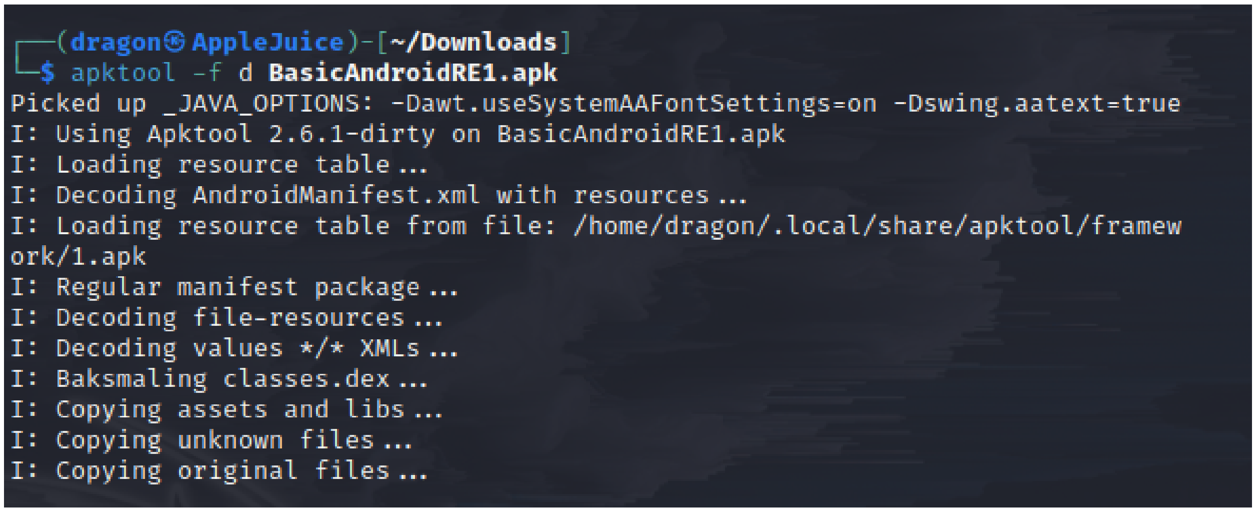
This challenge is about reverse engineering an android app to find flag
Used: apktool, d2j aka Dex-to-Jar, some kind of md5 decoder - I used *md5online.org*

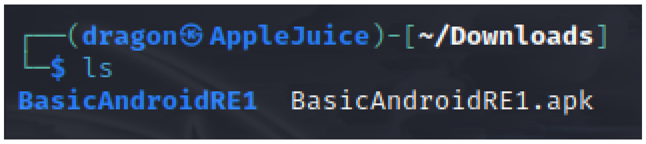
First thing to do is Decompile the archive .apk

- apktool -f d baBasicAndroidRE1.apk to have apktool deconstruct the binaries



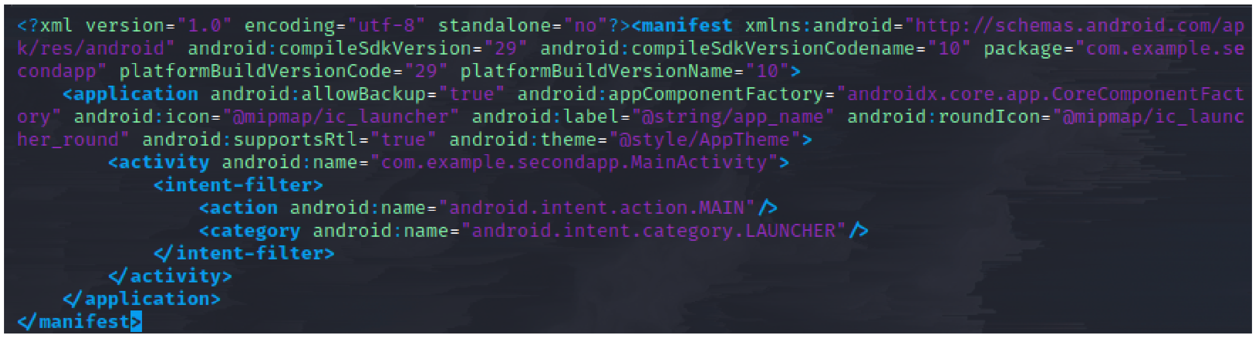
```
┌──(dragon㉿AppleJuice)-[~/Downloads]
└─$ apktool -f d BasicAndroidRE1.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.6.1-dirty on BasicAndroidRE1.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/dragon/.local/share/apktool/framew
ork/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...
```

Now in file explorer you'll see code files



```
┌──(dragon㉿AppleJuice)-[~/Downloads]
└─$ ls
BasicAndroidRE1  BasicAndroidRE1.apk
```

Inside the newly created directory (which is from deconstructing the APK) we have an AndroidManifext.xml which can be viewed inside of vim.



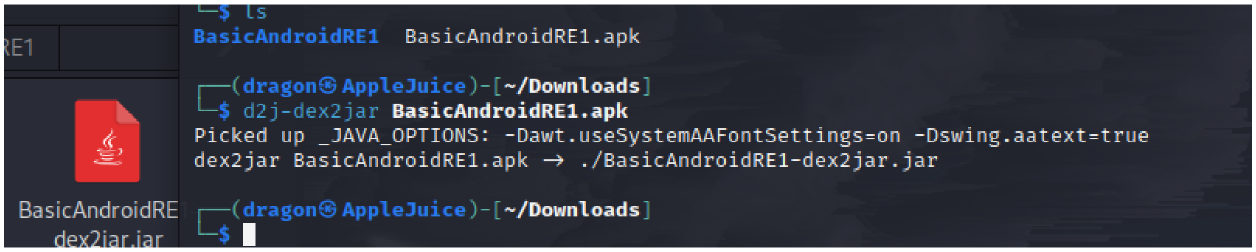
```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/ap
k/res/android" android:compileSdkVersion="29" android:compileSdkVersionCodename="10" package="com.example.se
condapp" platformBuildVersionCode="29" platformBuildVersionName="10">
    <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFact
ory" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launc
her_round" android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:name="com.example.secondapp.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

This is handy as you can review the permissions and configurations of the android app

Extracting Java Code

Inside the original apk you'll see a .dex, which houses all the compiled java code. The below command will allow you to decompile this code

- D2j-dex2jar BasicAndroidRE1.apk



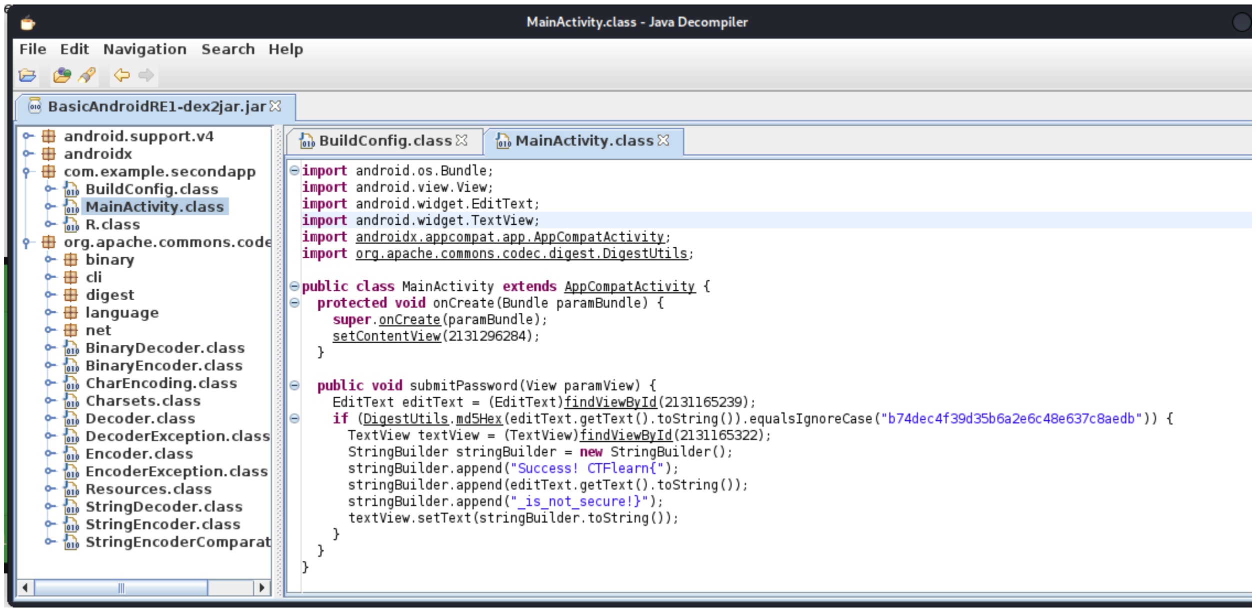
```
└─$ ls
BasicAndroidRE1  BasicAndroidRE1.apk

┌──(dragon㉿AppleJuice)-[~/Downloads]
└─$ d2j-dex2jar BasicAndroidRE1.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar BasicAndroidRE1.apk -> ./BasicAndroidRE1-dex2jar.jar

┌──(dragon㉿AppleJuice)-[~/Downloads]
└─$
```

As we can see, it creates a .jar. These file types are java files that can be viewed with jd-gui!

Just Drag the .jar into the jd-gui tool and you'll have a java code in human readable format

Analyzing the Java In Human Readable Format!

From the manifest file, we know the package is contained inside the *com.example.secondapp* location. By looking in MainActivity.class we can start to see pieces fall into place.

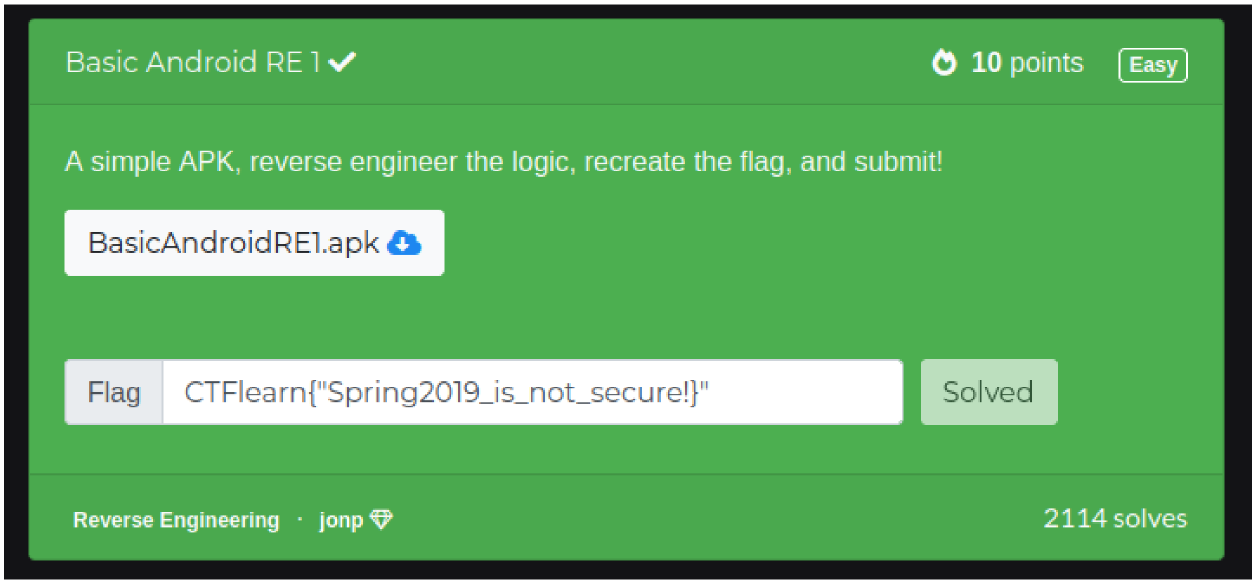


```java
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import org.apache.commons.codec.digest.DigestUtils;

public class MainActivity extends AppCompatActivity {
  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2131296284);
  }

  public void submitPassword(View paramView) {
    EditText editText = (EditText)findViewById(2131165239);
    if (DigestUtils.md5Hex(editText.getText().toString()).equalsIgnoreCase("b74dec4f39d35b6a2e6c48e637c8aedb")) {
      TextView textView = (TextView)findViewById(2131165322);
      StringBuilder stringBuilder = new StringBuilder();
      stringBuilder.append("Success! CTFlearn{");
      stringBuilder.append(editText.getText().toString());
      stringBuilder.append("_is_not_secure!}");
      textView.setText(stringBuilder.toString());
    }
  }
}
```

Here we need to be a little observant. By reading through the program we can see several string *appends*. In the second append command; rather than a string literal, there is an object instead. Several lines upwards we see an md5hex, when you copy and paste that string beginning with "b74dec…" into the decoder, you'll get the missing part of the string

Found : Spring2019
(hash = b74dec4f39d35b6a2e6c48e637c8aedb)

By combining all the pieces of the string we get:
CTFlearn{"Spring2019_is_not_secure!}"
Try that into ctflearn.com and voila! Another challenge has been completed:


Basic Android RE 1 ✓ 10 points Easy

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

BasicAndroidRE1.apk

Flag CTFlearn{"Spring2019_is_not_secure!}" Solved

Reverse Engineering · jonp 2114 solves