Introduction

This is a walkthrough for the injuredandroid CTF we did as part of a mobile security training. It gives a good overview of the static and dynamic analysis techniques you have at your disposal to test the security of a mobile application.

The ainjuredandroidctf APK can be found here:

https://github.com/B3nac/InjuredAndroid

Preparation:

Required: Rooted device or emulator (We use Android 8 on the emulator)

On device (emulator):

Install injuredandroid via adb Install DrozerAgent APK via adb

on Kali:

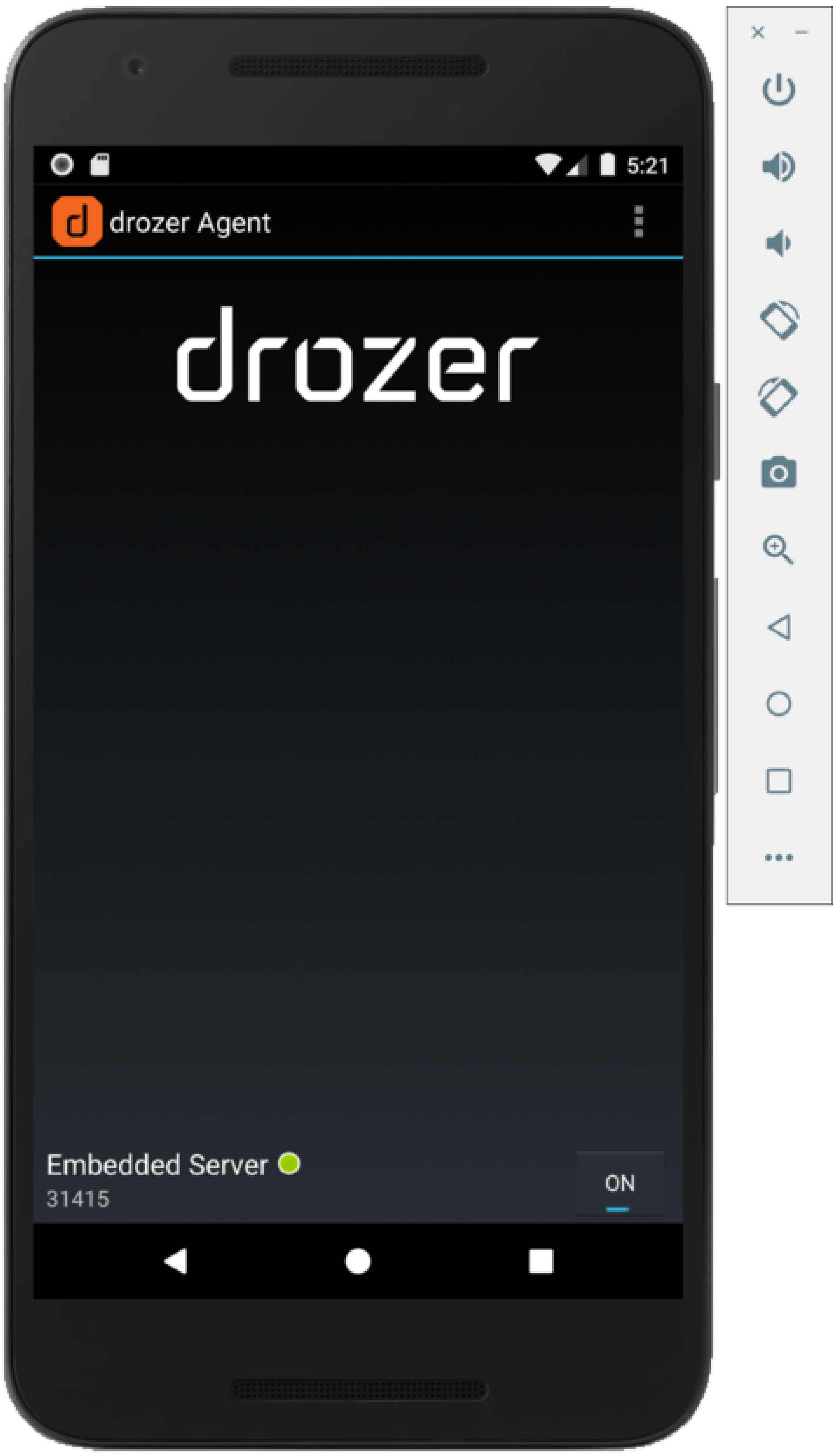
Install jadx-gui to examine source code Install FridaServer Install apktool Install drozer (note, drozer may throw an error about java_path, install the deb file instead) (https://github.com/FSecureLABS/drozer/issues/346)

Decompile apk with apktool

```
apktool d -o APK injuredandroid.apk
```

Will create a directory APK with decompiled resources and smali code

For drozer: Start the agent on device:



On Kali:

```
adb forward tcp:31415 tcp:31415
drozer console connect
```

Gives:

```
Selecting 3c586d02566b0458 (Google Android SDK built for x86 8.0.0)

            ..                    ..:.
           ..o..                  .r..
            ..a..  . ....... .  ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotectorandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v2.4.4)
```

To use Frida (on 32 bit emulated device. for real maybe use 64 bit version)

```
xz -d < frida-server-12.8.6-android-x86.xz > frida-server
adb push frida-server /data/local/tmp/
adb shell
cd /data/local/tmp
chmod 755 frida-server
./frida-server &
```

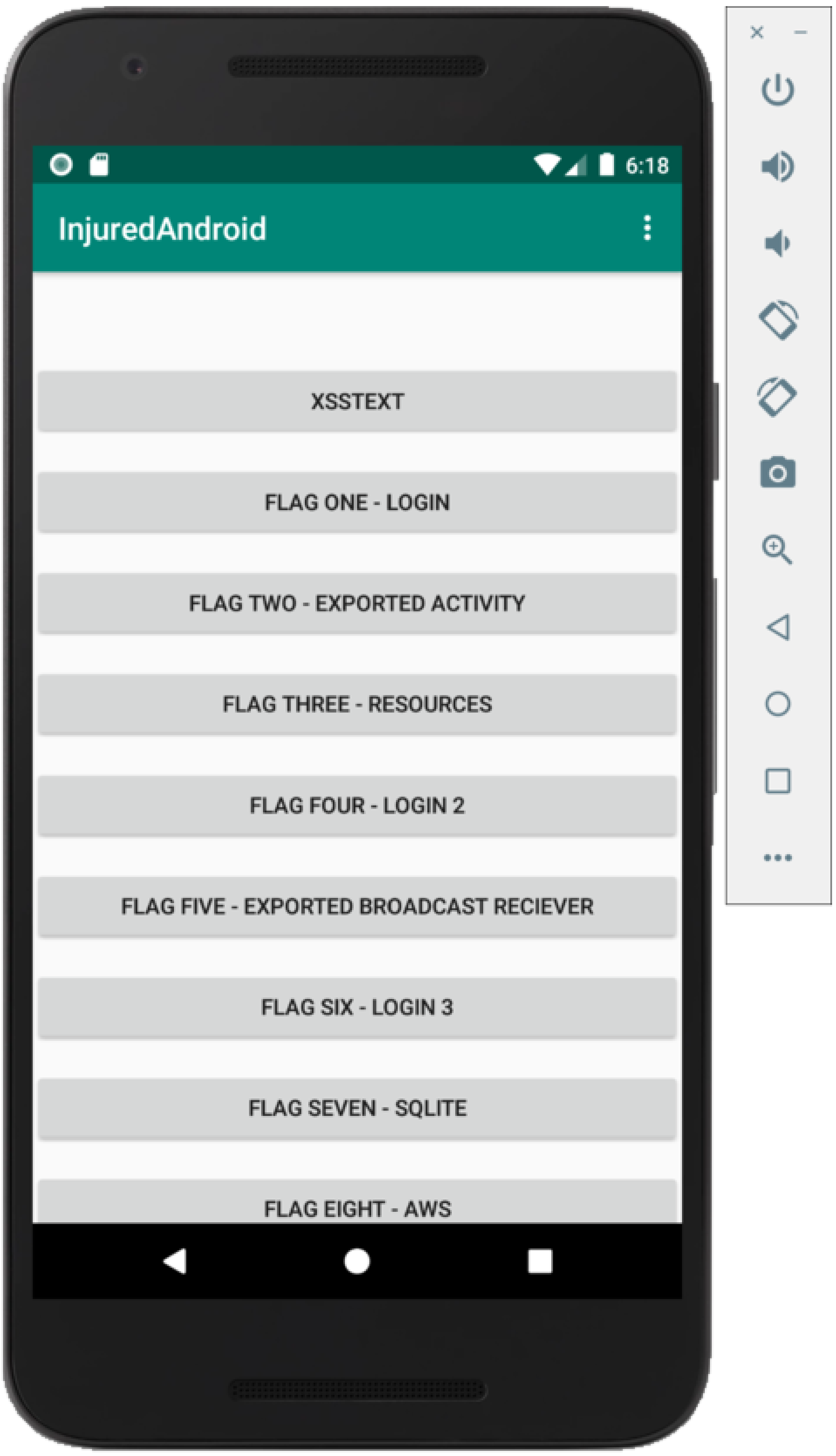
switch to another terminal:

```
adb forward tcp:27042 tcp:27042
frida-ps -H 127.0.0.1
```

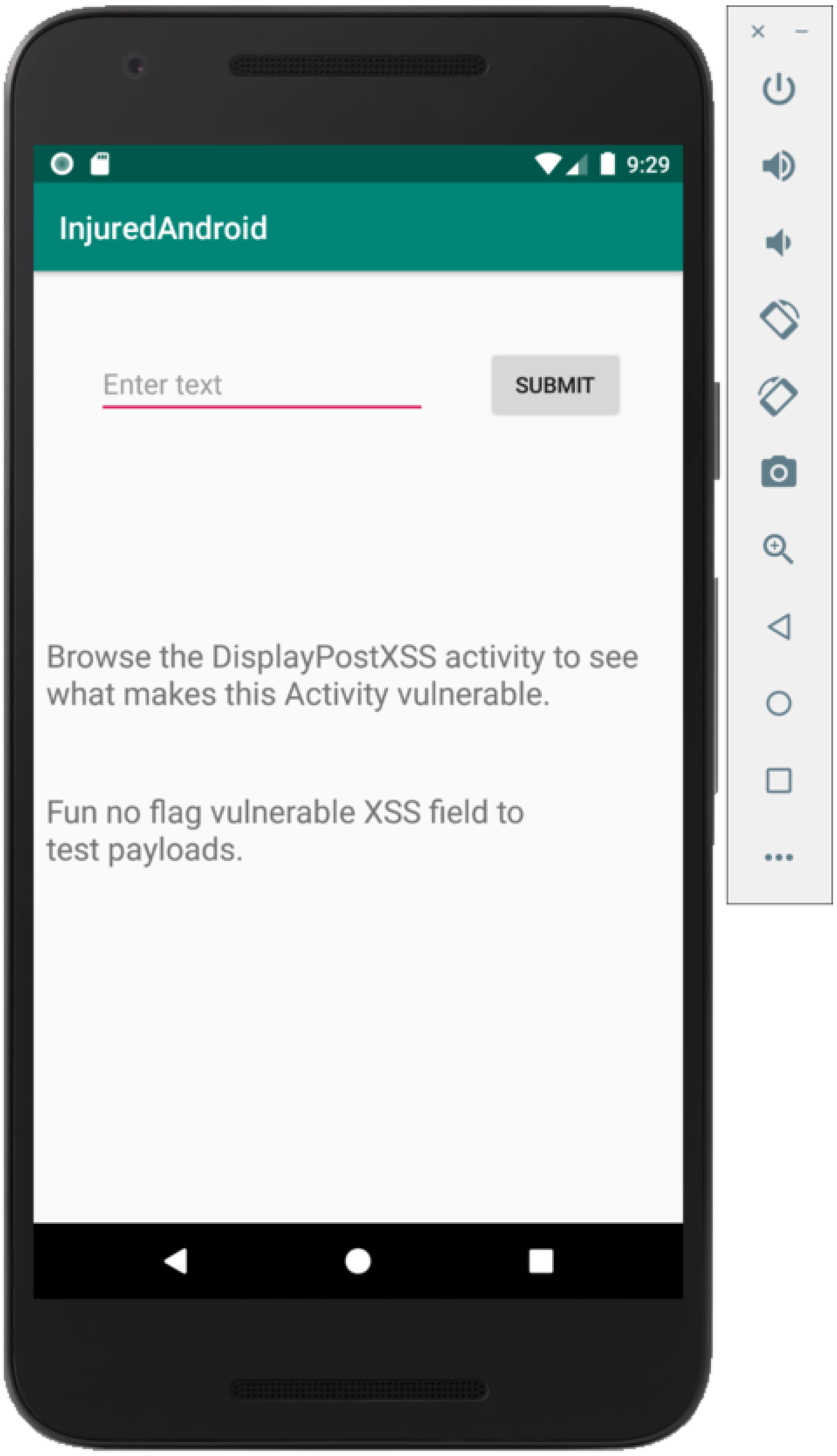
This should list all processes on the mobile device:

```
  PID  Name
-----  ---------------------------------------------------
 6408  adbd
 1399  android.hardware.audio@2.0-service
 1566  android.hardware.biometrics.fingerprint@2.1-service

...

 1675  dhcpserver
 1535  drmserver
 8944  frida-server
 1556  gatekeeperd
 1411  healthd
 1673  hostapd
 1342  hwservicemanager
    1  init
 1536  installd
 1560  ip6tables-restore
 1554  iptables-restore
 1670  ipv6proxy
 1537  keystore
 1415  lmkd
 1417  logcat
 8946  logcat
 1340  logd
 6411  mdnsd
 1547  media.codec
 1539  media.extractor
 1540  media.metrics
 1538  mediadrmserver
 1541  mediaserver
 1414  misc-pipe
 1542  netd
 1413  qemu-props
 1548  rild
 1341  servicemanager
 1419  sh
 1662  sh
 1663  sleep
 1543  storaged
 1416  surfaceflinger
 1689  system_server
 1559  tombstoned
 1049  ueventd
 1343  vndservicemanager
 1345  vold
 1941  webview_zygote32
 1544  wificond
 1989  wpa_supplicant
 1532  zygote
```

Start the app on the device:



Challenge 0 (xss):

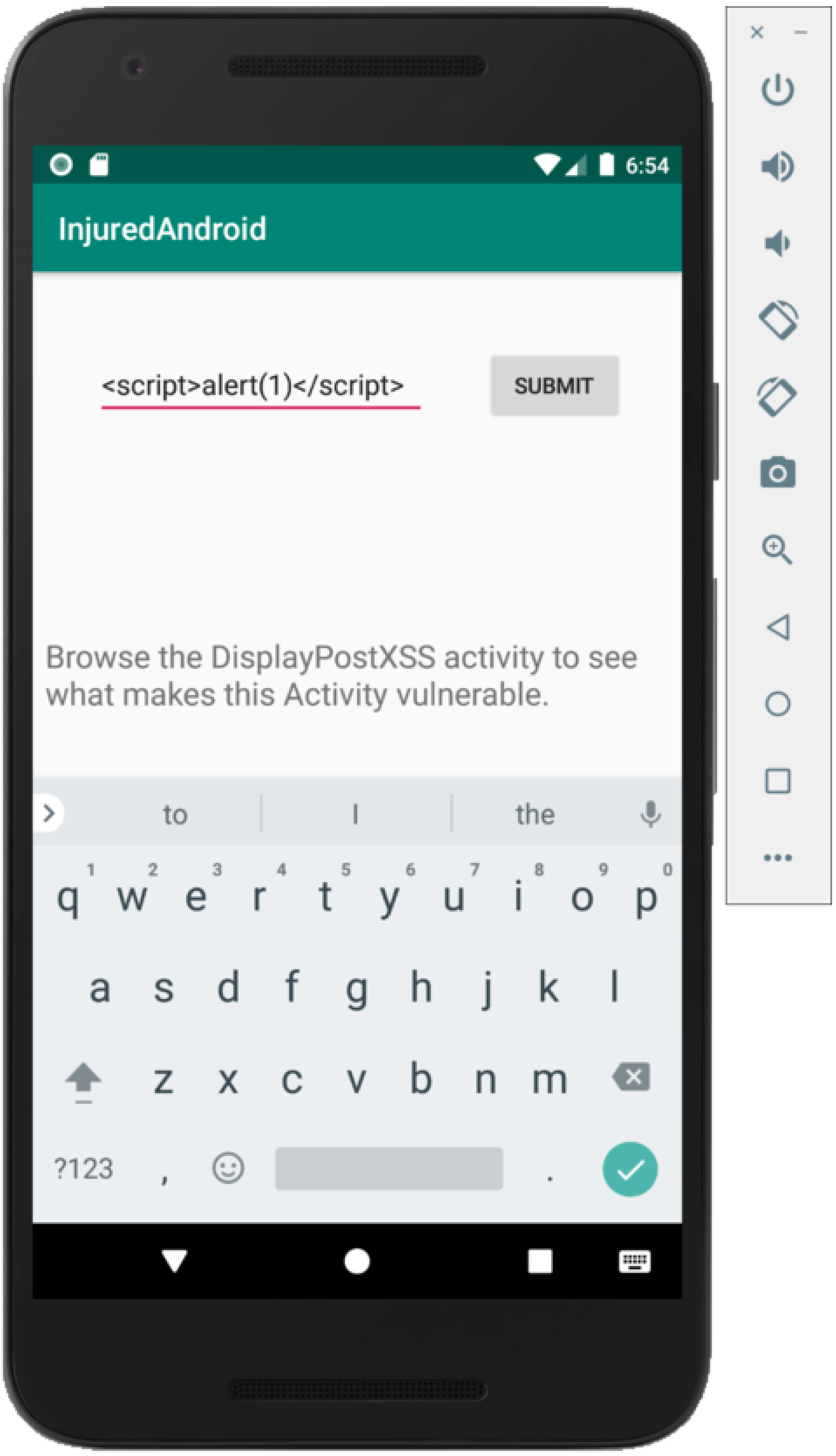


Browse the DisplayPostXSS activity to see what makes this Activity vulnerable.

Fun no flag vulnerable XSS field to test payloads.

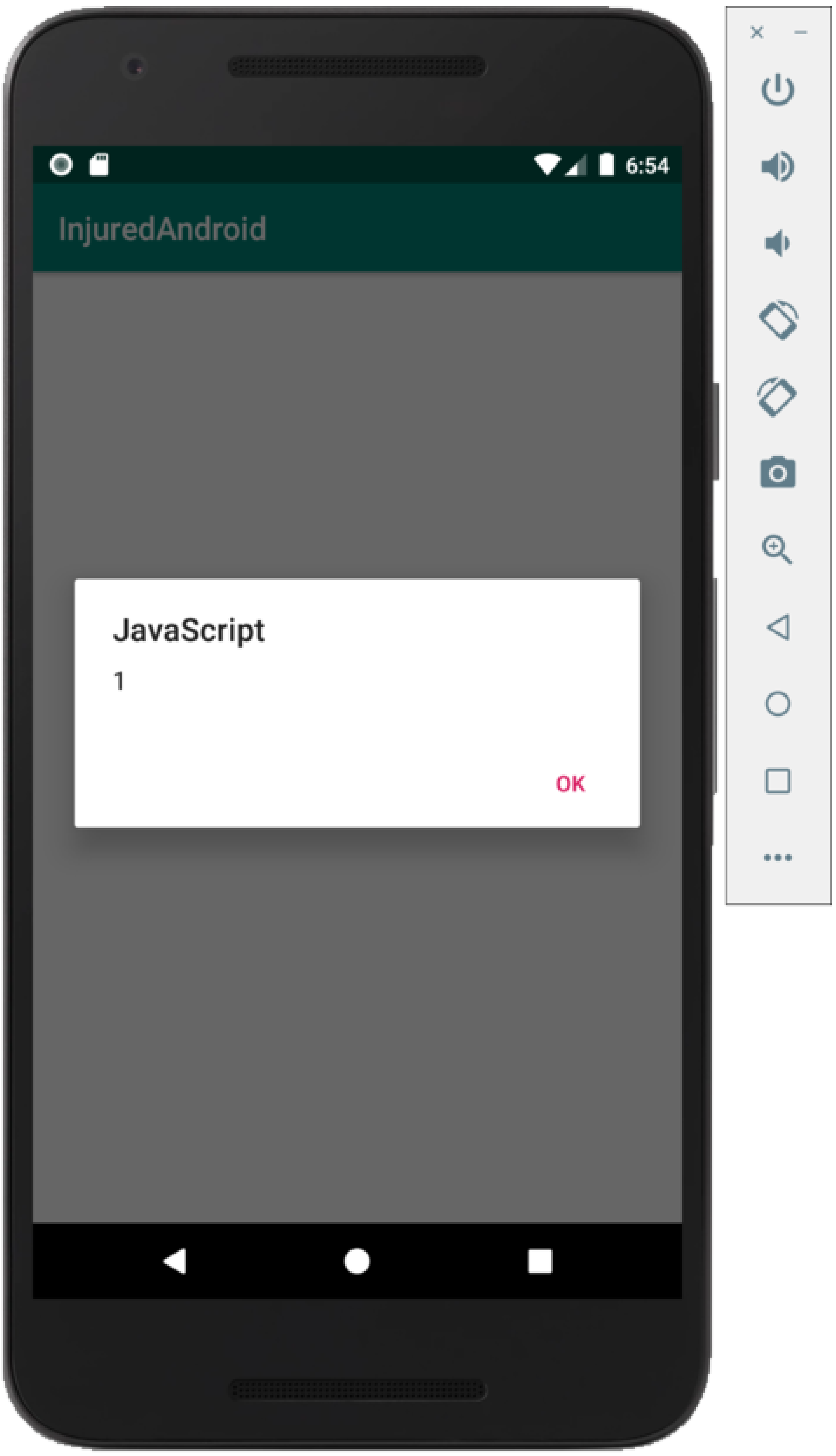
Enter

```
<script>alert('Android');</script>
```



Browse the DisplayPostXSS activity to see what makes this Activity vulnerable.

pop an alert:



JavaScript

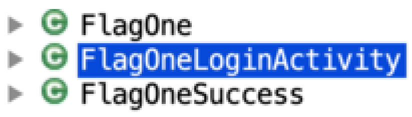
1

OK

Done

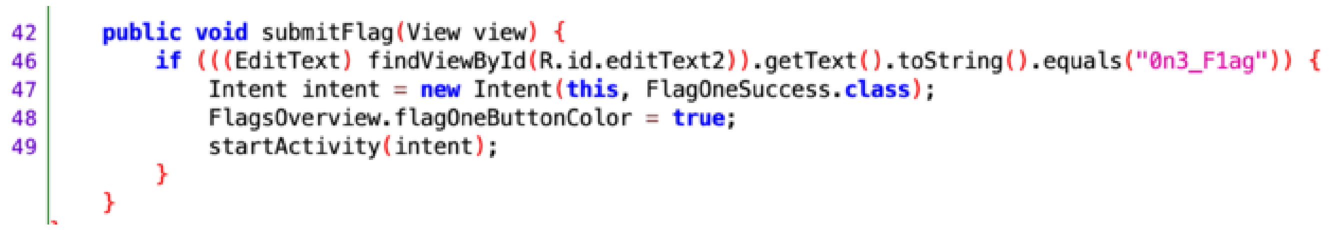
Challenge 1:

Use JADX to decompile the APK to Java.



```
▶ FlagOne
▶ FlagOneLoginActivity
▶ FlagOneSuccess
```

Read the source of FlagOneLoginActivity, find the `toString.equals`:



```
    public void submitFlag(View view) {
        if (((EditText) findViewById(R.id.editText2)).getText().toString().equals("0n3_F1ag")) {
            Intent intent = new Intent(this, FlagOneSuccess.class);
            FlagsOverview.flagOneButtonColor = true;
            startActivity(intent);
        }
    }
```

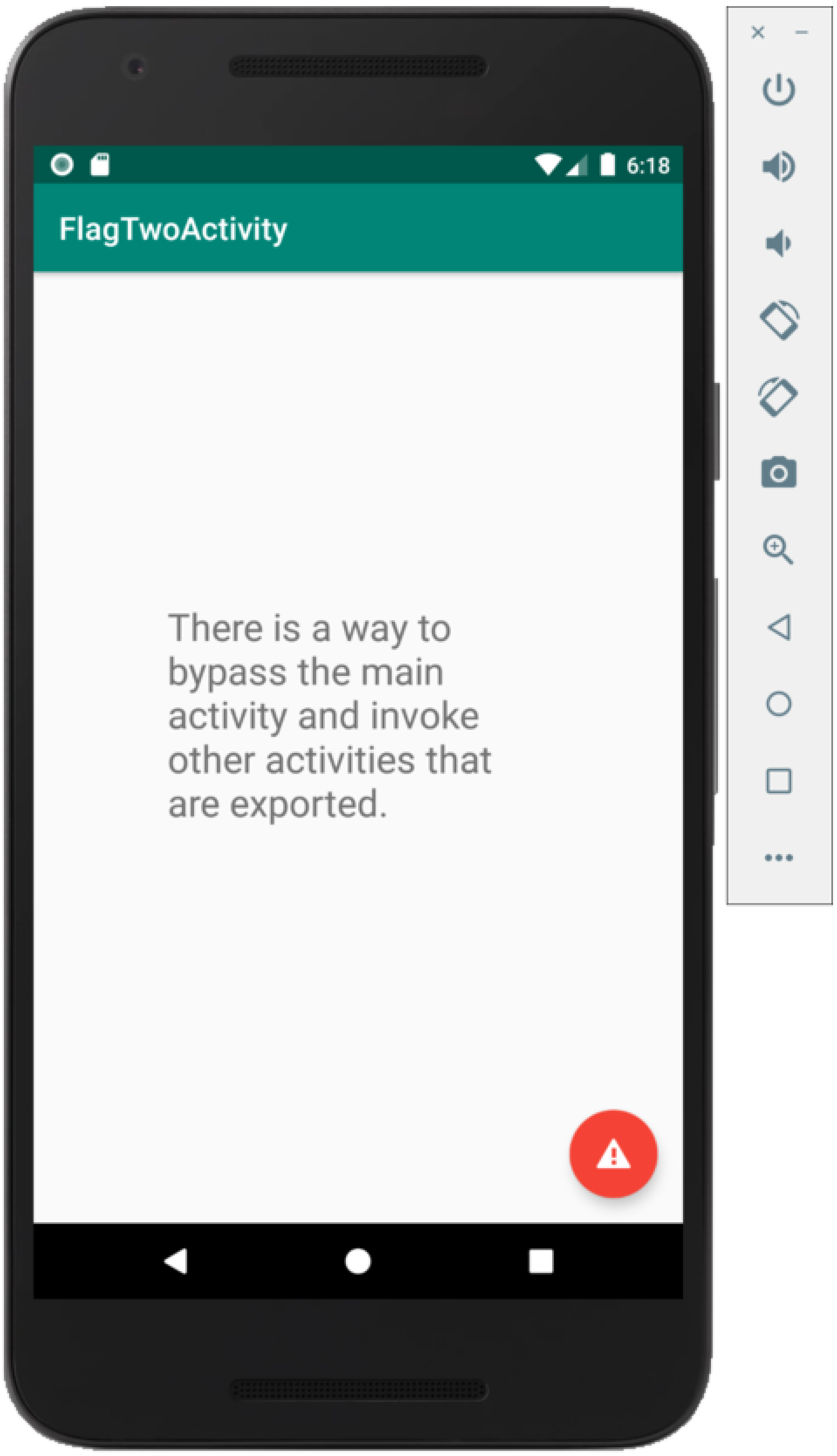
Enter the flag.

flag: 0n3_F1ag

Done

Challenge 2:

This flag has no buttons or entry fields:



There is a way to bypass the main activity and invoke other activities that are exported.

Hint says to trigger an activity.

List activities:

First find the name of the package (not identical to apk filename):

Using ADB:

```
adb shell pm list packages | grep -i injured

package:b3nac.injuredandroid
```

Then, to list all activities:

```
adb shell dumpsys package | grep -i "b3nac.injuredandroid" | grep Activity
```

```
dc78b20 b3nac.injuredandroid/.MainActivity
```

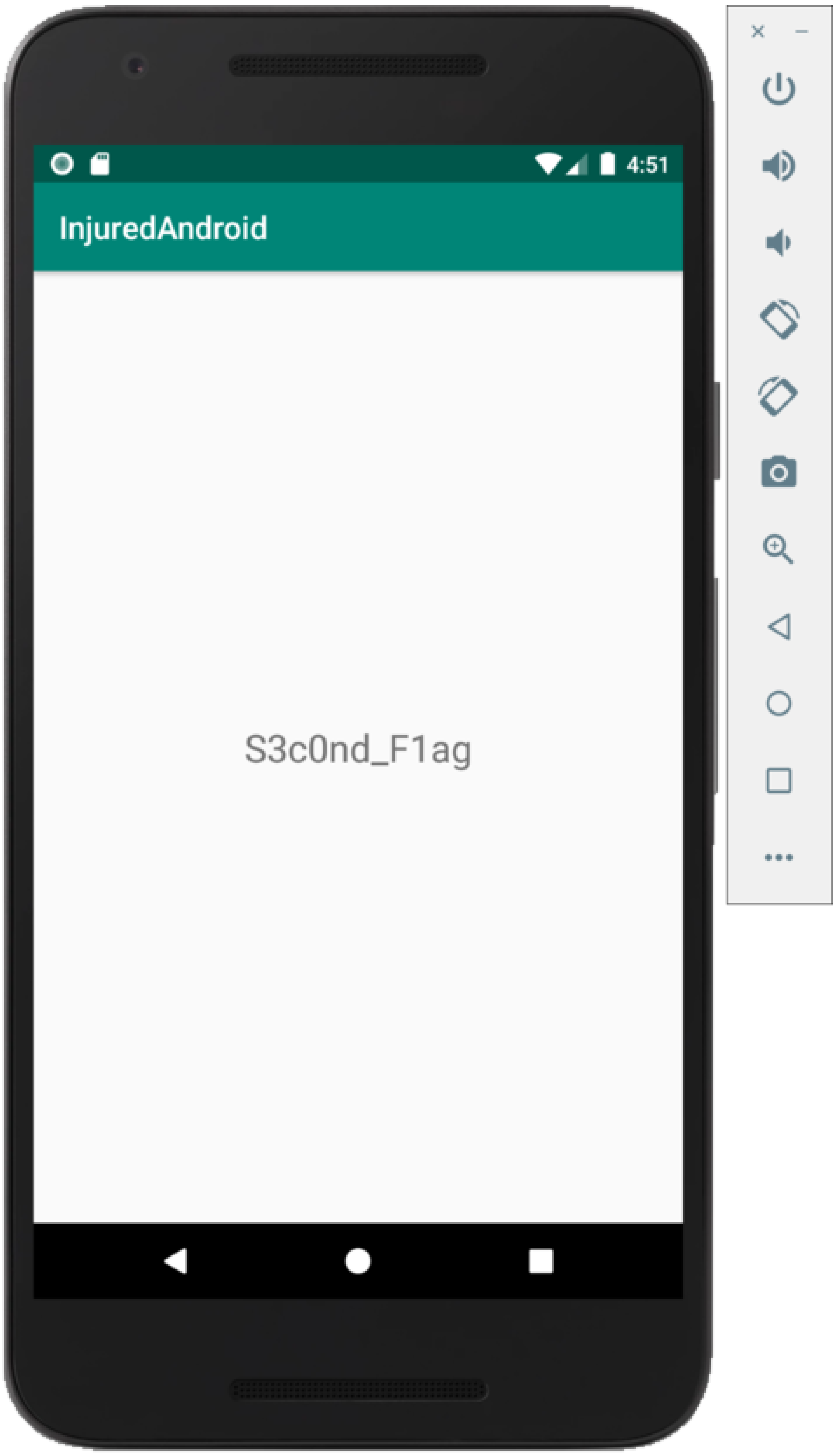
This is a limited list, look in AndroidManifest.xml for exported activities:

```
        <activity android:name="b3nac.injuredandroid.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="b3nac.injuredandroid.XSSTextActivity"/>
        <activity android:name="b3nac.injuredandroid.DisplayPostXSS"/>
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="FlagOneLoginActivity"
android:name="b3nac.injuredandroid.FlagOneLoginActivity"/>
        <activity android:name="b3nac.injuredandroid.FlagOneSuccess"/>
        __<activity android:name="b3nac.injuredandroid.b25lActivity"
android:exported="true"/>__
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="@string/title_activity_flag_two"
android:name="b3nac.injuredandroid.FlagTwoActivity"/>
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="@string/title_activity_flag_three"
android:name="b3nac.injuredandroid.FlagThreeActivity"/>
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="@string/title_activity_flag_four"
android:name="b3nac.injuredandroid.FlagFourActivity"/>
        <receiver android:name="b3nac.injuredandroid.FlagFiveReceiver"
android:exported="true"/>
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="@string/title_activity_flag_five"
android:name="b3nac.injuredandroid.FlagFiveActivity"/>
        <activity android:theme="@style/AppTheme.NoActionBar"
android:label="@string/title_activity_test_broadcast_reciever"
android:name="b3nac.injuredandroid.TestBroadcastReceiver"
android:exported="true"/>
        <activity android:name="b3nac.injuredandroid.ContactActivity"/>
```

Most likely candidate is `b25lActivity`, start it via adb:

```
adb shell am start -n b3nac.injuredandroid/.b25lActivity
```

Will trigger the FlagTwo Activity screen:



S3c0nd_F1ag

Alternative

With Drozer:

In the console enter:

```
run app.activity.info -a b3nac.injuredandroid
```

```
Package: b3nac.injuredandroid
  b3nac.injuredandroid.MainActivity
    Permission: null
  b3nac.injuredandroid.b25lActivity
    Permission: null
  b3nac.injuredandroid.TestBroadcastReceiver
    Permission: null
```

lists activities of an App.

Run an activity:

```
run app.activity.start --component b3nac.injuredandroid
b3nac.injuredandroid.b25lActivity
```

this will pop the screen for the second flag

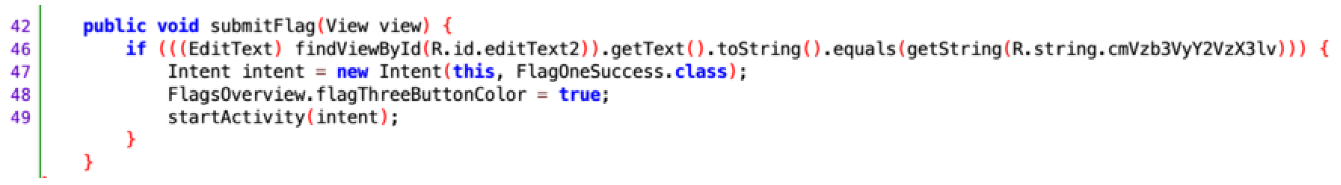
Flag: s3c0nd_F1ag

Done

Challenge 3:

In this case, the flag is compared to a string stored as a resource.

Look at the compare function in FlagThreeActivity:



```
    public void submitFlag(View view) {
        if (((EditText) findViewById(R.id.editText2)).getText().toString().equals(getString(R.string.cmVzb3VyY2VzX3lv))) {
            Intent intent = new Intent(this, FlagOneSuccess.class);
            FlagsOverview.flagThreeButtonColor = true;
            startActivity(intent);
        }
    }
```

Look for resource/string with resource name cmVzb3VyY2VzX3lv

Easiest is to grep through the resource (res) dir in the (APKtool unpacked) APK directory:

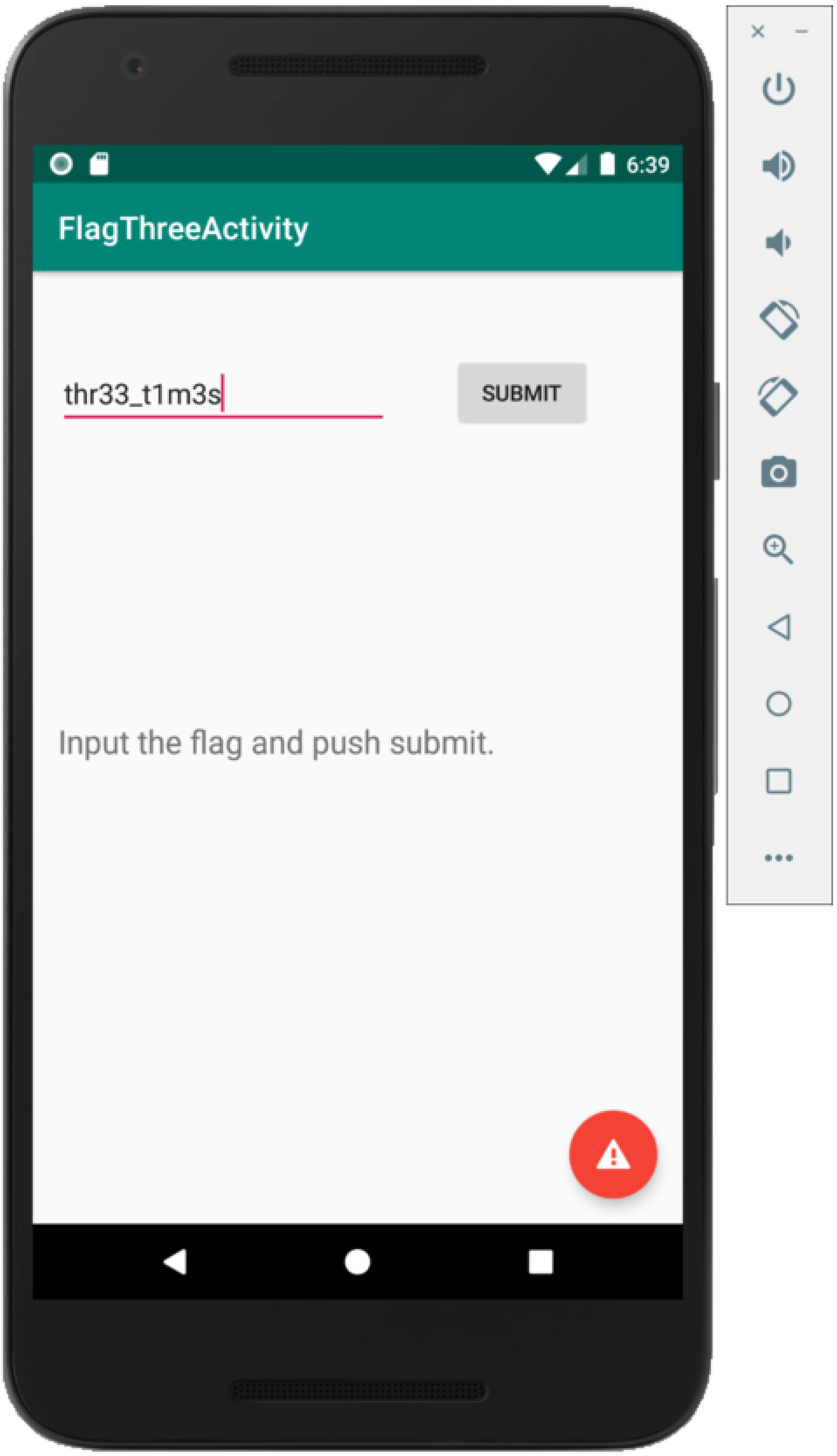
```
grep -iR cmVzb3VyY2VzX3lv .
```

```
./values/public.xml:    <public type="string" name="cmVzb3VyY2VzX3lv"
id="0x7f0d002f" />
./values/strings.xml:    <string
name="cmVzb3VyY2VzX3lv">thr33_t1m3s</string>
```

Or look in resources.arsc / res / strings.xml

```
<string name="cmVzb3VyY2VzX3lv">thr33_t1m3s</string>
```

Enter the flag:

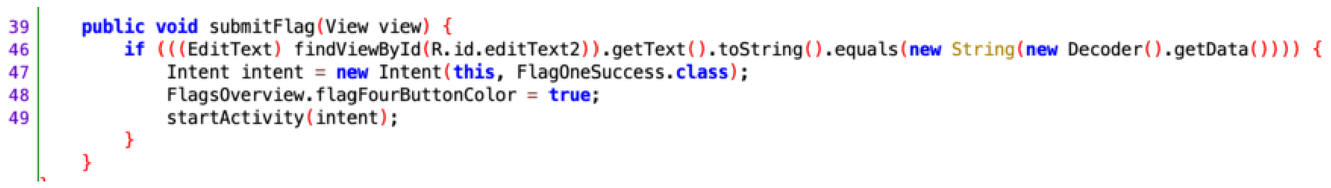


Flag: thr33_t1m3s

Done

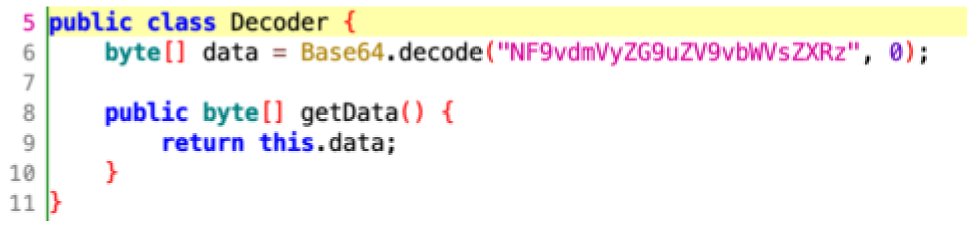
Challenge 4:

Looking at FlagFourActivity, flag is compared to data returned by Encoder.getData:



```java
    public void submitFlag(View view) {
        if (((EditText) findViewById(R.id.editText2)).getText().toString().equals(new String(new Decoder().getData()))) {
            Intent intent = new Intent(this, FlagOneSuccess.class);
            FlagsOverview.flagFourButtonColor = true;
            startActivity(intent);
        }
    }
```

The decoder returns a decoded Base64 encoded string:



```java
public class Decoder {
    byte[] data = Base64.decode("NF9vdmVyZG9uZV9vbWVsZXRz", 0);

    public byte[] getData() {
        return this.data;
    }
}
```

Data is base64 encoded(not encrypted!!!) in Decoder class. getData method returns the base64 decoded data. base64 data can be decoded on the command line:

```
echo 'NF9vdmVyZG9uZV9vbWVsZXRz' | base64 -d
```

```
4_overdone_omelets
```

Which is the flag.

Alternative

Sometimes the encoding is not so easy, then you need to intercept the data at runtime, using Frida. Make sure frida-server is running on device and port has been forwarded with adb.

intercept `getData` method and show the flag:

Define a function that gets a pointer to the class and then overloads the `getData` method. Call this function to activate it.

```
function intercept() {
    // Check if frida has located the JNI
    if (Java.available) {

        // Switch to the Java context
        Java.perform(function() {
            const mydecoder = Java.use('b3nac.injuredandroid.Decoder');
            //We need to overwrite $ instead of $new, since $new =
allocate + init.
            mydecoder.getData.overload().implementation = function () {
                var flag = this.getData()
                // flag is a byte array, need to convert to String, don't
you just love Java?
                var result = "";
                for(var i = 0; i < flag.length; ++i){
                    result += (String.fromCharCode(flag[i] & 0xff)); // here!!
                }
                console.log('[+] decoded data (flag) ' + result);
                return flag;
            }
            console.log('[+] Decoder.getData hooked - check a flag')
        }
    )}
}

intercept()
```

If app is already running, call it with:

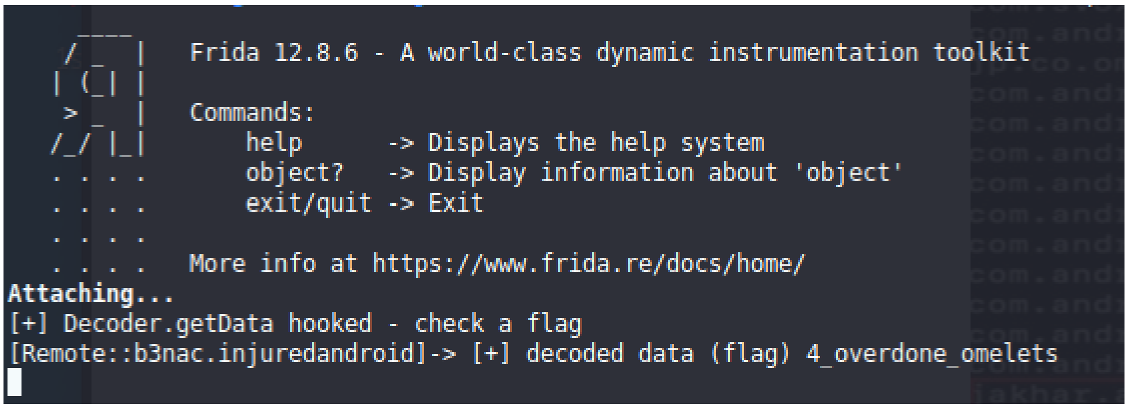
```
frida -H 127.0.0.1 -n $(adb shell pm list packages | grep injured | cut -d: -f2) -l flag4decoder.js
```

If not, then:

```
frida -H 127.0.0.1 -f $(adb shell pm list packages | grep injured | cut -d: -f2) -l flag4decoder.js
```

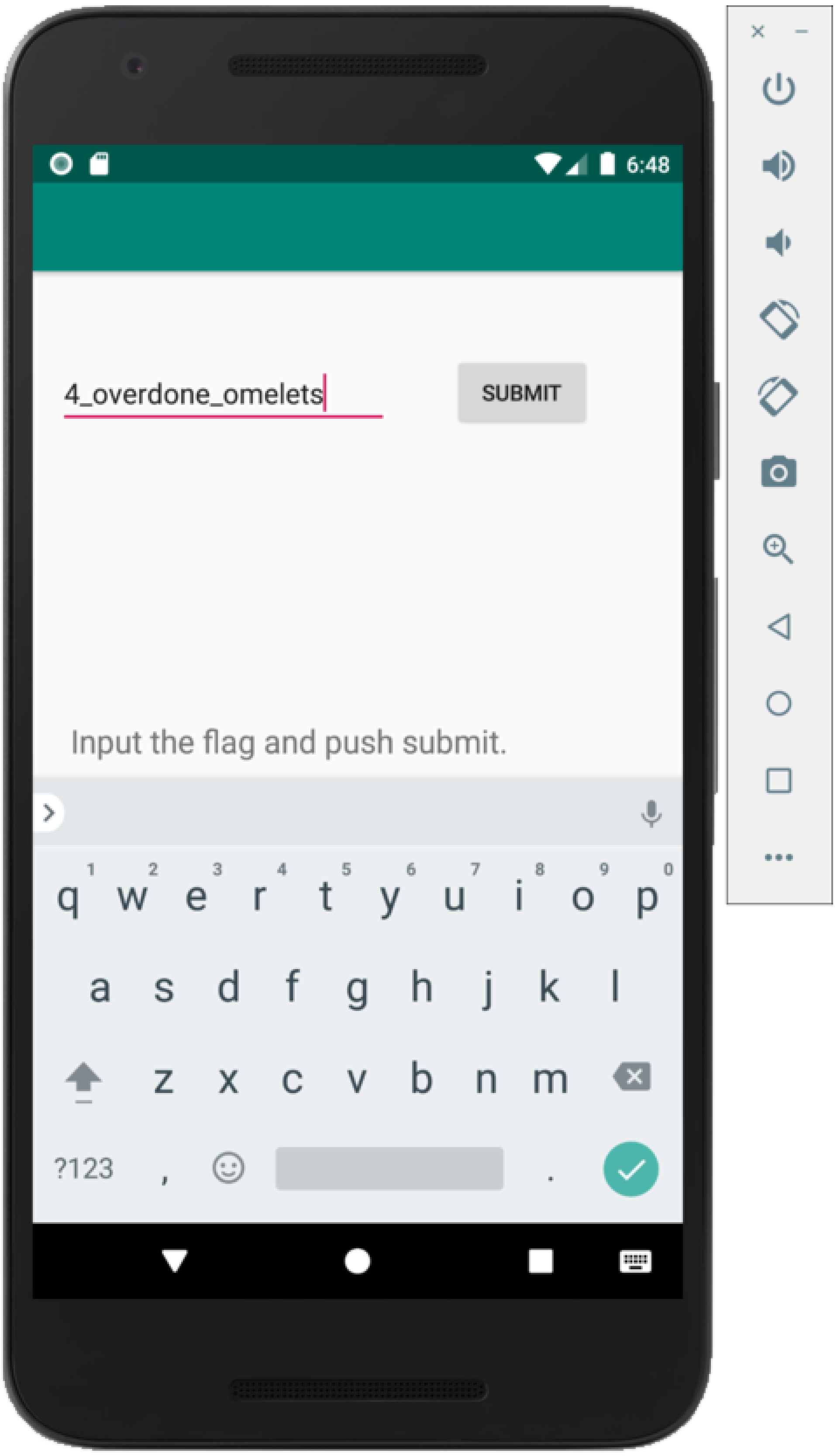
This will start the app.

Now enter a dummy flag to trigger the compare:



```
     ____
    / _  |   Frida 12.8.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Attaching...
[+] Decoder.getData hooked - check a flag
[Remote::b3nac.injuredandroid]-> [+] decoded data (flag) 4_overdone_omelets
```

Which will show the flag.

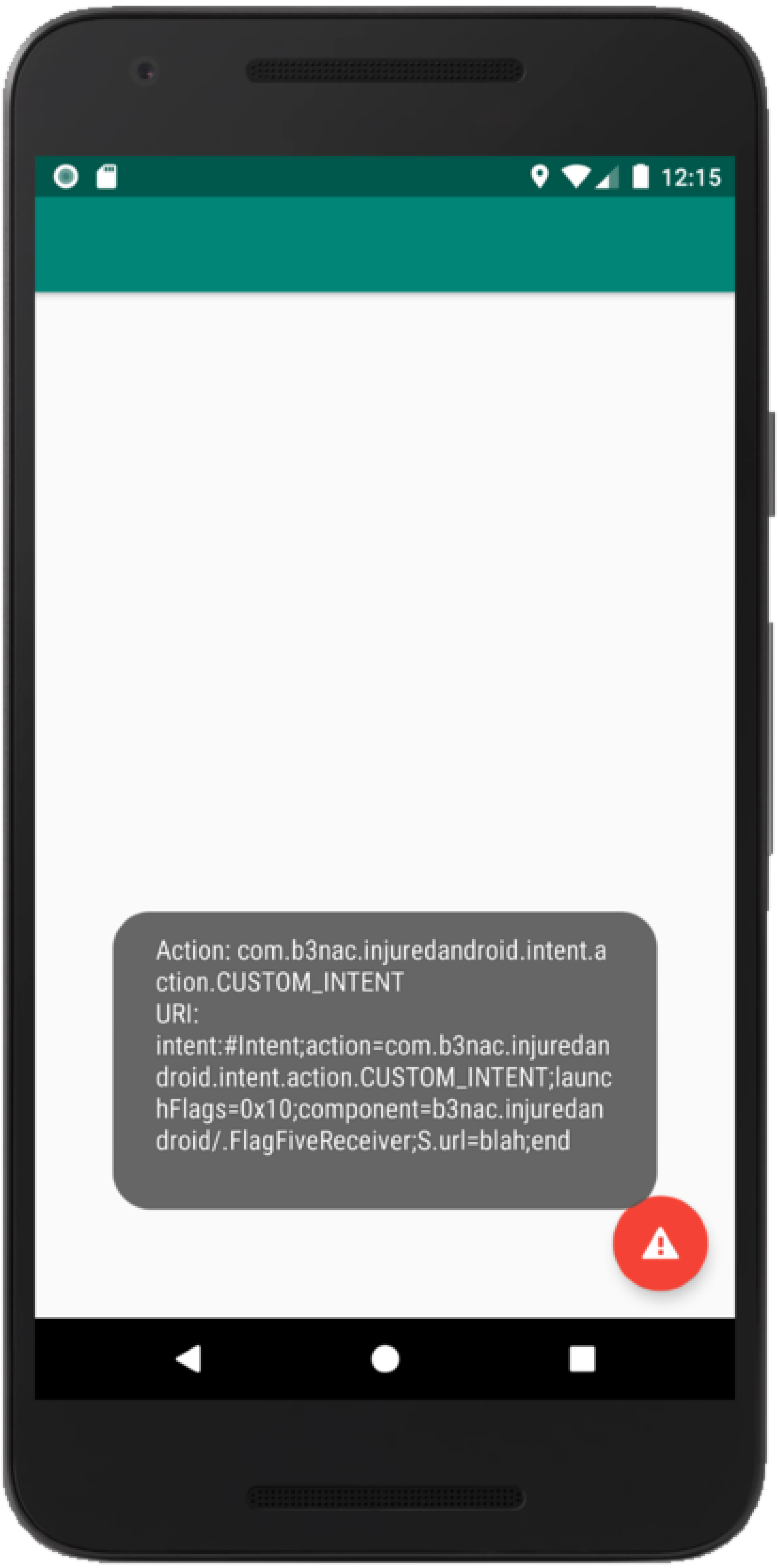


Flag: 4_overdone_omelets

Done

Challenge 5

This is about broadcast receivers.



Action: com.b3nac.injuredandroid.intent.action.CUSTOM_INTENT
URI:
intent:#Intent;action=com.b3nac.injuredandroid.intent.action.CUSTOM_INTENT;launchFlags=0x10;component=b3nac.injuredandroid/.FlagFiveReceiver;S.url=blah;end

Use drozer to list the broadcast receivers for the app:

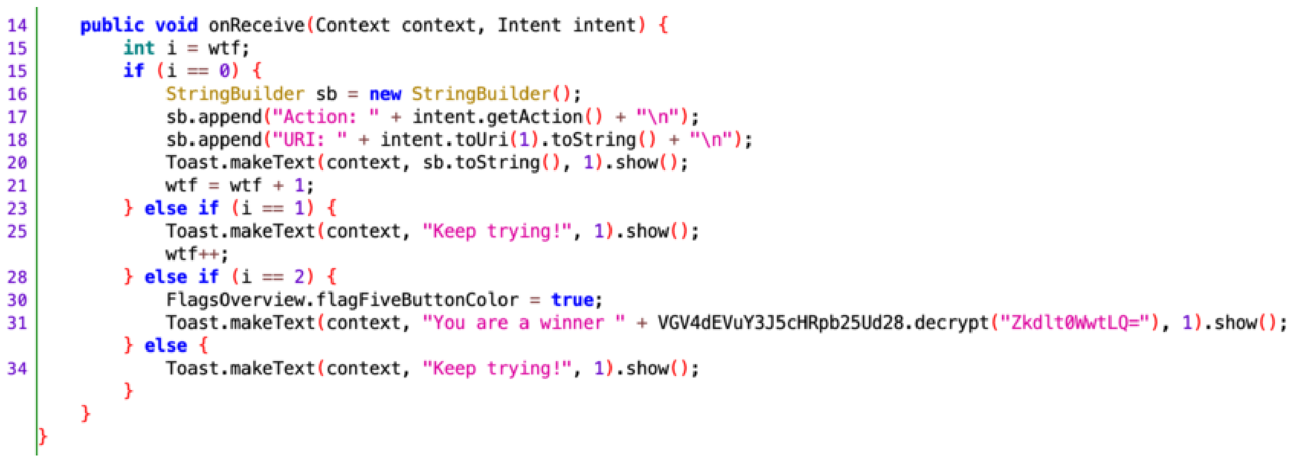
```
run app.broadcast.info -a b3nac.injuredandroid -i
```

```
Package: b3nac.injuredandroid
  b3nac.injuredandroid.FlagFiveReceiver
    Permission: null
```

Permission: null, anyone can send to the receiver. It is a bit of a strange receiver, no real parameters

Due to some bug in Drozer, intents are not listed, Drozer throws a java_path error. Replace the kali apt version with the separate .deb file.

Use JADX to look at the code of the receiver. It has no parameters, you need to trigger it a few times.



```java
    public void onReceive(Context context, Intent intent) {
        int i = wtf;
        if (i == 0) {
            StringBuilder sb = new StringBuilder();
            sb.append("Action: " + intent.getAction() + "\n");
            sb.append("URI: " + intent.toUri(1).toString() + "\n");
            Toast.makeText(context, sb.toString(), 1).show();
            wtf = wtf + 1;
        } else if (i == 1) {
            Toast.makeText(context, "Keep trying!", 1).show();
            wtf++;
        } else if (i == 2) {
            FlagsOverview.flagFiveButtonColor = true;
            Toast.makeText(context, "You are a winner " + VGV4dEVuY3J5cHRpb25Ud28.decrypt("Zkdlt0WwtLQ="), 1).show();
        } else {
            Toast.makeText(context, "Keep trying!", 1).show();
        }
    }
}
```

Using drozer did not work, not sure why. Using adb works:

```
adb shell am broadcast -n b3nac.injuredandroid/.FlagFiveReceiver
```

(note: no action specified, just triggered the receiver) (ref: https://riptutorial.com/android/example/10642/sending-broadcast)

For debugging I used a simple Frida script to trace the activation of the receiver:

```javascript
function intercept() {
    // Check if frida has located the JNI
    if (Java.available) {
        // Switch to the Java context
        Java.perform(function() {
            const myreceiver =
Java.use('b3nac.injuredandroid.FlagFiveReceiver');
            myreceiver.onReceive.overload('android.content.Context',
'android.content.Intent').implementation = function (context, intent) {
                console.log('[+] received a broadcast');
                this.onReceive( context, intent );
            }
            console.log('[+] FlagFiveReceiver.onReceive hooked')
        }
    )}
}
```

```
intercept()
```

More elaborate tracing:

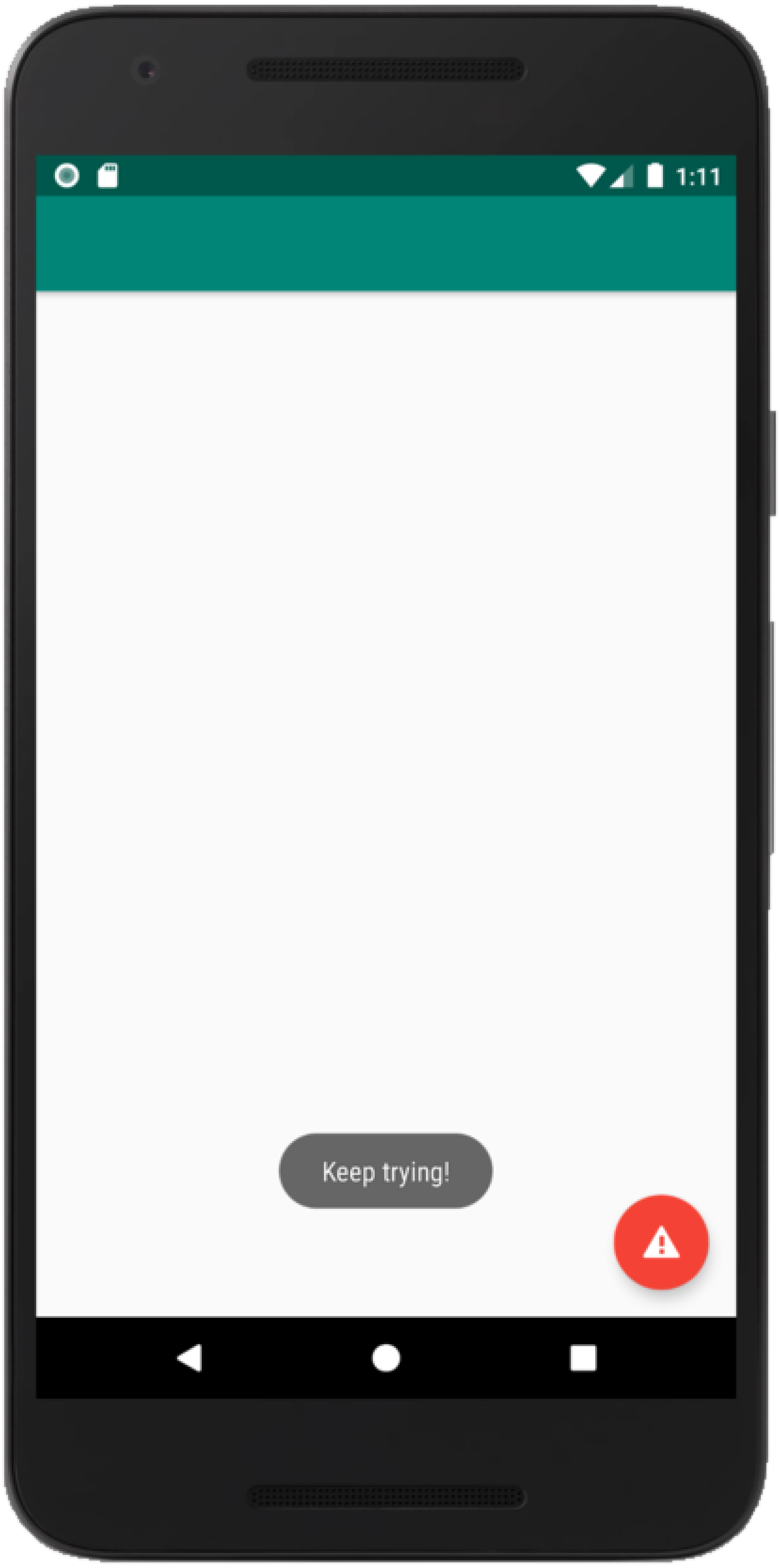
```
function intercept() {
    // Check if frida has located the JNI
    if (Java.available) {
        // Switch to the Java context
        Java.perform(function() {
            const myreceiver =
Java.use('b3nac.injuredandroid.FlagFiveReceiver');
            var Activity = Java.use("android.app.Activity");
            var Intent = Java.use("android.content.Intent");
            myreceiver.onReceive.overload('android.content.Context',
'android.content.Intent').implementation = function (context, intent) {
                console.log('[+] received a broadcast');
                var myintent    = Java.cast(intent, Intent);
                console.log('[+] intent is ' + myintent.toUri(0));
                var myaction    = myintent.getAction();
                var mycomponent = myintent.getComponent();
                var myextras    = myintent.getExtras();
                console.log('[+] action is    ' + myaction.toString());
                console.log('[+] component is ' + mycomponent.toString());
                if( myextras ) {
                    console.log('[+] extras is ' + myextras.toString());
                }
                this.onReceive( context, intent );
            }
            console.log('[+] FlagFiveReceiver.onReceive hooked')
        }
    )}
}

intercept()
```

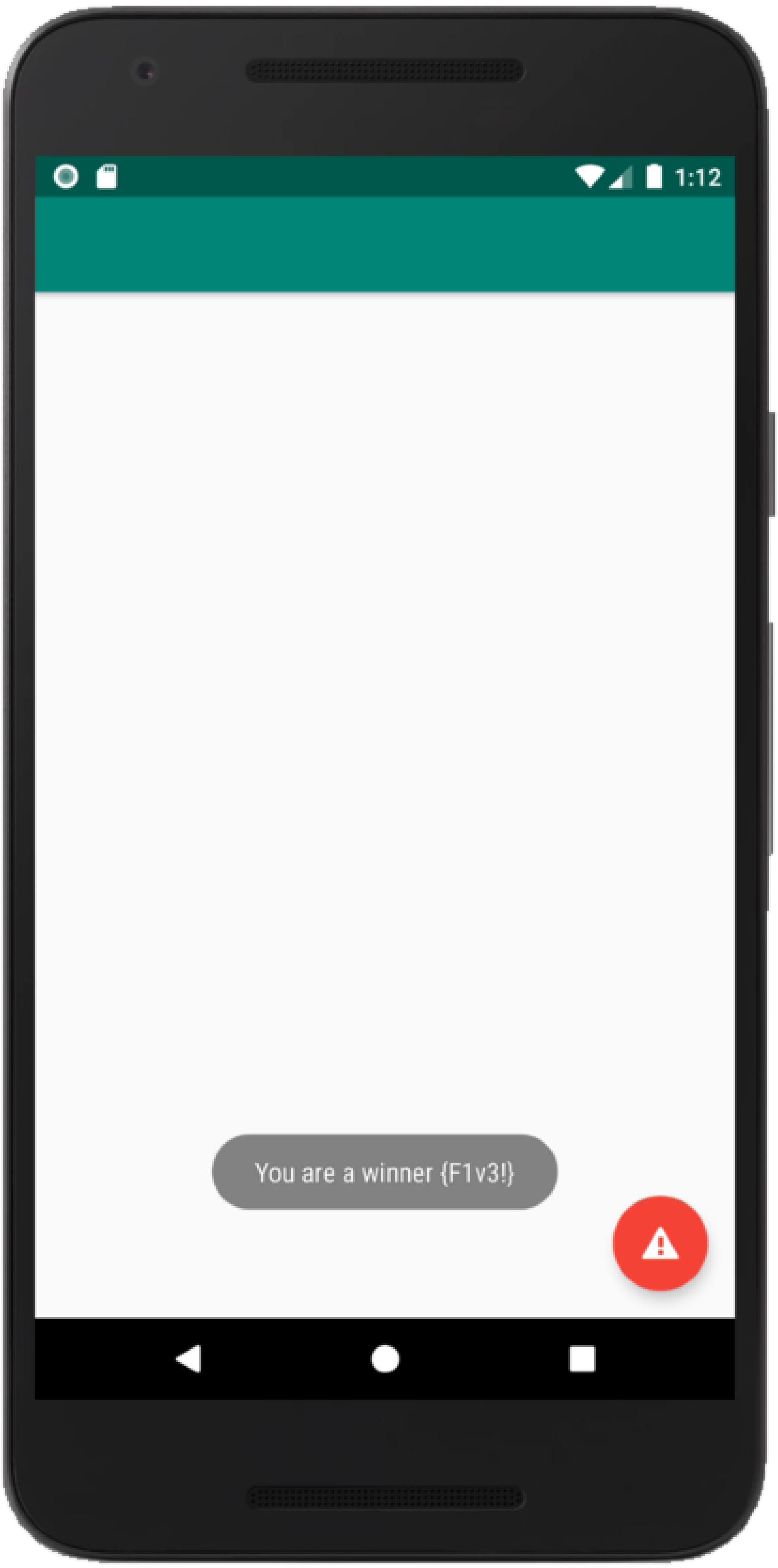
Gives the following output when a message is received:

```
[+] FlagFiveReceiver.onReceive hooked
[Remote::b3nac.injuredandroid]-> [+] received a broadcast
[+] intent is
#Intent;action=Pwned;launchFlags=0x400010;package=bar;component=b3nac.inju
redandroid/.FlagFiveReceiver;S.s=foo;end
[+] action is    Pwned
[+] component is
ComponentInfo{b3nac.injuredandroid/b3nac.injuredandroid.FlagFiveReceiver}
[+] extras is Bundle[{s=foo}]
```

when triggered.

Triggering the receiver two times yields the flag:



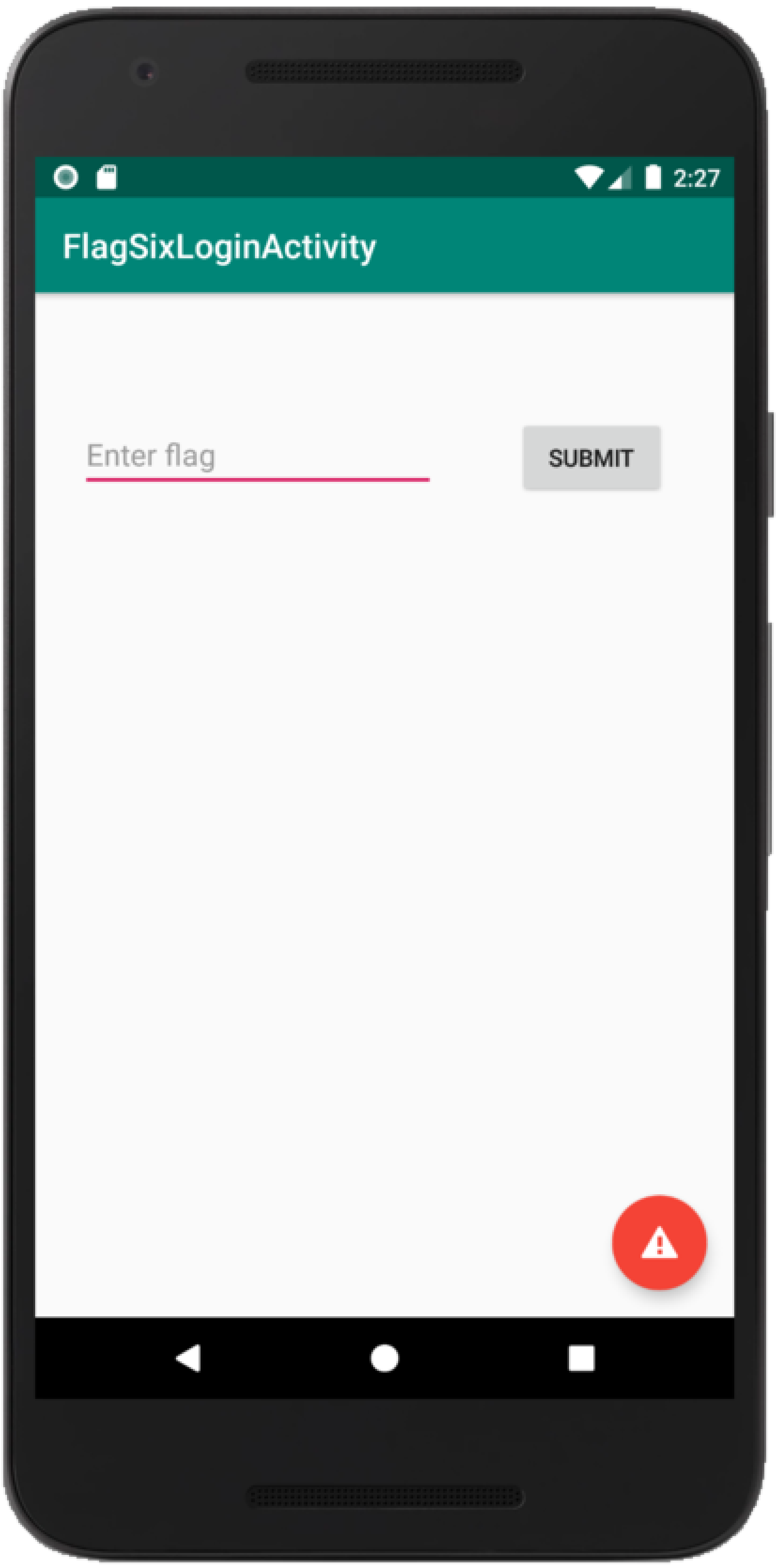


You are a winner {F1v3!}

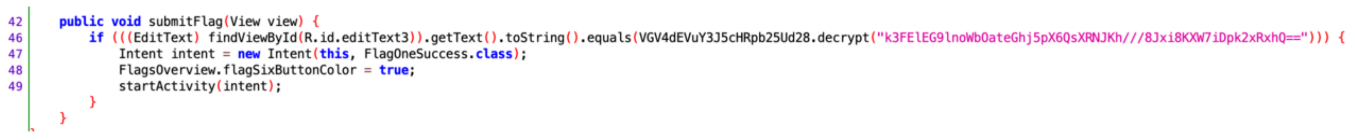
Flag {F1v3!}

Done

Challenge 6



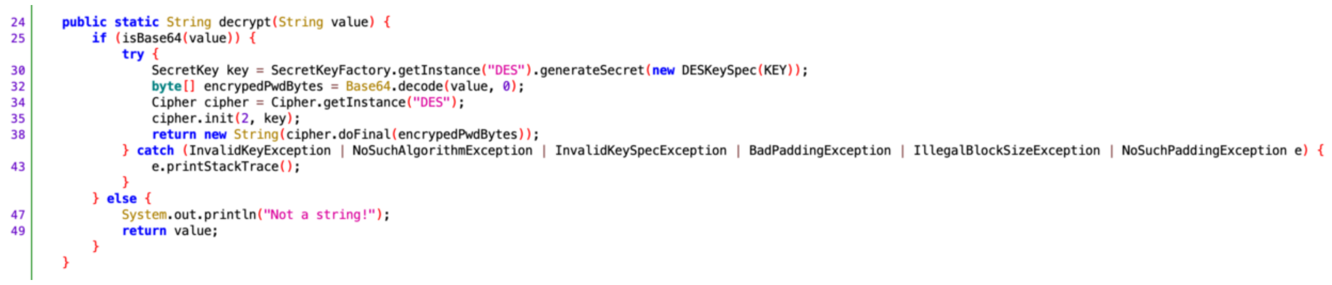
This challenge uses 'real' encryption (DES):



```java
    public void submitFlag(View view) {
        if (((EditText) findViewById(R.id.editText3)).getText().toString().equals(VGV4dEVuY3J5cHRpb25Ud28.decrypt("k3FElEG9lnoWbOateGhj5pX6QsXRNJKh///8Jxi8KXW7iDpk2xRxhQ=="))) {
            Intent intent = new Intent(this, FlagOneSuccess.class);
            FlagsOverview.flagSixButtonColor = true;
            startActivity(intent);
        }
    }
```

The entered flag iis compared to the flag that is decrypted via

```
VGV4dEVuY3J5cHRpb25Ud28.decrypt("k3FElEG9lnoWbOateGhj5pX6QsXRNJKh///8Jxi8K
XW7iDpk2xRxhQ==")
```

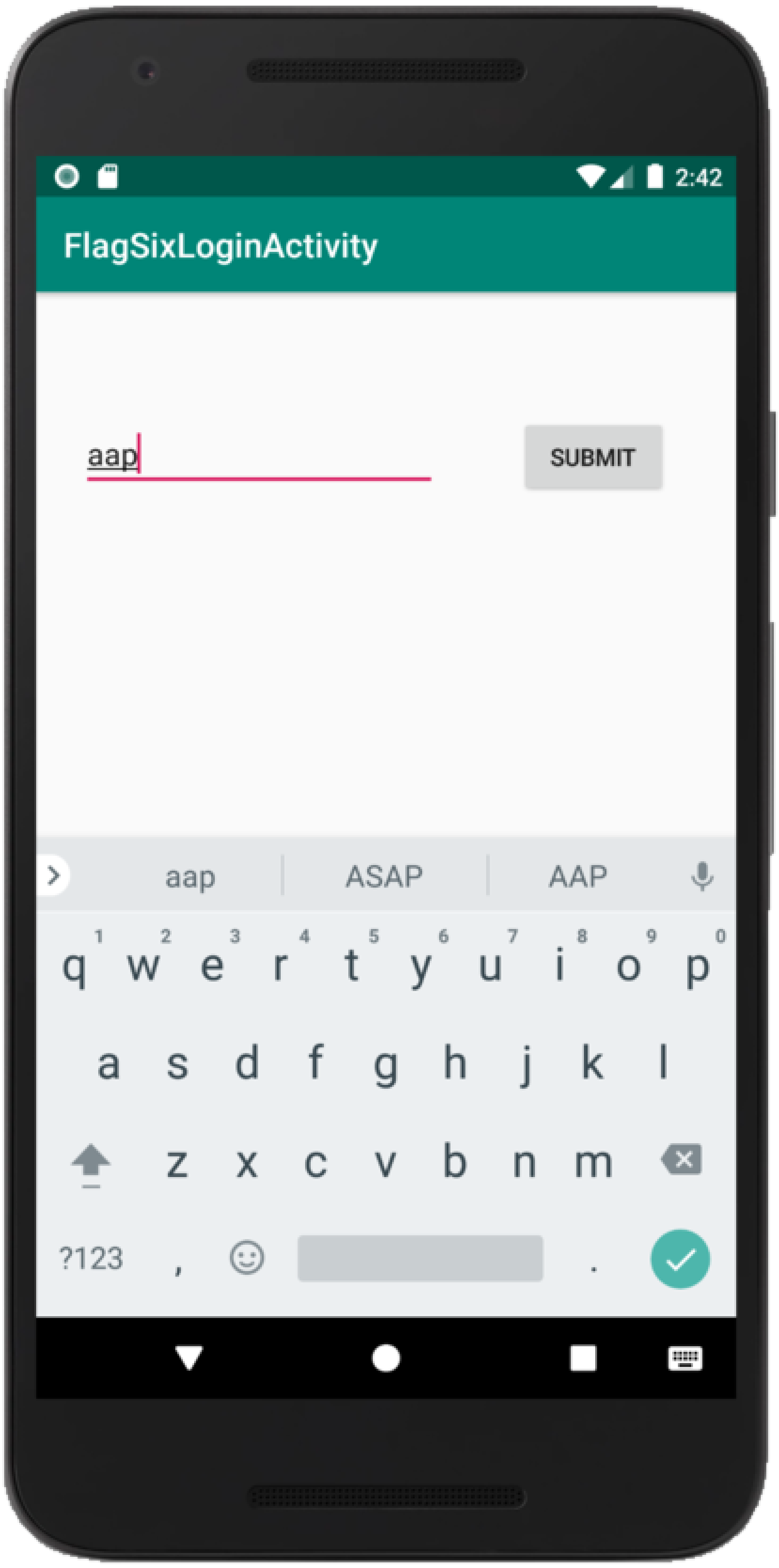


```
public static String decrypt(String value) {
    if (isBase64(value)) {
        try {
            SecretKey key = SecretKeyFactory.getInstance("DES").generateSecret(new DESKeySpec(KEY));
            byte[] encrypedPwdBytes = Base64.decode(value, 0);
            Cipher cipher = Cipher.getInstance("DES");
            cipher.init(2, key);
            return new String(cipher.doFinal(encrypedPwdBytes));
        } catch (InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e) {
            e.printStackTrace();
        }
    } else {
        System.out.println("Not a string!");
        return value;
    }
}
```

This is base64 encoded data that is decrypted with the key via DES. By far the easiest approach is to do the same thing as in Challenge 4, overload the class and catch the decrypted data, using Frida.

```
function intercept() {
    // Check if frida has located the JNI
    if (Java.available) {

        // Switch to the Java context
        Java.perform(function() {
            const mydecrypt =
Java.use('b3nac.injuredandroid.VGV4dEVuY3J5cHRpb25Ud28');
            mydecrypt.decrypt.overload('java.lang.String').implementation
= function (data) {
                console.log('[+] decrypting data ' + data );
                var flag = this.decrypt(data);
                console.log('[+] decrypted data (flag) ' + flag);
                return flag;
            }
            console.log('[+] VGV4dEVuY3J5cHRpb25Ud28.decrypt hooked -
check a flag')
        }
    )}
}

intercept()
```

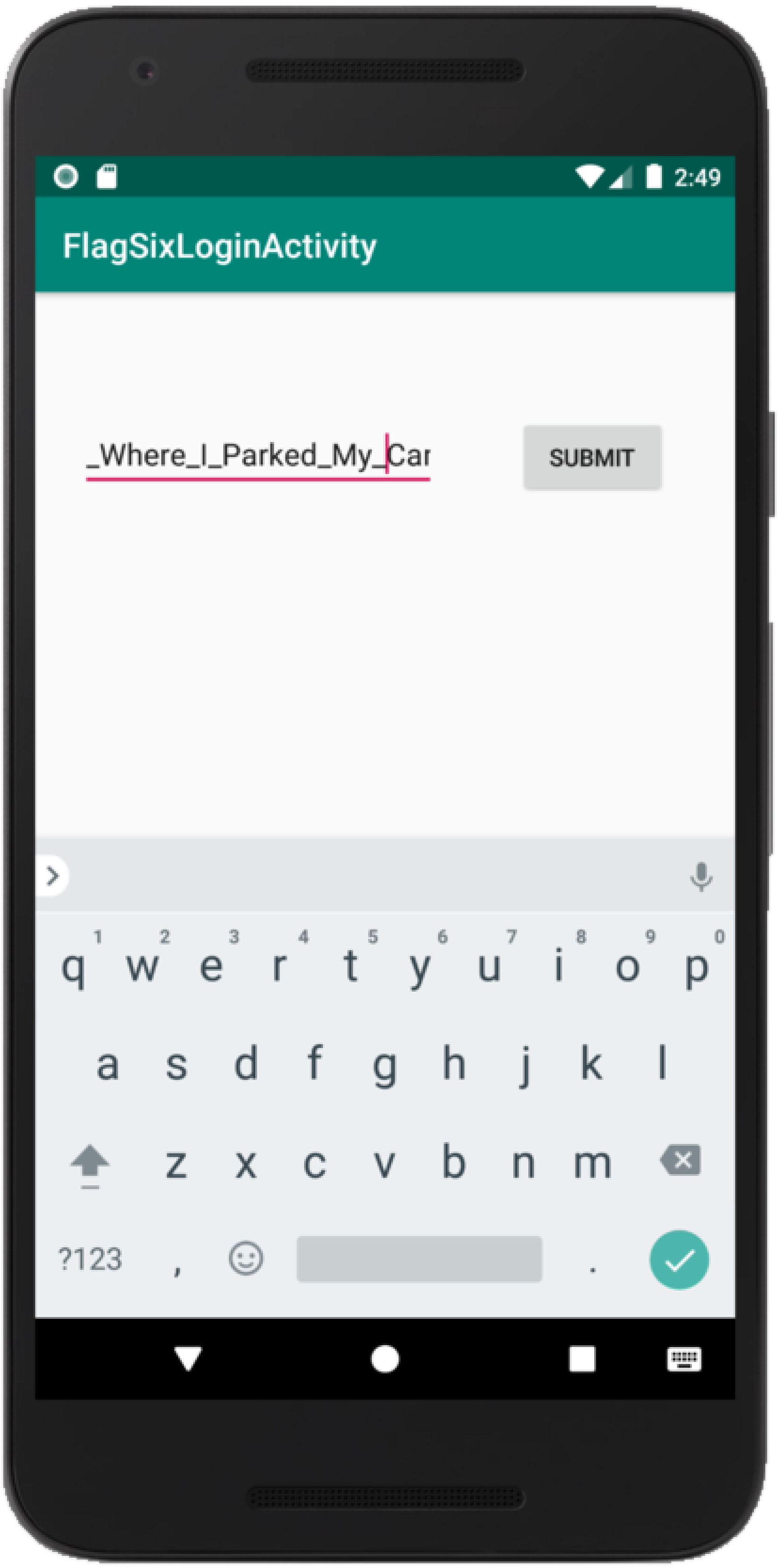
Sending a dummy flag triggers the Frida code and reveals the flag:



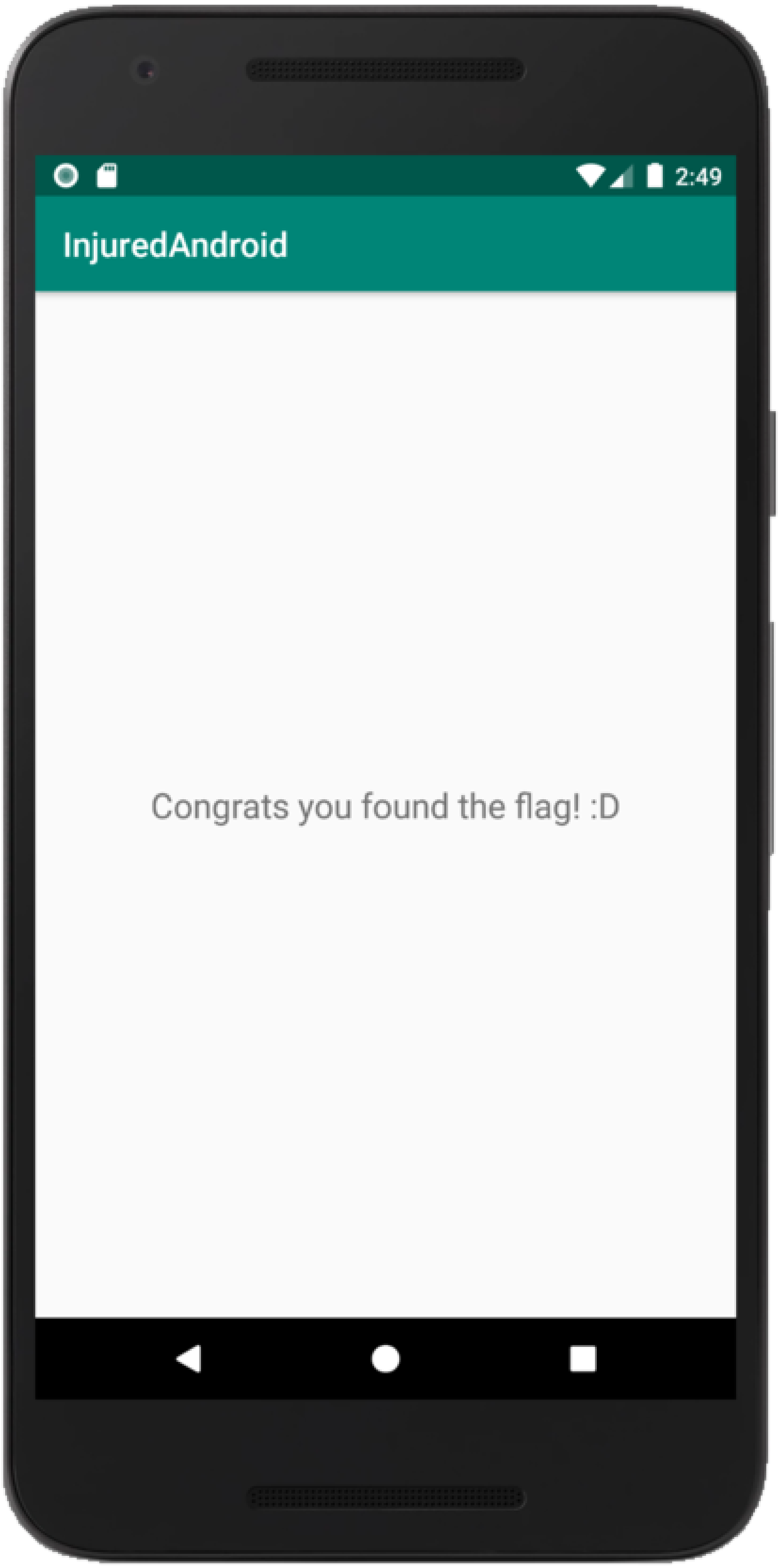
Gives:

```
     ____
    / _  |   Frida 12.8.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Attaching...
[+] VGV4dEVuY3J5cHRpb25Ud28.decrypt hooked - check a flag
[Remote::b3nac.injuredandroid]-> [+] decrypting data
k3FElEG9lnoWbOateGhj5pX6QsXRNJKh///8Jxi8KXW7iDpk2xRxhQ==
[+] decrypted data (flag) {This_Isn't_Where_I_Parked_My_Car}
```

Enter the real flag:



Result:

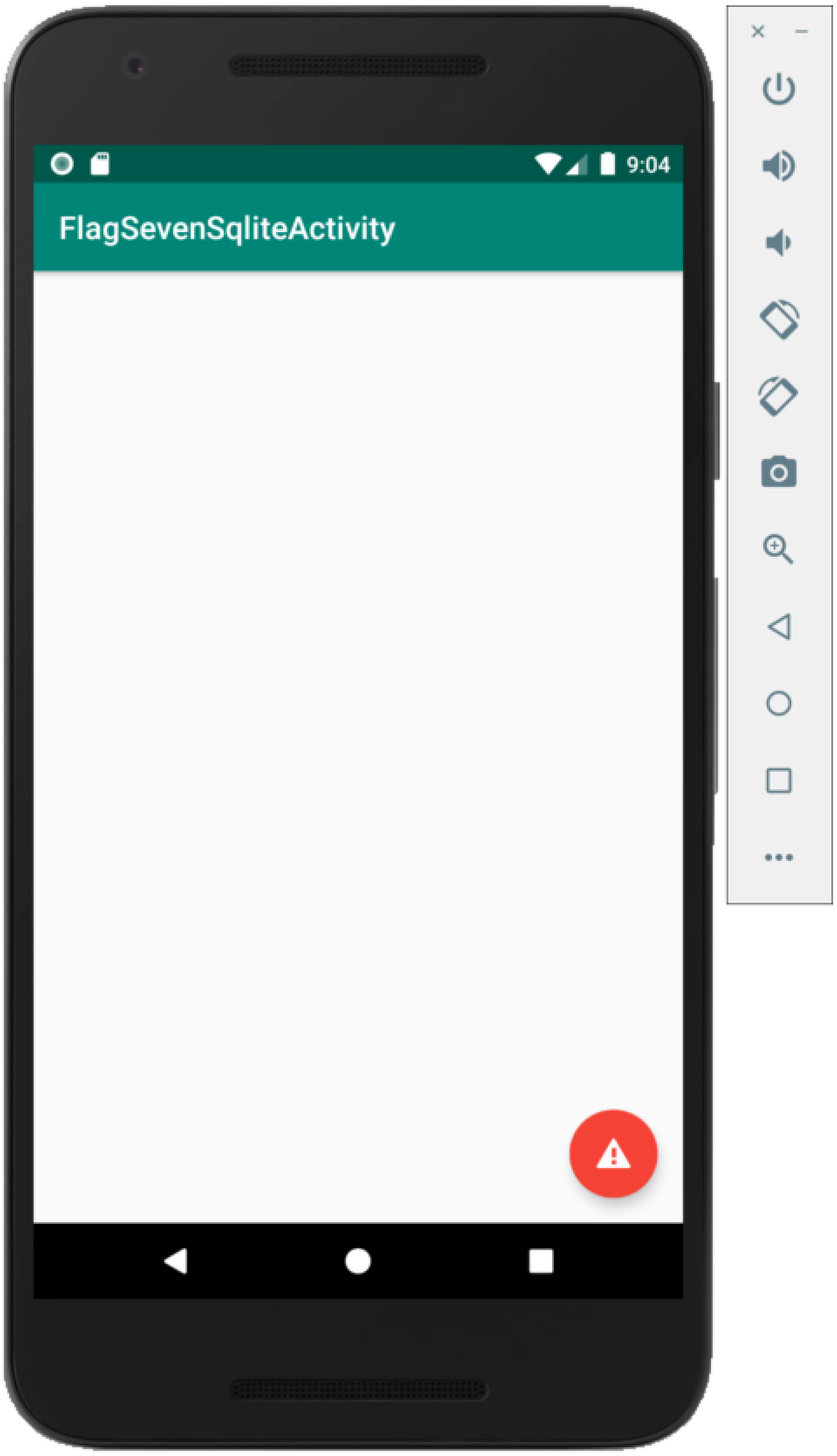


Congrats you found the flag! :D

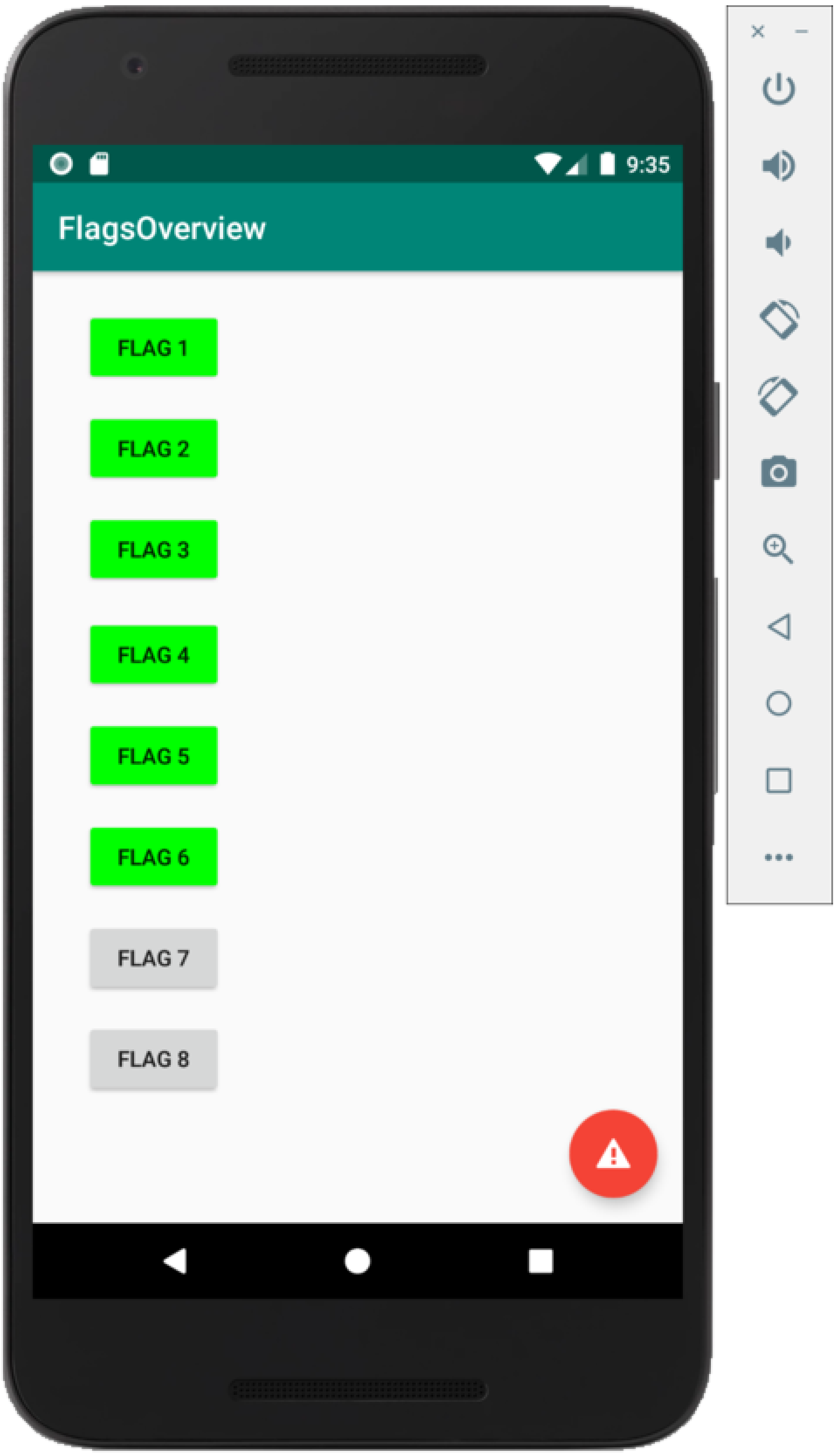
Flag {This_Isn't_Where_I_Parked_My_Car}

Done

Challenge 7

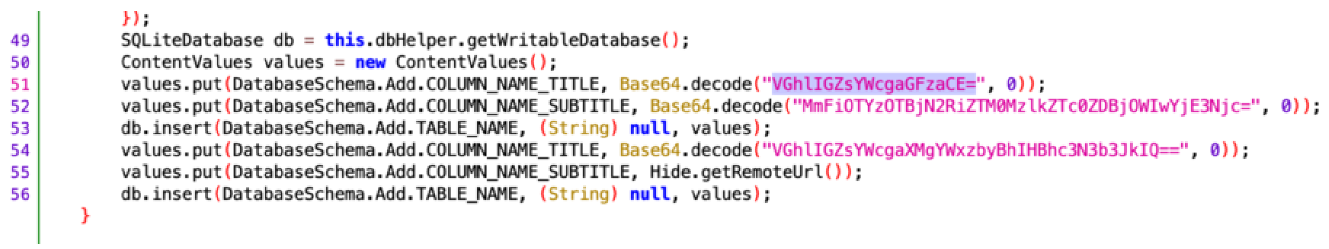


SQLite challenge. It creates a database, stores data in the database. Note that this challenge only starts when you have completed the previous 6.



If you stop the app you have to start over again. (Exercise for student, can you write a frida script that sets all six flag to done to be able to skip that?)

Code contains 3 Base64 encoded strings:



```
        });
        SQLiteDatabase db = this.dbHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(DatabaseSchema.Add.COLUMN_NAME_TITLE, Base64.decode("VGhlIGZsYWcgaGFzaCE=", 0));
        values.put(DatabaseSchema.Add.COLUMN_NAME_SUBTITLE, Base64.decode("MmFiOTYzOTBjN2RiZTM0MzlkZTc0ZDBjOWIwYjE3Njc=", 0));
        db.insert(DatabaseSchema.Add.TABLE_NAME, (String) null, values);
        values.put(DatabaseSchema.Add.COLUMN_NAME_TITLE, Base64.decode("VGhlIGZsYWcgaXMgYWxzbyBhIHBhc3N3b3JkIQ==", 0));
        values.put(DatabaseSchema.Add.COLUMN_NAME_SUBTITLE, Hide.getRemoteUrl());
        db.insert(DatabaseSchema.Add.TABLE_NAME, (String) null, values);
    }
```

```
echo 'VGhlIGZsYWcgaGFzaCE=' | base64 -d    --> The flag hash!
```

```
echo 'MmFiOTYzOTBjN2RiZTM0MzlkZTc0ZDBjOWIwYjE3Njc=' | base64 -d   -->
2ab96390c7dbe3439de74d0c9b0b1767
```

```
echo 'VGhlIGZsYWcgaXMgYWxzbyBhIHBhc3N3b3JkIQ==' | base64 -d   -->  The
flag is also a password!
```

The Hide.getRemoteUrl is also involved:

```
private static byte[] encKey = Base64.decode("Q2FwdHVyM1RoMXM=", 0);
private static byte[] encKeyTwo =
Base64.decode("e0NhcHR1cjNUaDFzVG9vfQ==", 0);
private static String remoteUrl = "uggcf://ceviabgr.pbz/GgDUZ39z";
```

The remoteUrl is not valid, is ROT13 encoded (use Cyberchef to decode)

```
https://privnote.com/TtQHM39m
```

Redirects to a non existing note.

Using adb to connect to the emulator and cd to the data directory of the application:

```
generic_x86:/data/data/b3nac.injuredandroid # ls -l
total 28
drwxrwx--x 2 u0_a85 u0_a85       4096 2020-01-13 15:47 app_textures
drwx------ 4 u0_a85 u0_a85       4096 2020-01-13 15:47 app_webview
drwxrws--x 3 u0_a85 u0_a85_cache 4096 2020-01-13 15:47 cache
drwxrws--x 2 u0_a85 u0_a85_cache 4096 2020-01-13 15:22 code_cache
drwxrwx--x 2 u0_a85 u0_a85       4096 2020-01-14 15:43 databases
drwxrwx--x 2 u0_a85 u0_a85       4096 2020-01-13 15:47 files
drwxrwx--x 2 u0_a85 u0_a85       4096 2020-01-13 15:47 shared_prefs
```

```
generic_x86:/data/data/b3nac.injuredandroid/databases # ls -al
total 28
drwxrwx--x 2 u0_a85 u0_a85  4096 2020-01-14 15:43 .
drwx------ 9 u0_a85 u0_a85  4096 2020-01-14 15:43 ..
-rw-rw---- 1 u0_a85 u0_a85 16384 2020-01-14 15:43 Thisisatest.db
-rw-rw---- 1 u0_a85 u0_a85     0 2020-01-14 15:43 Thisisatest.db-journal
```

Please note, the database only exists while the activity is running.

Copy the database to local system:

```
adb pull /data/data/b3nac.injuredandroid/databases/Thisisatest.db
```

And reading it with sqlite3:

```
sqlite3 Thisisatest.db
SQLite version 3.31.0 2019-12-29 00:52:41
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE Thisisatest (_id INTEGER PRIMARY KEY,title TEXT,subtitle
TEXT);
sqlite> select * from Thisisatest;
1|The flag hash!|2ab96390c7dbe3439de74d0c9b0b1767
2|The flag is also a password!|uggcf://ceviabgr.pbz/GgDUZ39z
sqlite>
```

The hash `2ab96390c7dbe3439de74d0c9b0b1767` is the md5 hash for `hunter2`

Let's consider that the flag. The code does not set the color of flag 7

Flag? hunter2

Done

Challenge 8

This is about AWS credentials stored in the app, but I could not find any.

Done