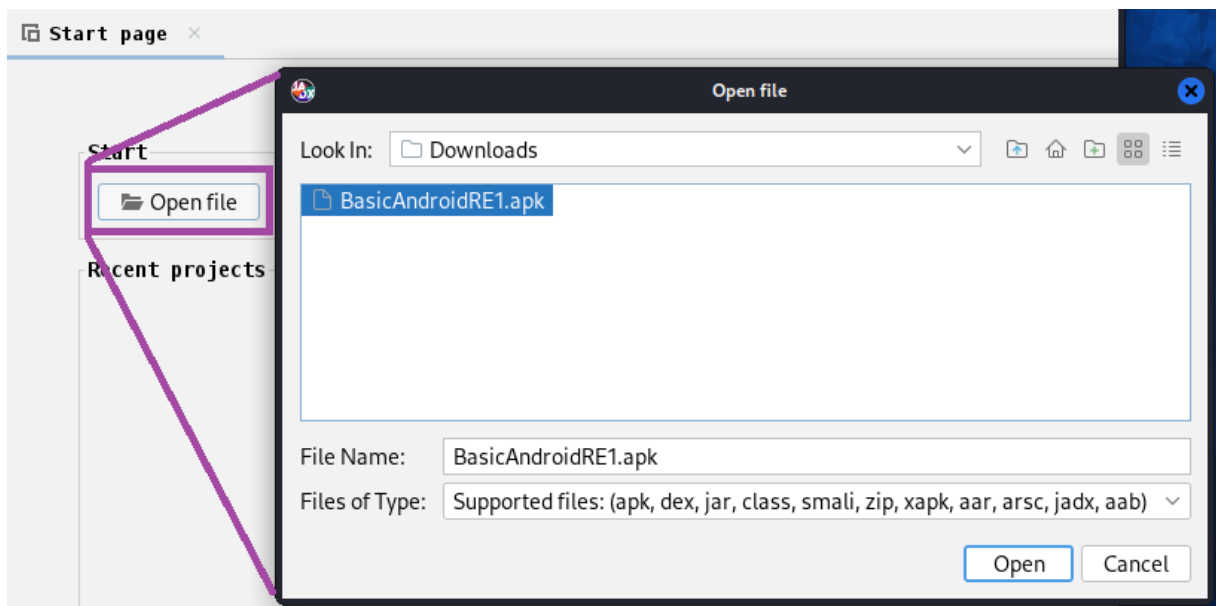


```
$ sudo apt install jadx
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  binutils-mingw-w64-i686 binutils-mingw-w64-x86-64 bluez-firmware cython3 debtags
  dh-elpa-helper docutils-common firmware-atheros firmware-brcm80211
  firmware-intel-sound firmware-iwlwifi firmware-libertas firmware-realtek
  firmware-sof-signed firmware-ti-connectivity firmware-zd1211 gcc-mingw-w64-base
  gcc-mingw-w64-i686-win32 gcc-mingw-w64-i686-win32-runtime gcc-mingw-w64-x86-64-win32
```

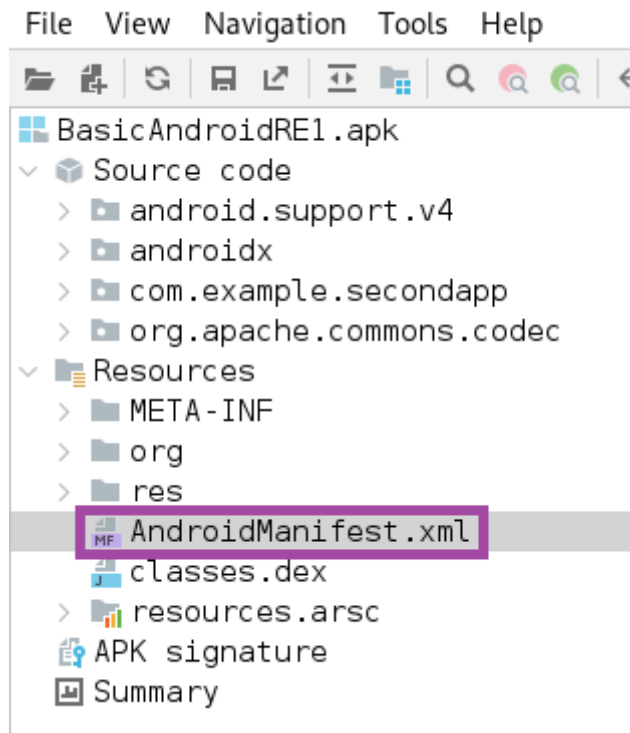
Comando: **jadx-gui**

```
(kali@kali)-[~/Downloads]
$ jadx-gui
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

Após execução da ferramenta *jadx* iremos localizar o pacote Android que será analisado (*BasicAndroidRE1.apk*) e abri-lo:



Todo o conteúdo que foi possível “descompilar” será apresentado a esquerda da tela, como sabemos, o Manifesto é um arquivo que contém as características de um aplicativo em formato XML, por isso como ponto de partida vamos verificar se podemos coletar quaisquer informações sobre o funcionamento do sistema a partir dele que nos dê um caminho até a *flag*:

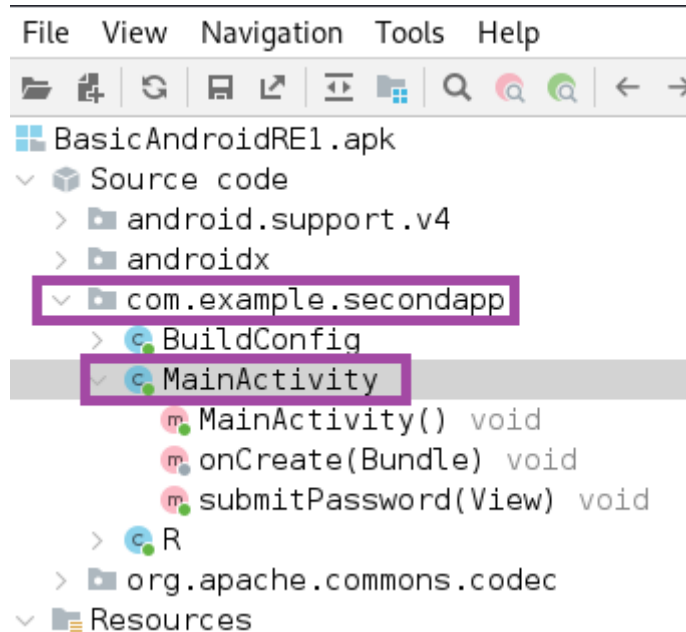


No conteúdo do arquivo “**AndroidManifest.xml**” podemos identificar que a atividade principal do software `<activity>` a ser executada quando o aplicativo for iniciado pode ser localizado no caminho: “*com.example.secondapp.mainActivity*”:

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0">
7   <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="29" />
11   <application android:theme="@style/AppTheme" android:label="@string/app_name">
19     <activity android:name="com.example.secondapp.MainActivity">
20       <intent-filter>
21         <action android:name="android.intent.action.MAIN" />
23         <category android:name="android.intent.category.LAUNCHER" />
20       </intent-filter>
19     </activity>
11   </application>
2 </manifest>
```



Portanto iremos verificar o conteúdo deste arquivo:




Avaliando o código fonte da *MainActivity* veremos que após o usuário digitar um texto em uma *EditText*, existe uma estrutura condicional (*if*) que compara o *Hash* MD5 do valor digitado com o *Hash hard-code* “**B74DEC4F39D35B6A2E6C48E637C8AEDB**”, também MD5. Caso as duas *Hashes* sejam iguais uma mensagem de sucesso juntamente com a *flag* é mostrada:

```
MainActivity
1 package com.example.secondapp;
2
3 import android.os.Bundle;
4 import android.view.View;
5 import android.widget.EditText;
6 import android.widget.TextView;
7 import androidx.appcompat.app.AppCompatActivity;
8 import org.apache.commons.codec.digest.DigestUtils;
9
10 /* loaded from: classes.dex */
11 public class MainActivity extends AppCompatActivity {
12     /* JADX INFO: Access modifiers changed from: protected */
13     @Override // androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, androidx.core.app.ComponentActivity,
14     public void onCreate(Bundle bundle) {
15         super.onCreate(bundle);
16         setContentView(R.layout.activity_main);
17     }
18
19     public void submitPassword(View view) {
20         EditText editText = (EditText) findViewById(R.id.editText2);
21         if (DigestUtils.md5Hex(editText.getText().toString()).equalsIgnoreCase("b74dec4f39d35b6a2e6c48e637c8aedb")) {
22             ((TextView) findViewById(R.id.textView)).setText("Success! [REDACTED] + editText.getText().toString() + [REDACTED]");
23         }
24     }
25 }
```

Sendo assim, primeiro precisamos descobrir qual o texto que gerou a *Hash* encontrada no código fonte, para isso podemos buscar no navegador por “**MD5 decrypt**” para encontrar um decodificador online, neste desafio utilizarei o site (<https://md5hashing.net/>).



**Md5 value**  
Reversed hash value

 **Copy Value**

**Blame this record**

Agora que sabemos qual o texto plano digitado pelo usuário satisfaz a condição verdadeira da condição, podemos seguir analisando o restante do código fonte:

```
public void submitPassword(View view) {  
    EditText editText = (EditText) findViewById(R.id.editText2);  
    if (DigestUtils.md5Hex(editText.getText().toString()).equalsIgnoreCase("b74dec4f39d35b6a2e6c48e637c8aedb")) {  
        ((TextView) findViewById(R.id.textView)).setText("Success! " + editText.getText().toString() + "  
    }  
}
```

Caso as duas *Hashes* sejam iguais é retornado o início da *flag* concatenado com o texto puro digitado pelo usuário, concatenado novamente com outra *string* que compõe a parte final da *flag*.

Após remontar com sucesso a *flag* podemos submetê-la na plataforma do **CTFLearn** e completarmos o desafio:



## Basic Android RE1 ✓

🔥 10 points

Easy

A simple APK, reverse engineer the logic, recreate the flag, and submit!

BasicAndroidRE1.apk 

Flag

Solved

Reverse Engineering · jonp 

3504 solves