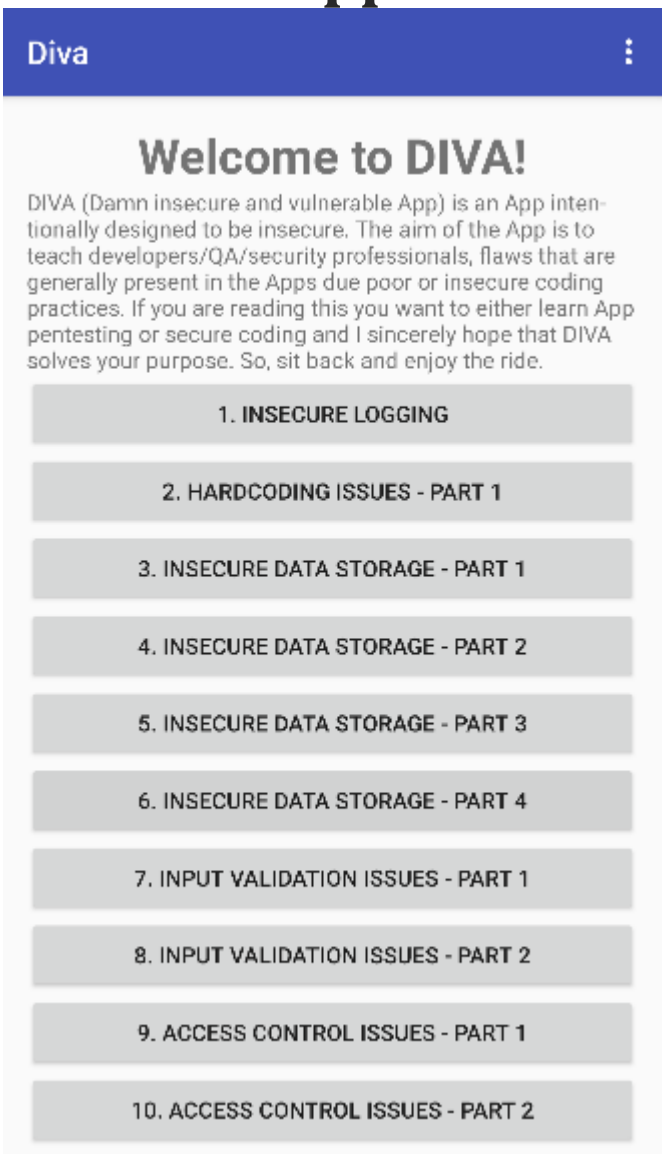
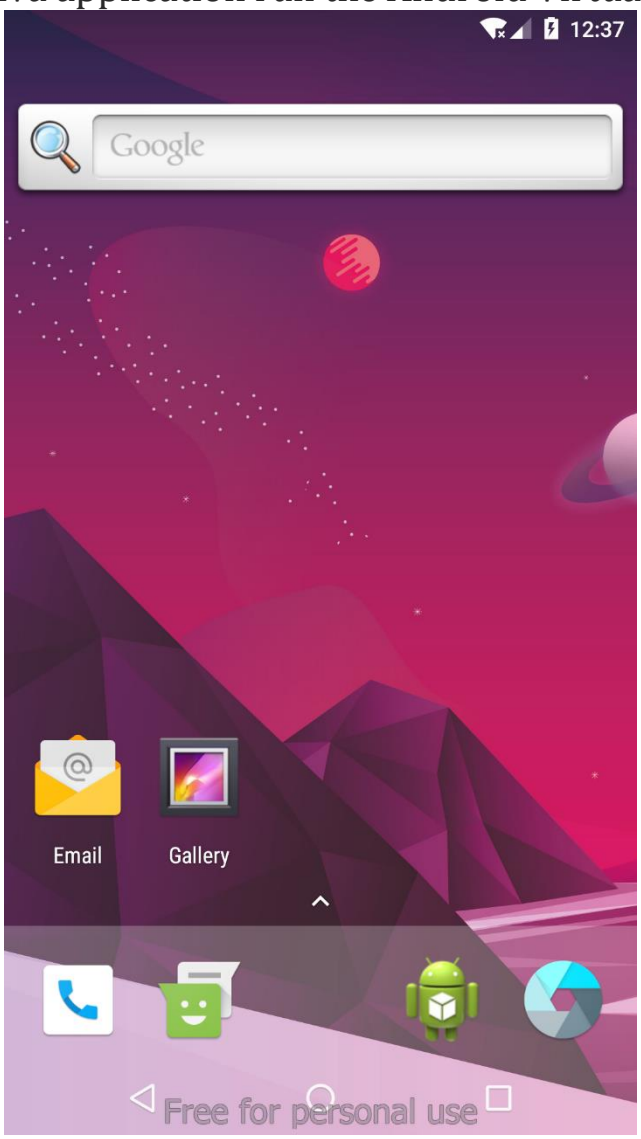
DIVA Android App – Pen-Testing



To download DIVA App : https://payatu.com/wp-content/uploads/2016/01/diva-beta.tar.gz

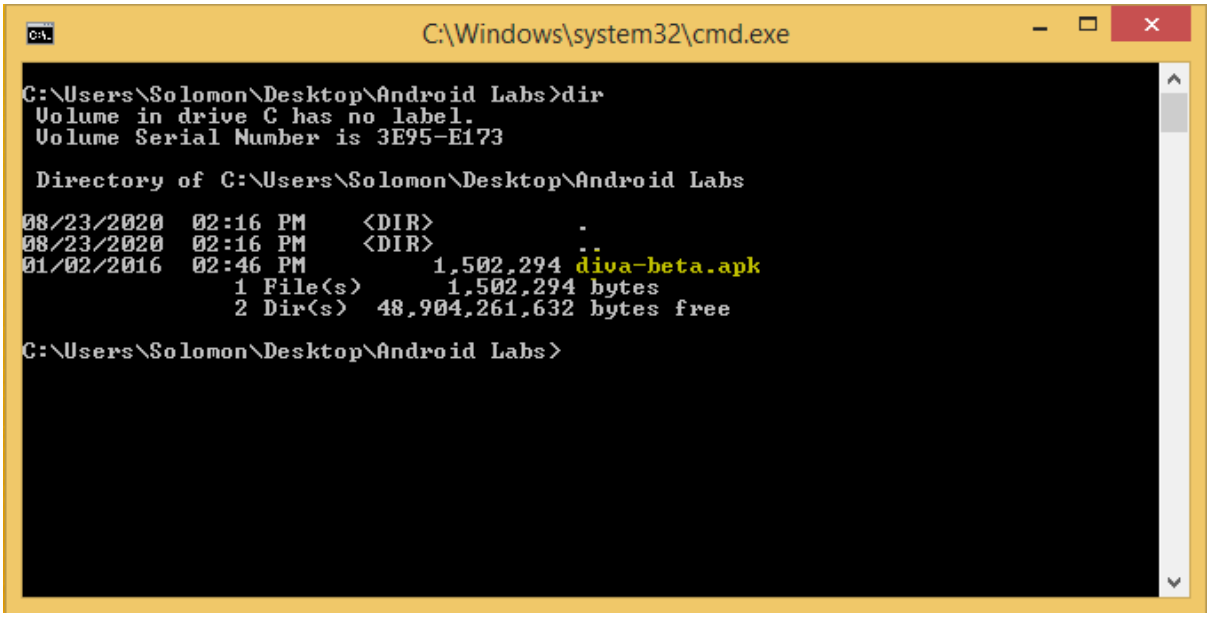
Installation:

In order to install the Diva application run the Android Virtual machine.



Either you can Drag and Drop the APK file of DIVA on Android VM or you can install it with Android Debug Bridge (adb). Installation with ADB will be discussed here.

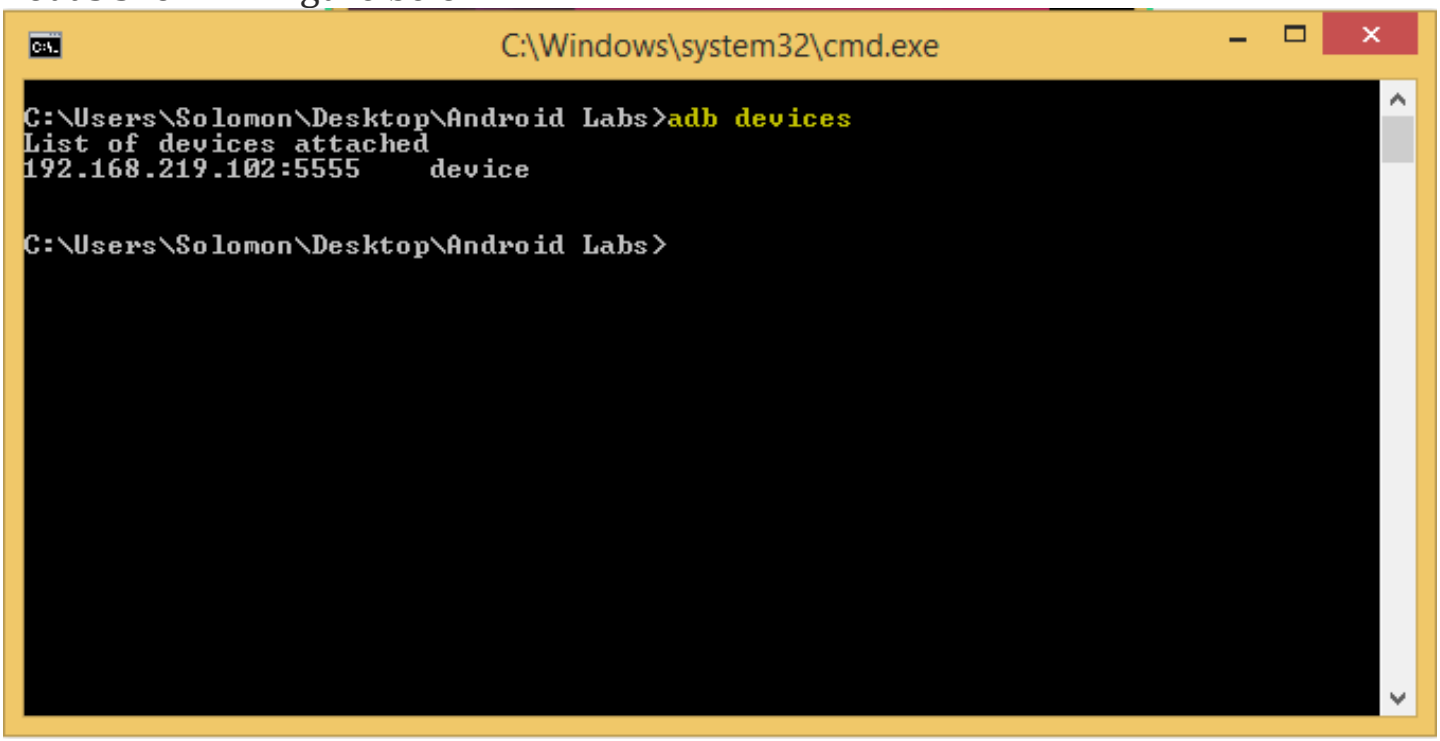
Open Command Prompt and Navigate to the location of DIVA APK file.



Now run following command:

adb devices

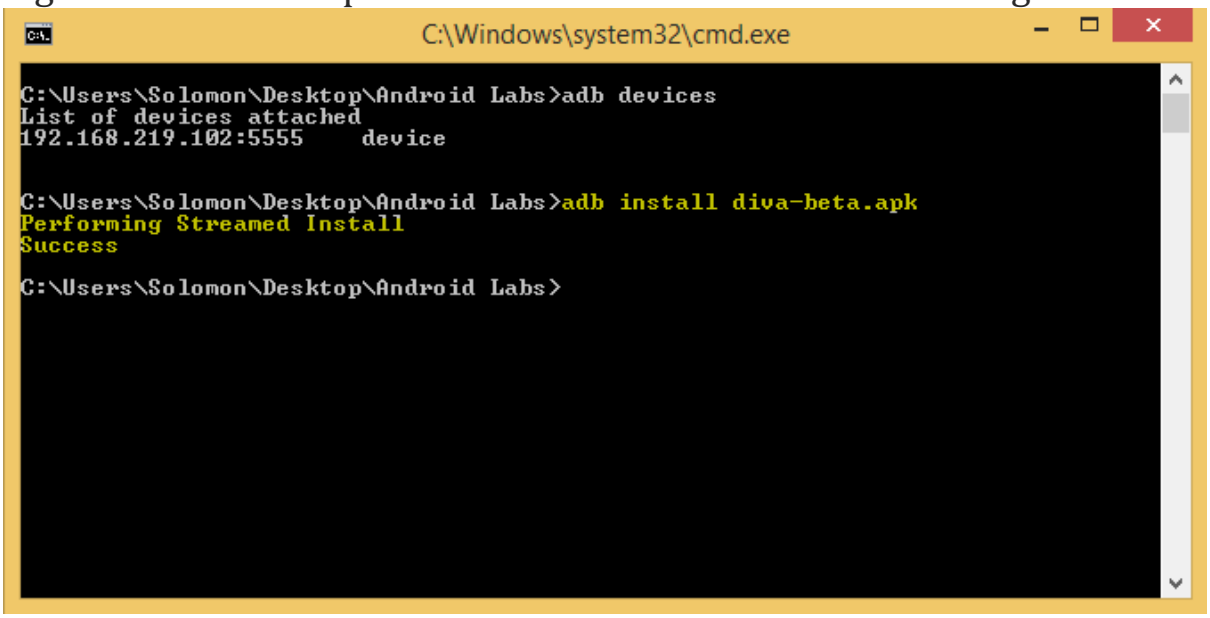
This command will show us status of any android device running on our system or not as shown in figure below



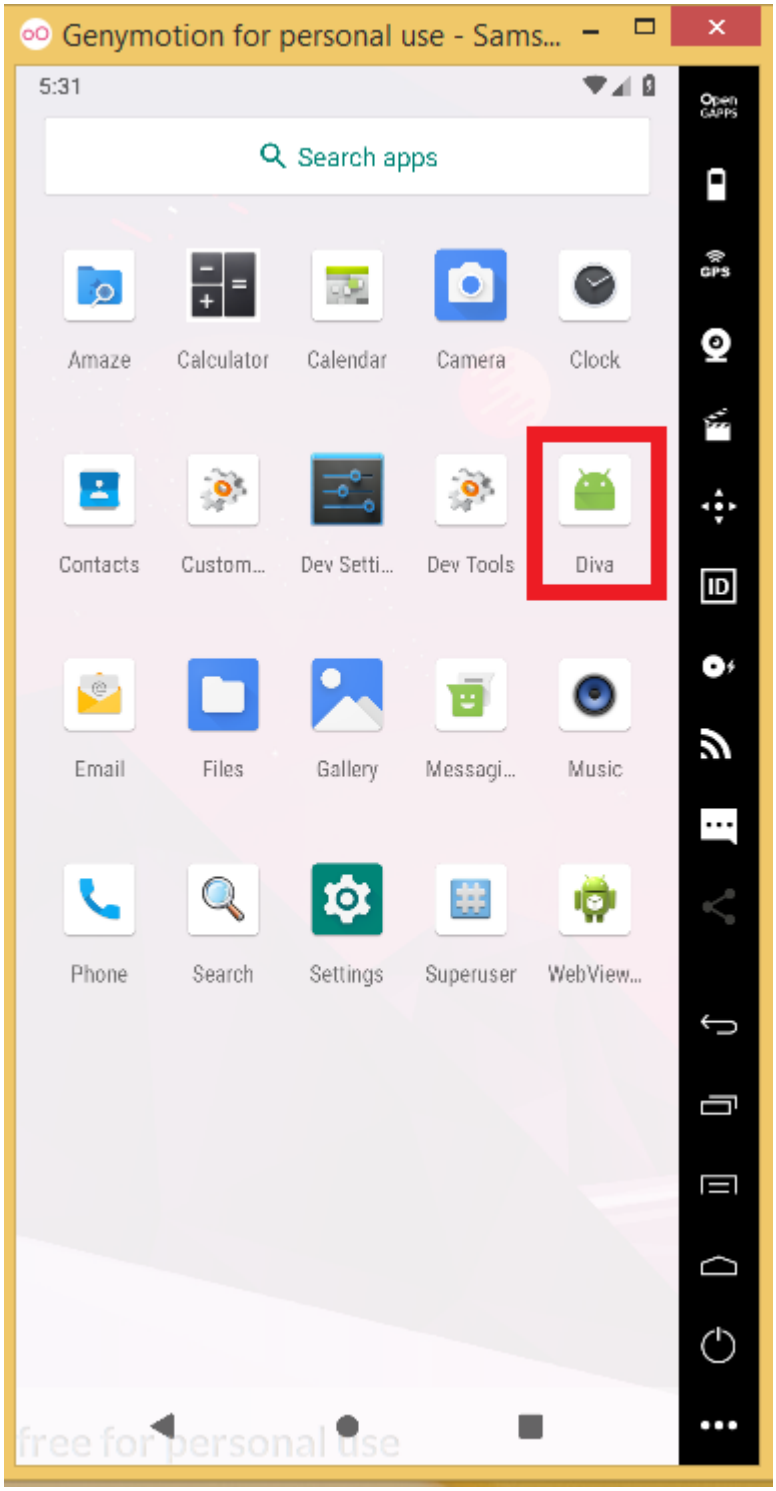
As VM which we started earlier is running, now it's time to install DIVA application. Run command given below and shown in figure 1.4:

adb install diva-beta.apk

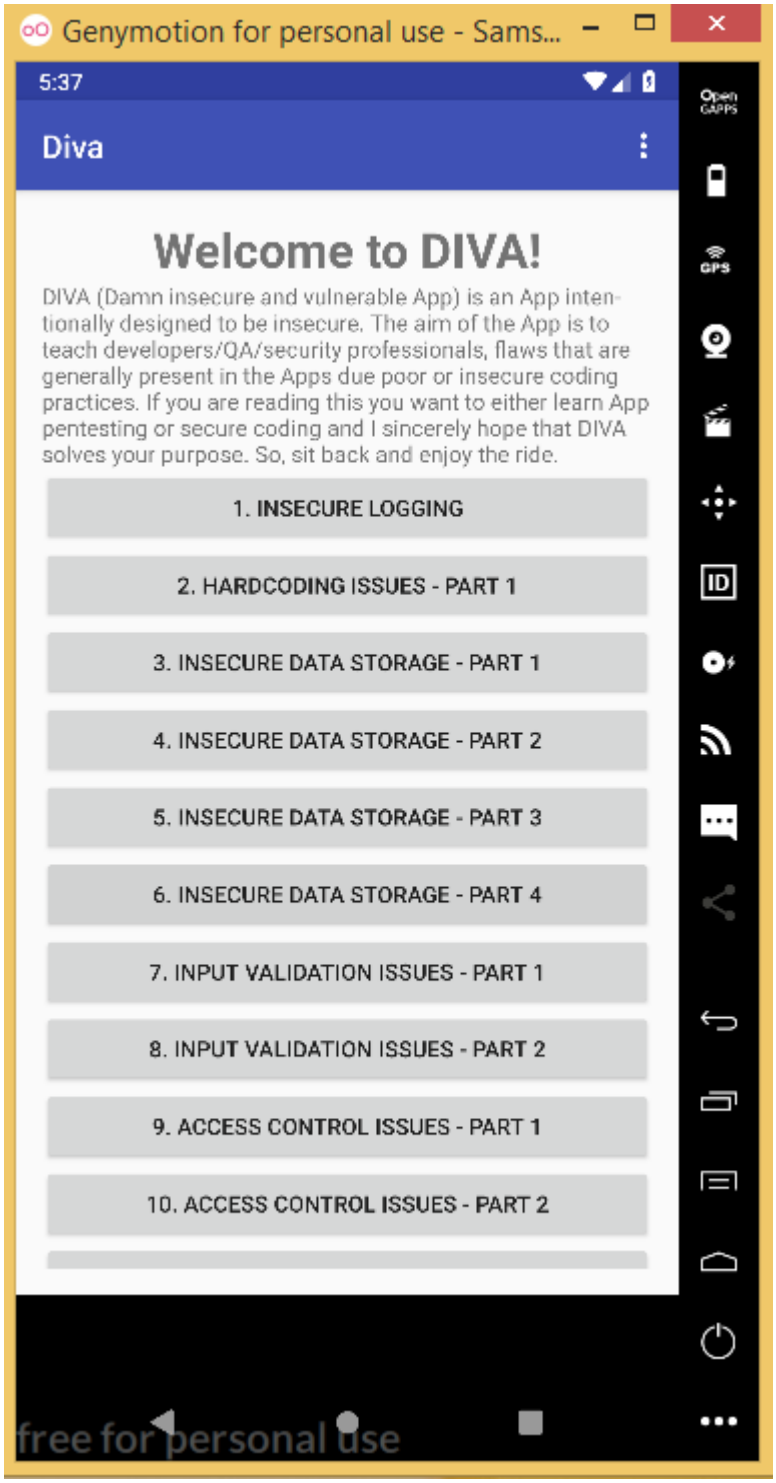
You will get success status printed on command line as shown in figure 1.4:



Icon of DIVA app will also appear on your VM as shown in figure 1.5 below:



Tap (Click) on the DIVA app Icon to launch the application.



INSECURE LOGGING:

Before solving this challenge please visit this LINK and read it. It is highly recommended.

Tap on Insecure Logging Button. A new activity will appear as shown in figure 1.7 below:

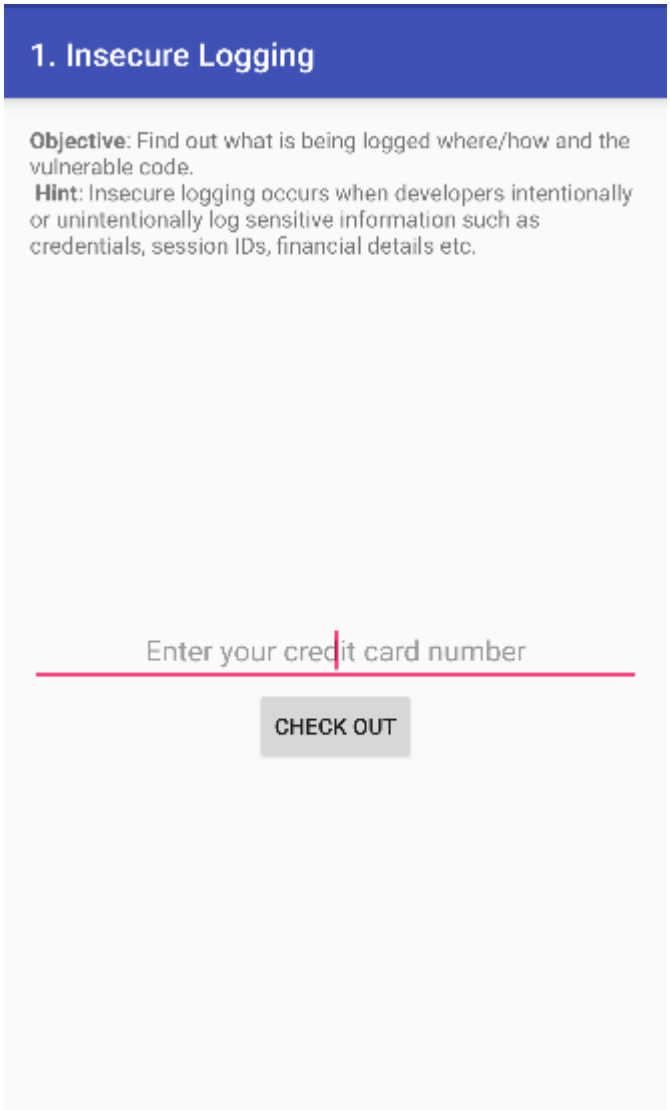


Figure 1.7

Now before typing on this screen go to your command line and execute command written and shown in figure 1.8 below:

adb shell

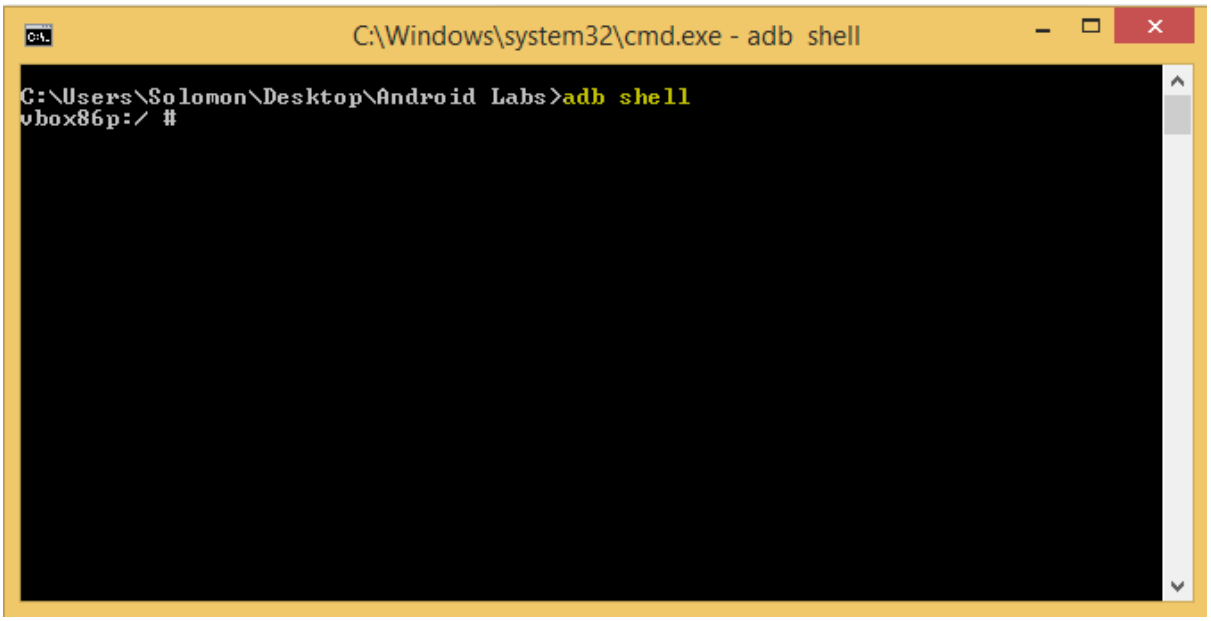


Figure 1.8

Shell will open there type the command:

logcat

Once you enter command logs will start appearing in front of you.

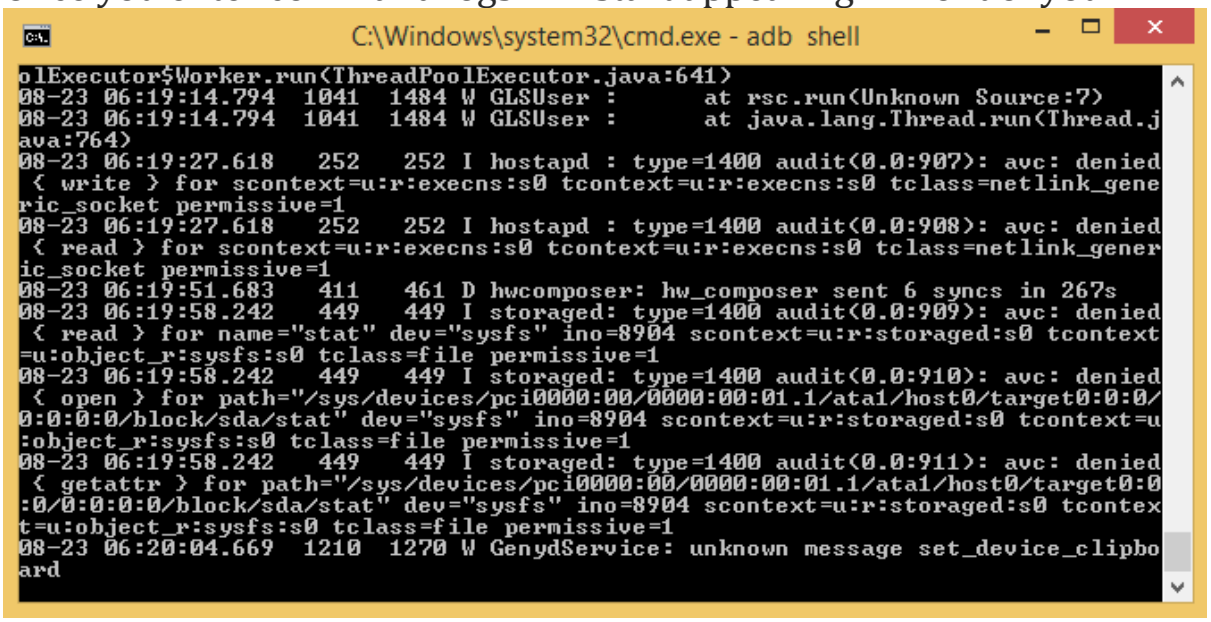


Figure 1.9

Now go to the Android VM and there enter credit card number

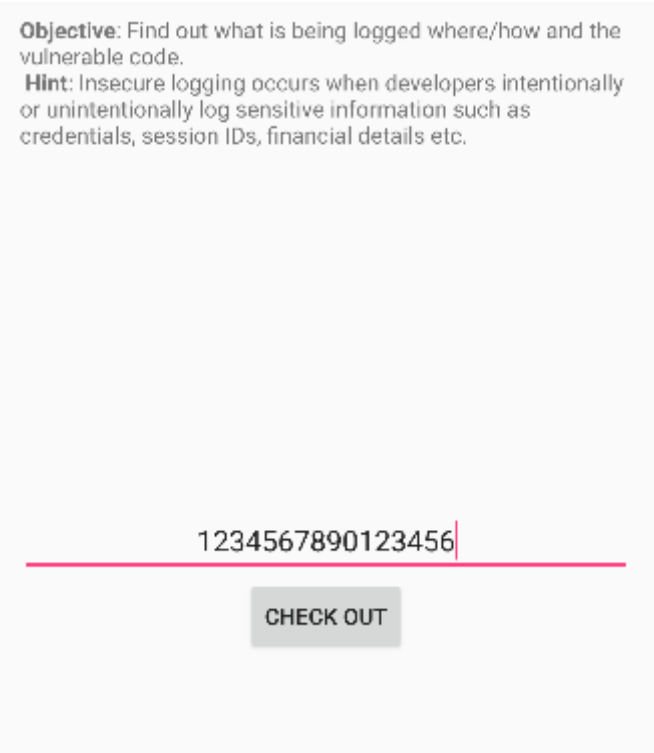


Figure 1.10

Now go back the the command line where logs are appearing you will find there Credit Card Number in plain text as shown in figure 1.11 below:

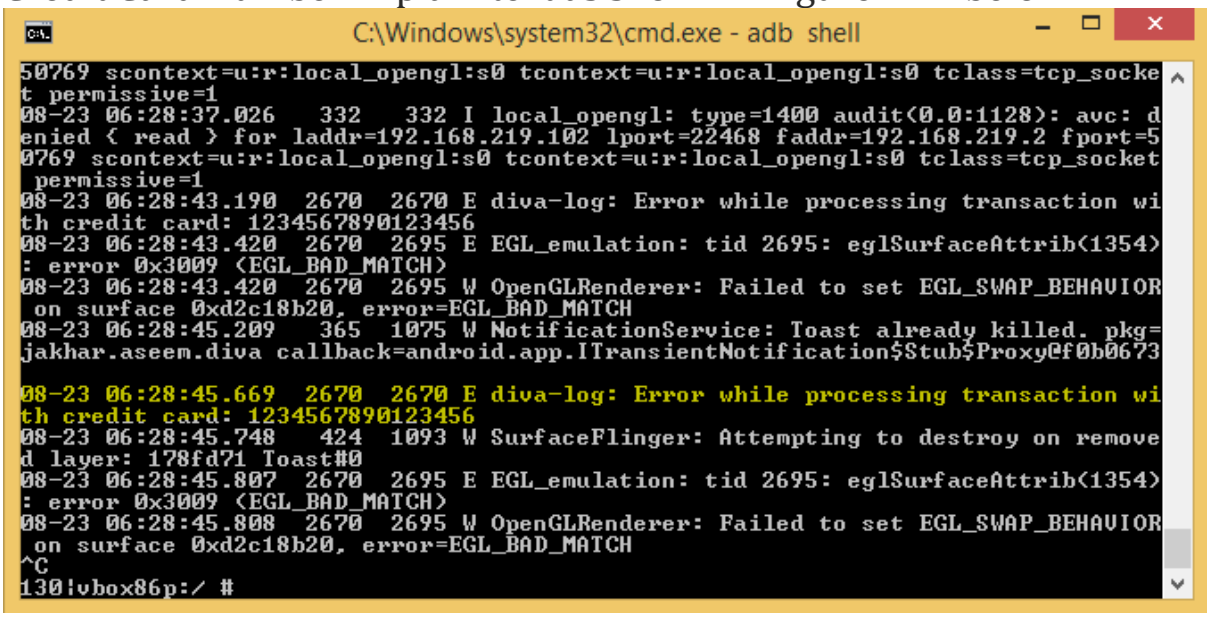


Figure 1.11

Here Insecure Logging challenge is completed.

HARDCODING ISSUES - PART 1:

Before solving this challenge please visit this LINK and read it. It is highly recommended.

Tap on Hardcoding Issues - Part 1 Button. A new activity will appear as shown in figure 1.12 below:

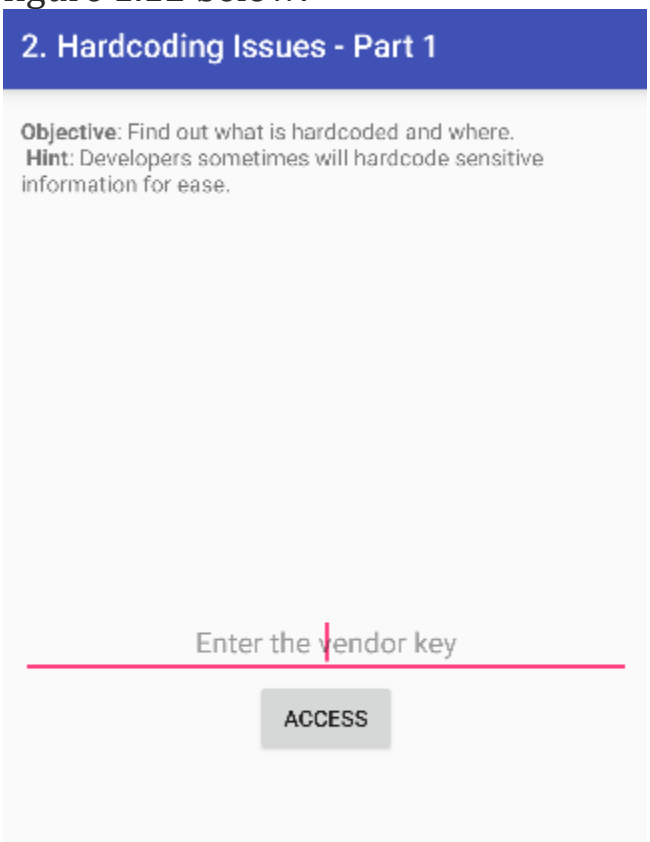


Figure 1.12

As this is hardcoding challenge this mean the Vendor Key is hardcoded in the application. In order to get the hardcoded key we need to do Reverse Engineering of this application.

You can use the Jadx-gui tool to find the code from APK

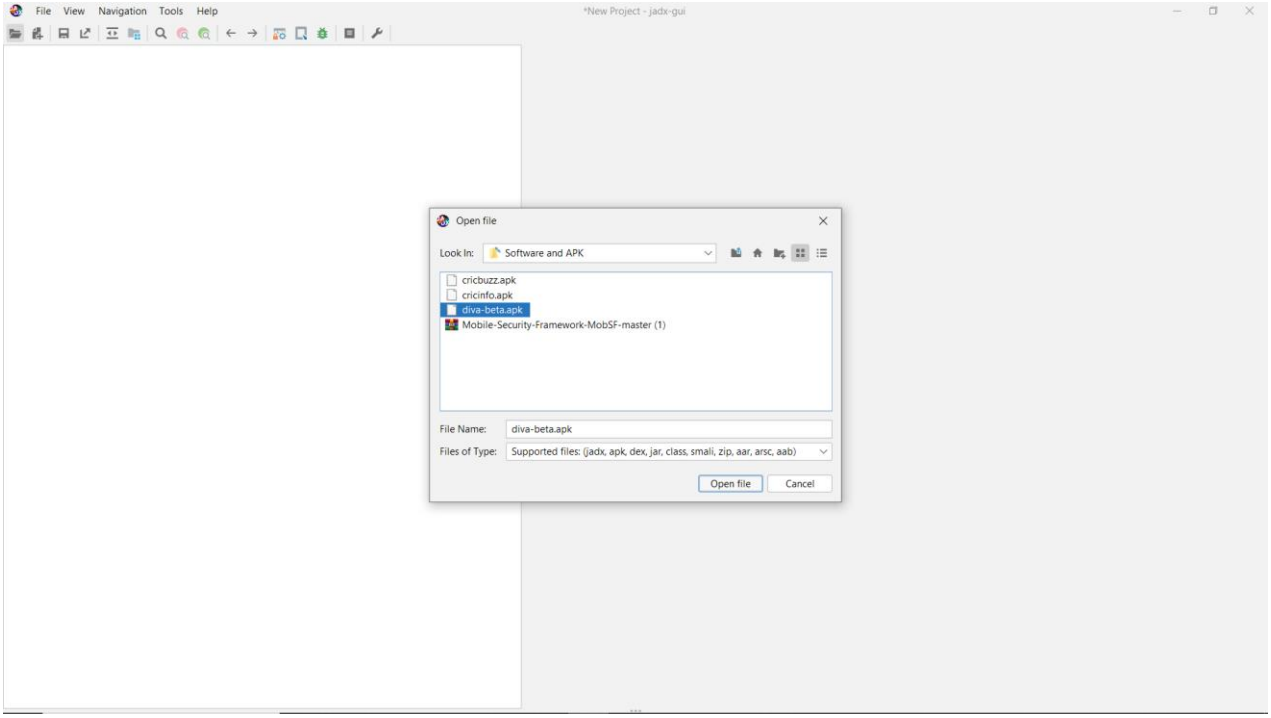


Figure 1.13

Just Open the diva-beta.apk as shown in figure

Now Just click on Resources -> res -> AndroidManifest.xml as shown in figure 1.14

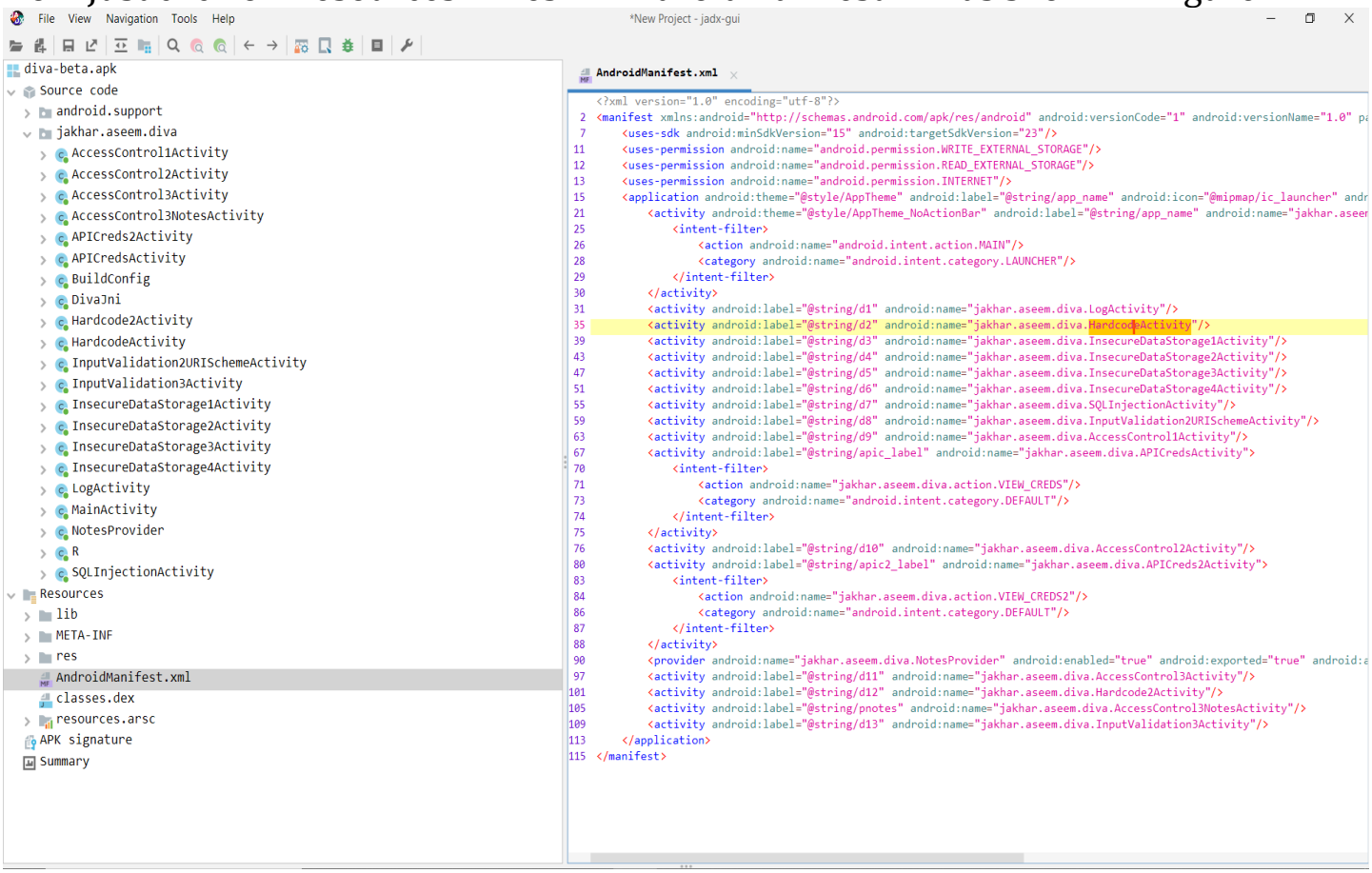


Figure 1.14

Now You can see that HardCodeActivity file is available on left side. Just click on that file. You will find that vendorsecretkey is the hardcoded password.

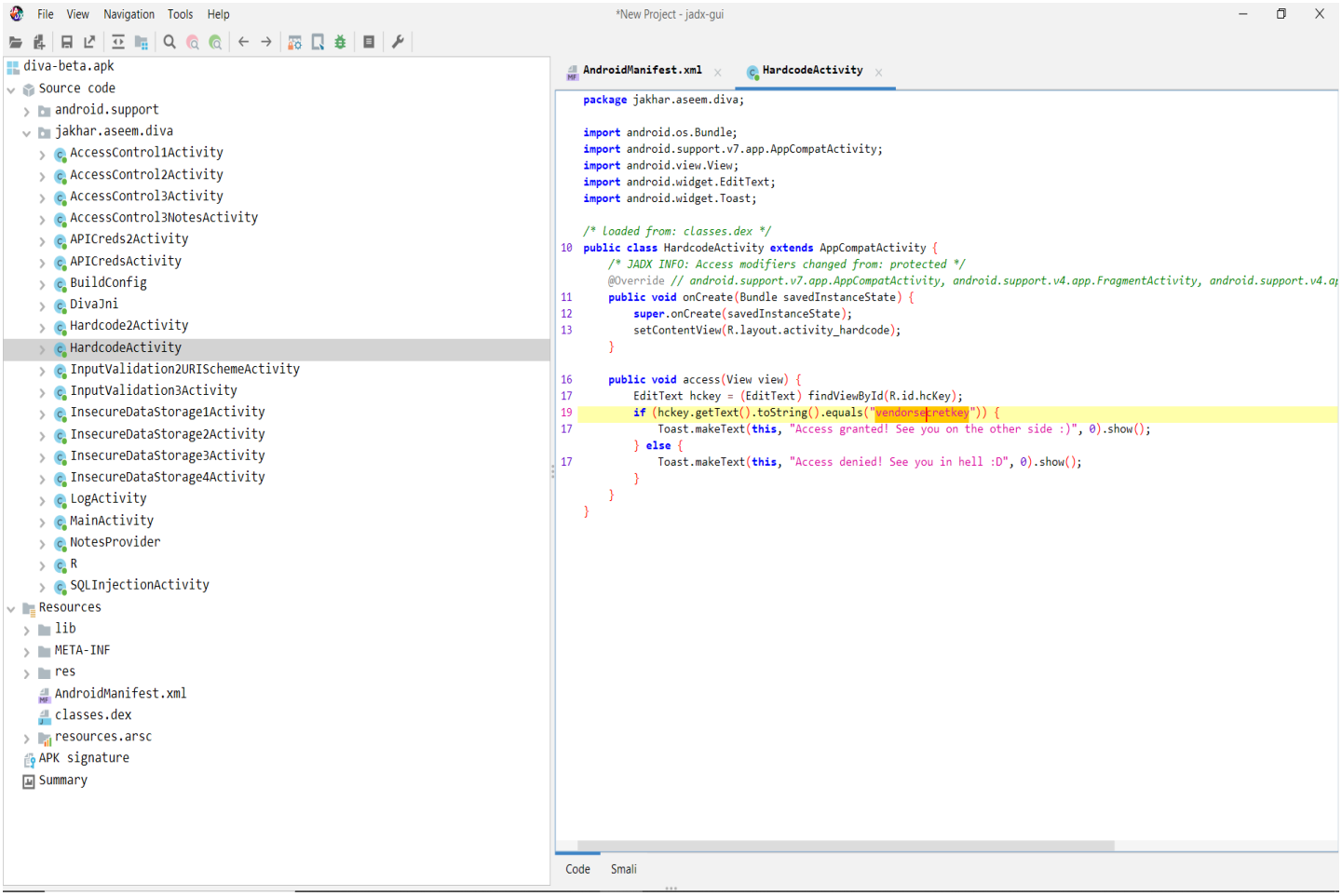


Figure 1.15

Now we got the vendor's secret key. Enter the Secret key to get access in app as shown in figure 1.16 below:

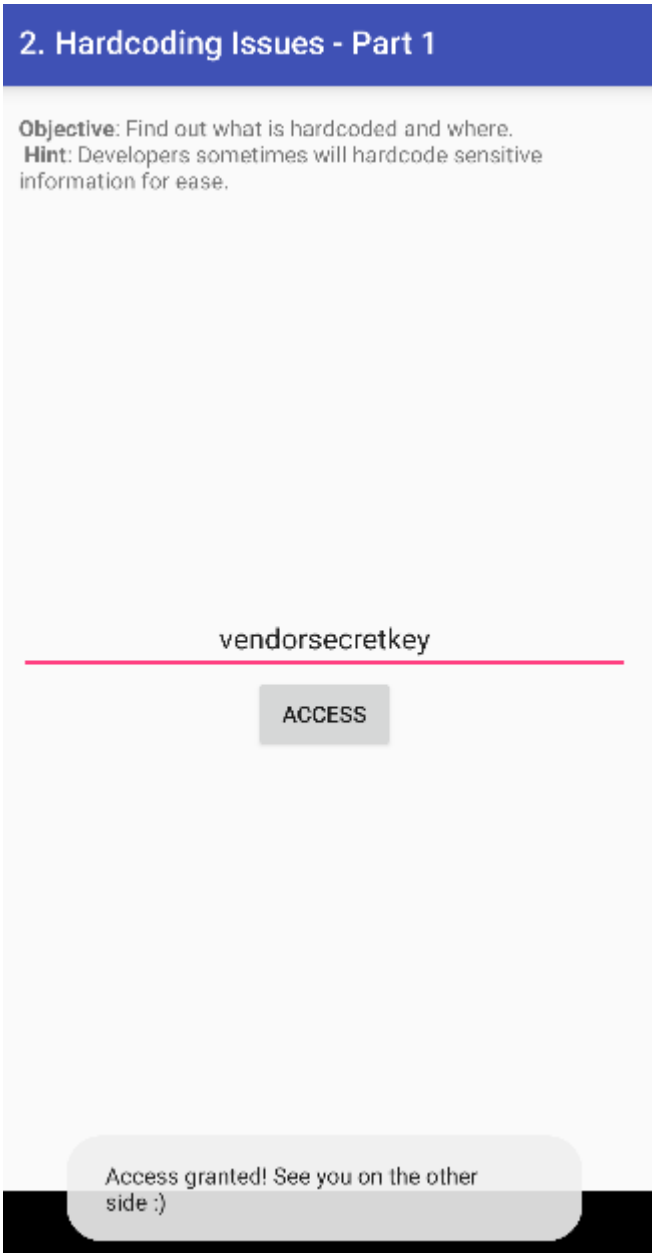


Figure 1.16

Here Hardcoding Issue - Part 1 challenge is completed.

INSECURE DATA STORAGE - PART 1:

In order to complete this challenge we have to move around in Directories with the help of shell. But first review the source code of this activity. We can see that credentials are stored in Shared Preferences as shown in figure 1.17 below:

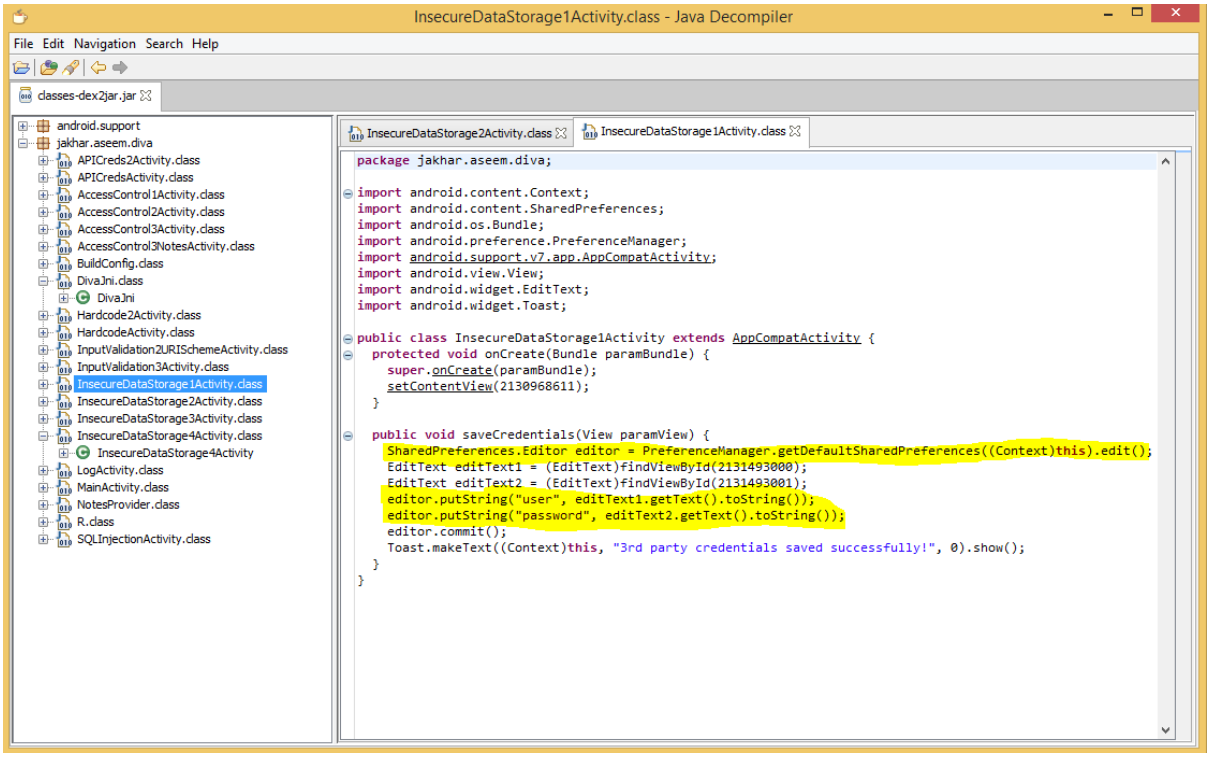


Figure 1.17

After accessing shell with adb go to /data/data/. This is the location where packages of all installed applications are stored. Find the package of diva application and then access it as shown in figure 1.18 below:

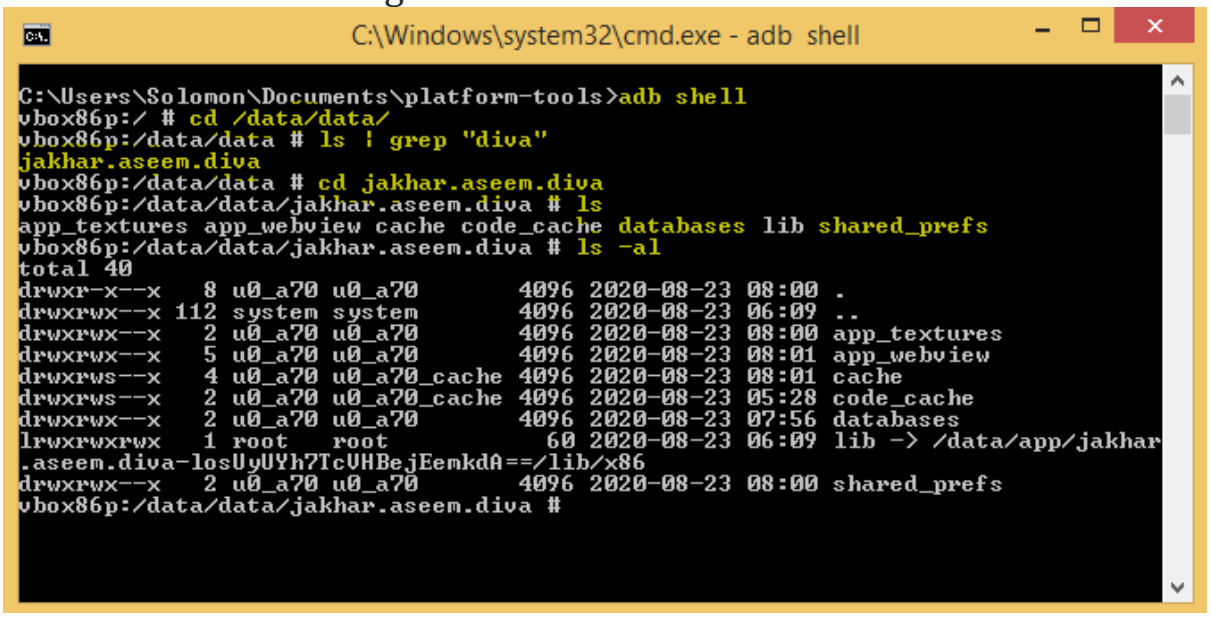


Figure 1.18

First enter username and password and save them before try to find them in these directories.

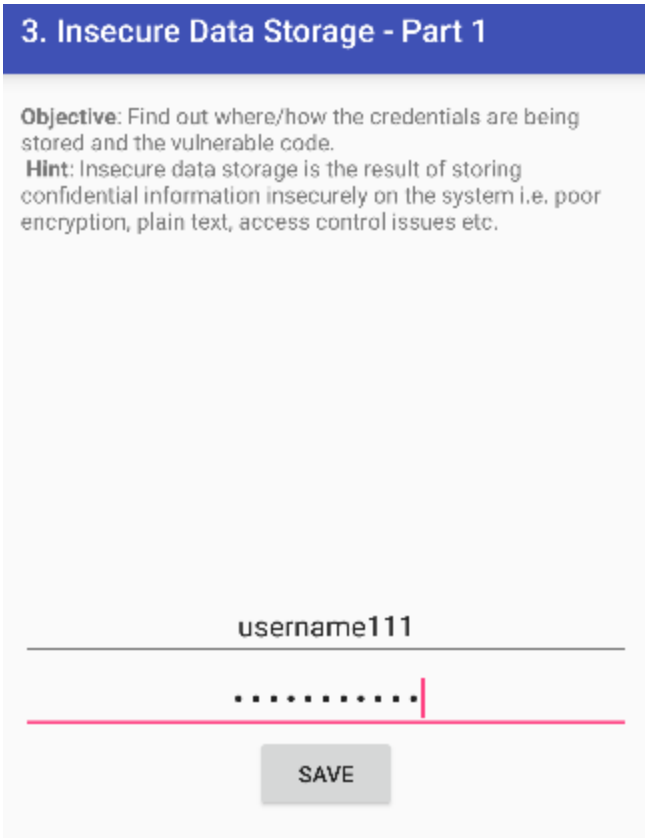


Figure 1.19

Now from shell find them. App is saving these credentials in shared preferences as shown in figure 1.20 below:

In order to view contents of XML file to get username and password type following command:

cat jakhar.aseem.diva_preferences.xml

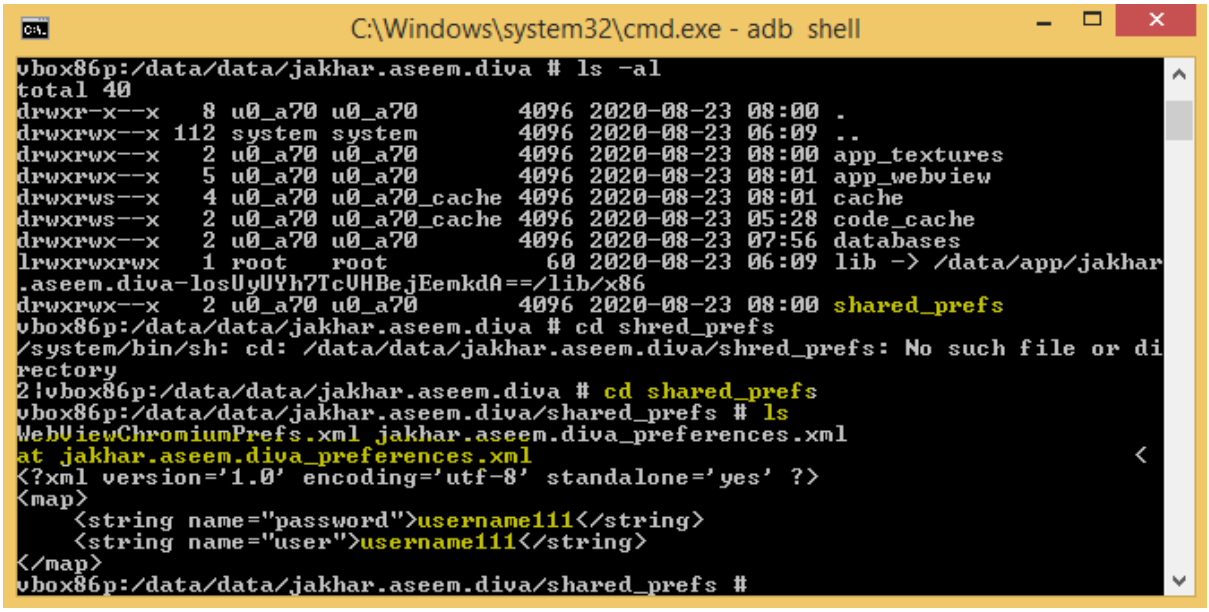


Figure 1.20

Here Insecure Data Storage - Part 1 challenge is completed.

INSECURE DATA STORAGE - PART 2:

This is similar challenge to previous one but credentials are stored in different location.

Before going to solve this challenge save credentials from in application.

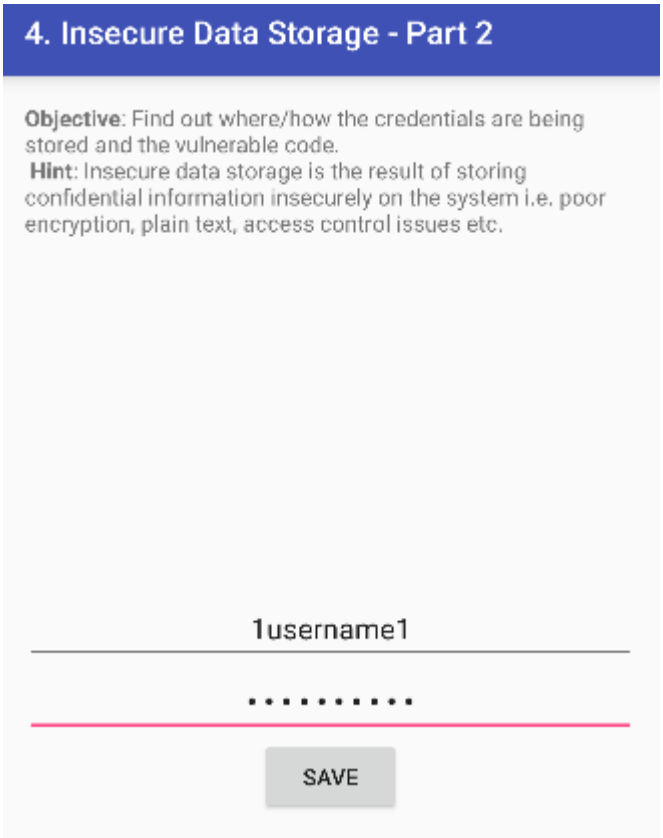


Figure 1.21

This time credentials were stored in database ids2 and in its myuser table as shown in figure 1.22 below:

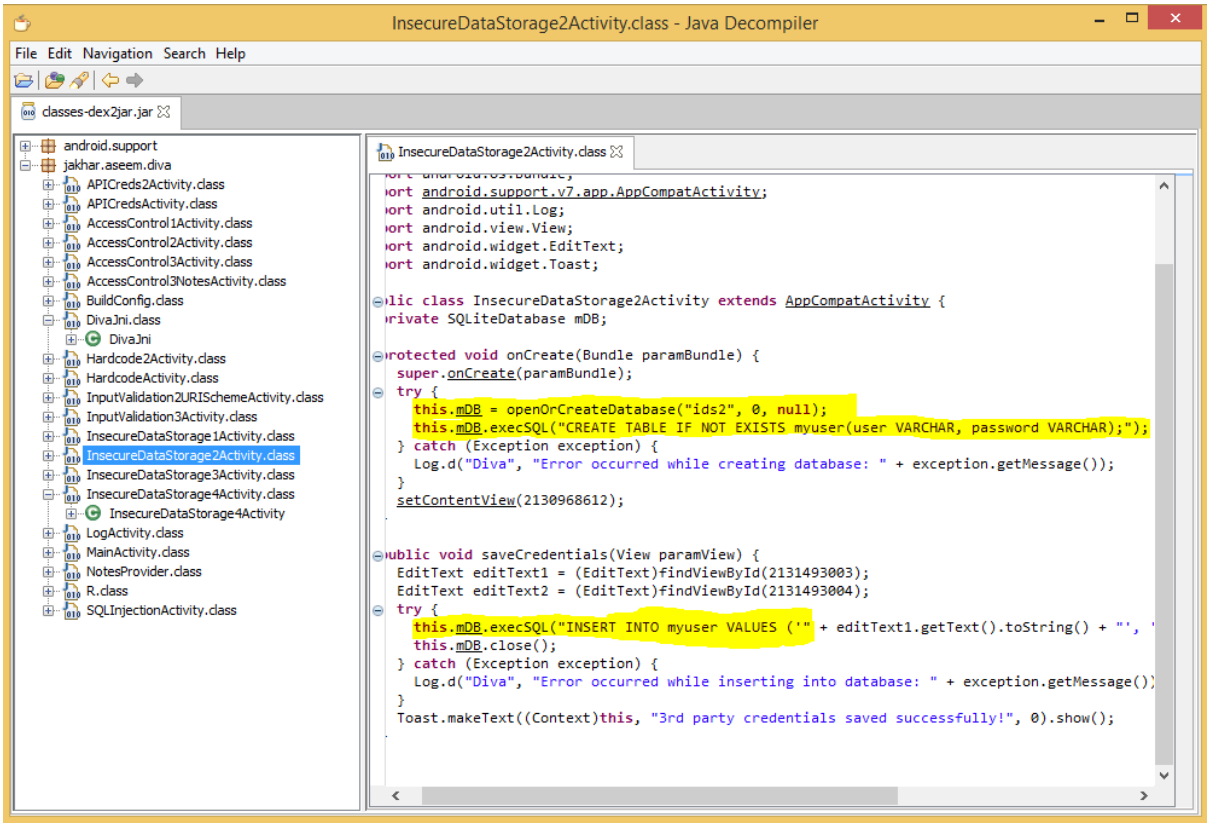


Figure 1.22

In order to access database files from command line we have a tool in platform tools folder named “sqlite3”.

Type following command in linux (android) shell:

sqlite3 <database_name>

sqlite3 ids2

sqlite>.tables

This .tables command will show all of the tables available in that particular database.

to exit from this tool .exit command is used.

sqlite>.exit

This will return you to your previous shell on which you were working.

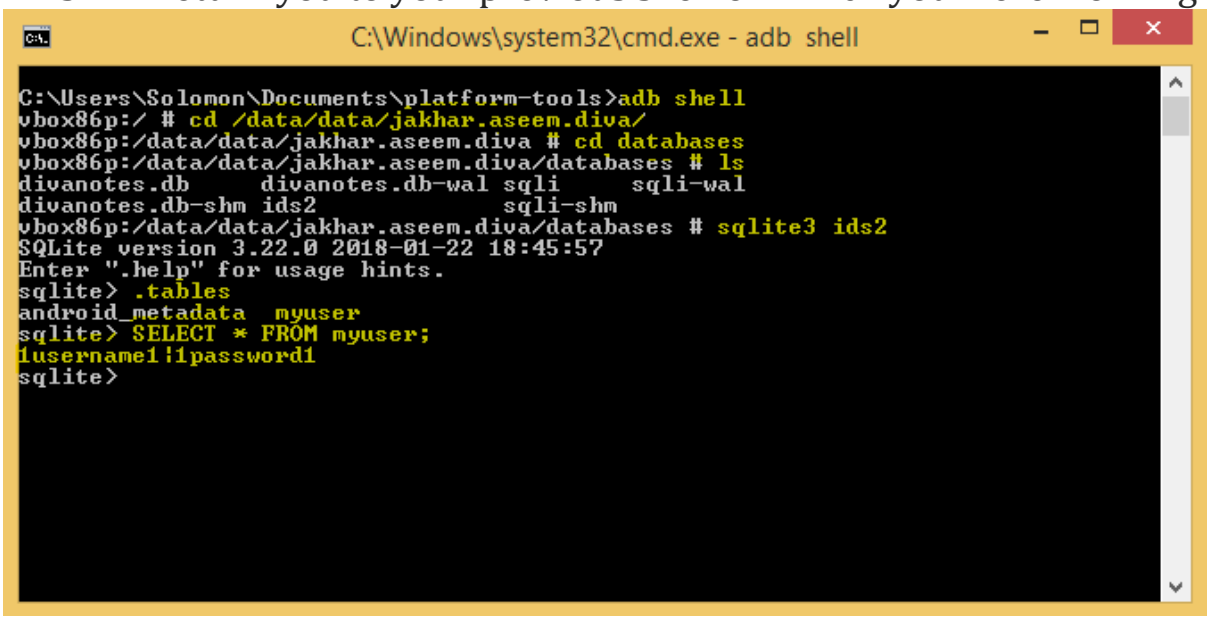


Figure 1.23

Here Insecure Data Storage - Part 2 challenge is completed.

INSECURE DATA STORAGE - PART 3:

Enter the credentials from application.

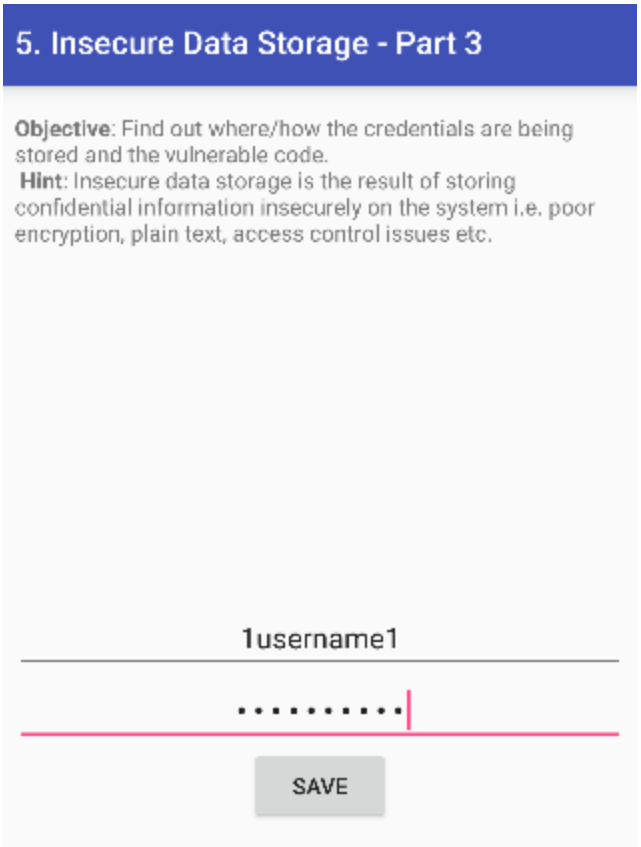


Figure 1.24

The credentials were stored in temporary file as shown in figure 1.25 below:

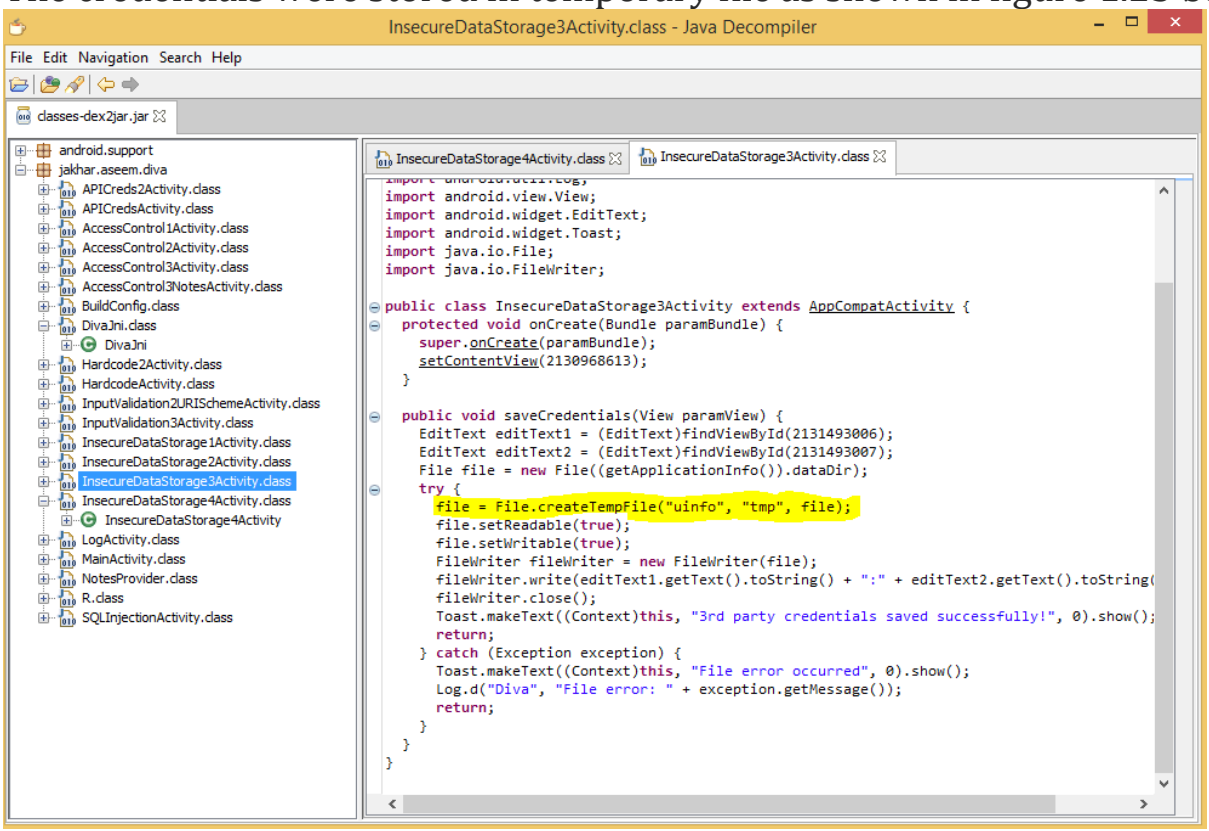


Figure 1.25

Let's access those temporary file from shell.

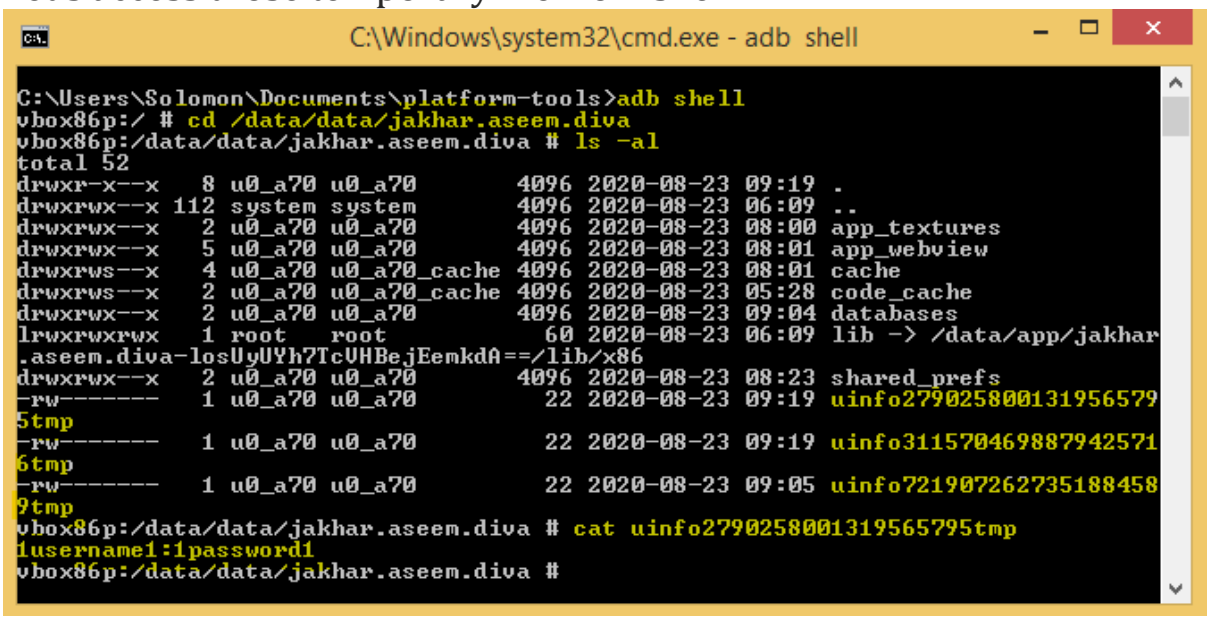


Figure 1.26

Here Insecure Data Storage - Part 3 challenge is completed.

INSECURE DATA STORAGE - PART 4:

Enter the credentials from application.

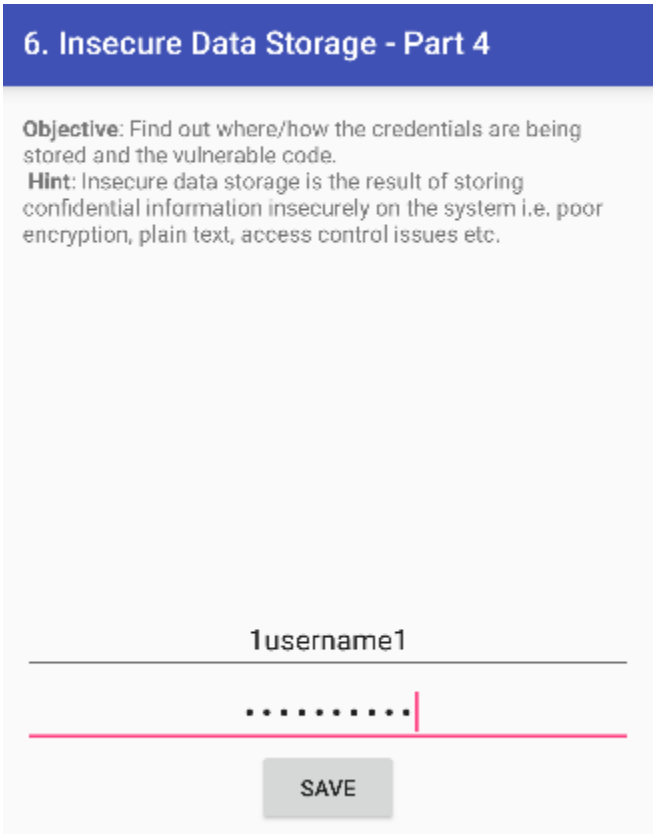


Figure 1.27

The app is storing credentials in external storage.

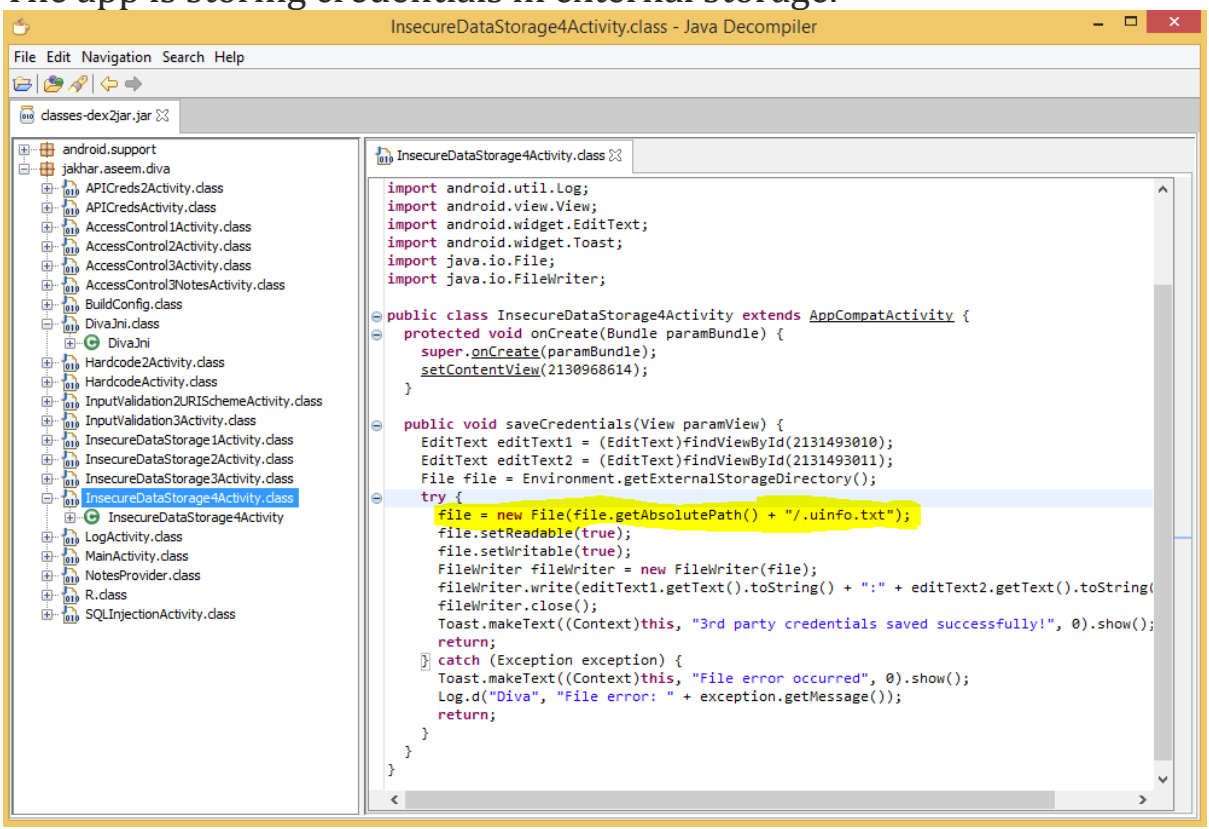


Figure 1.28

We got the location, now access them from shell as shown in figure below 1.29:

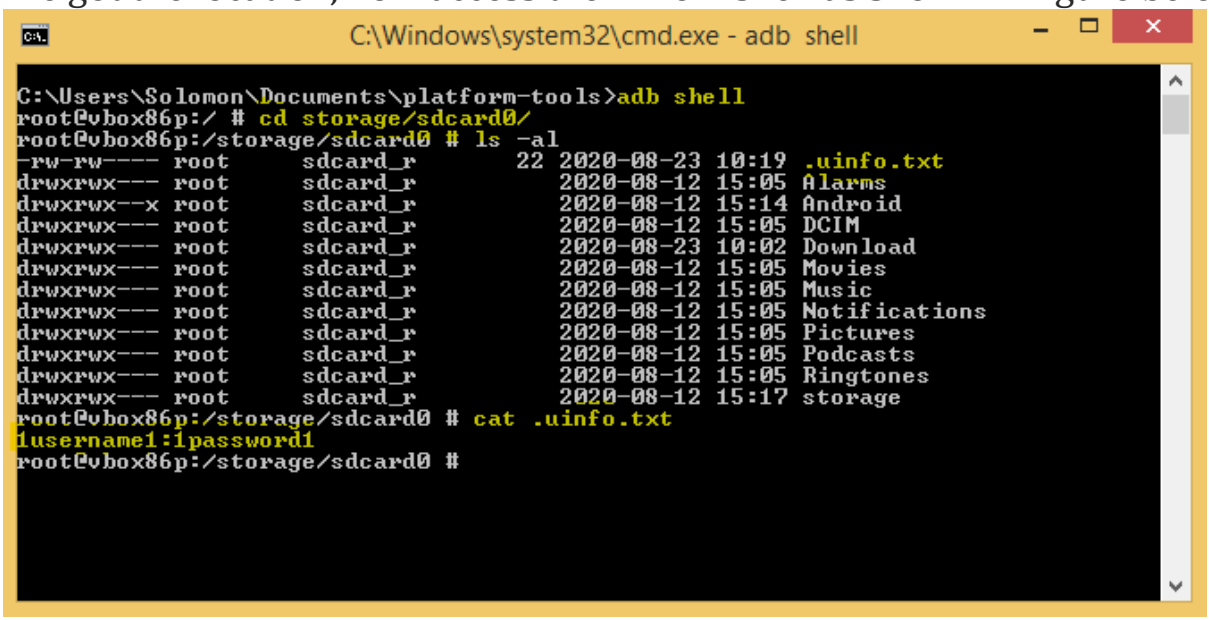


Figure 1.29

Here Insecure Data Storage - Part 4 challenge is completed.

INPUT VALIDATION ISSUES - PART 1:

This challenge is about SQL Injection.

First try to enter single quote (‘) as input and check result.

Try to enter single quote twice (‘’) and then check result.

You will see the difference in the output of the toast.

Once you realize that your inputs are working then play with text field.

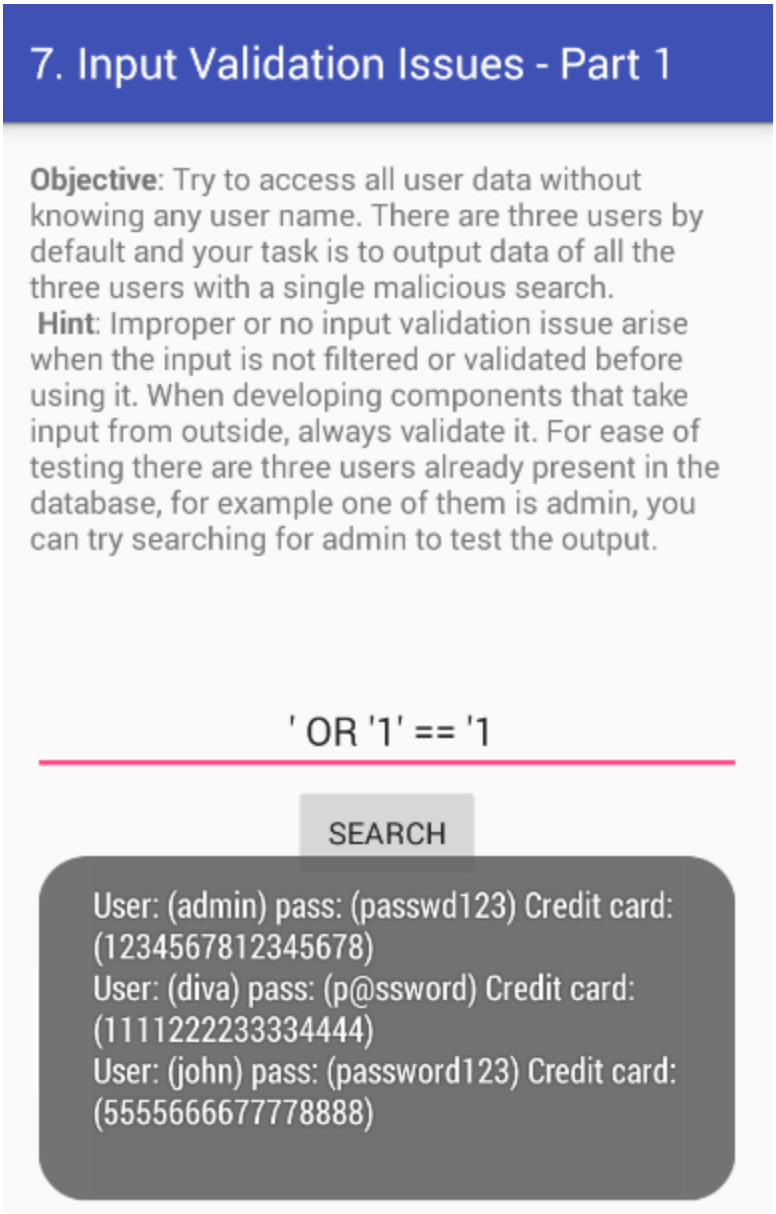


Figure 1.30

Here Input Validation Issues - Part 1 challenge is completed.

INPUT VALIDATION ISSUES - PART 2:

In this challenge we have to access local files using URL.

first let's try to access Google as shown in figure 1.31 below:

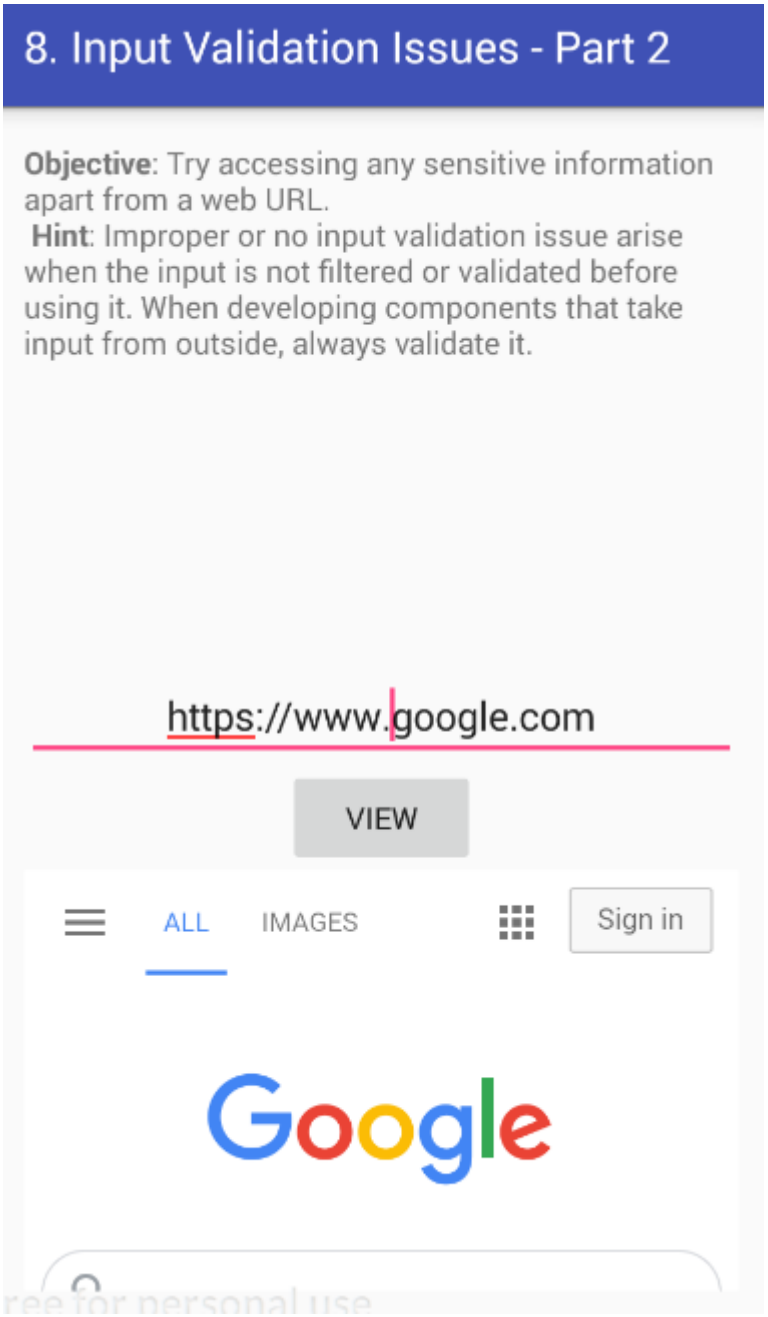


Figure 1.31

Let's try to change the URL and try to access a file from device.

file:////etc/hosts

You can see it worked in figure 1.32 below:

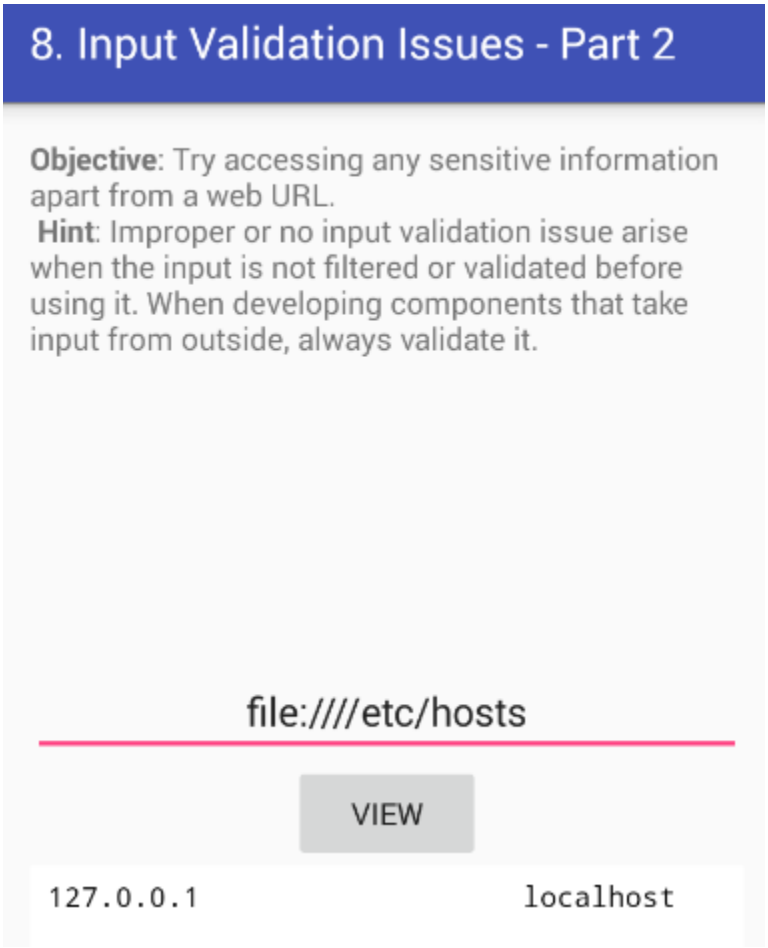


Figure 1.32

Now let's try to access a file from shared preferences where credentials are stored. It is the file we saw in earlier challenges.

file:///data/data/jakhar.aseem.diva/shared_prefs/jakhar.aseem.diva_preferences.xml

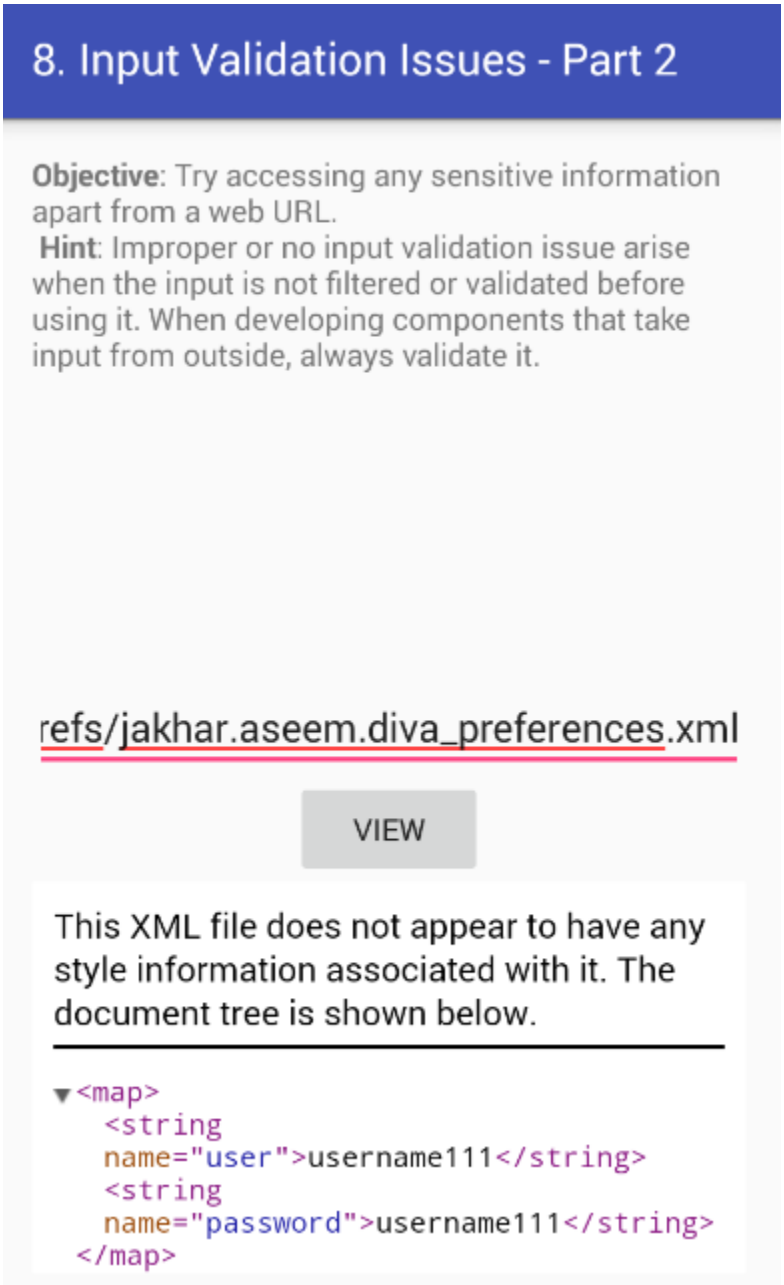


Figure 1.33

Here Input Validation Issues - Part 2 challenge is completed.

ACCESS CONTROL ISSUES - Part 1:

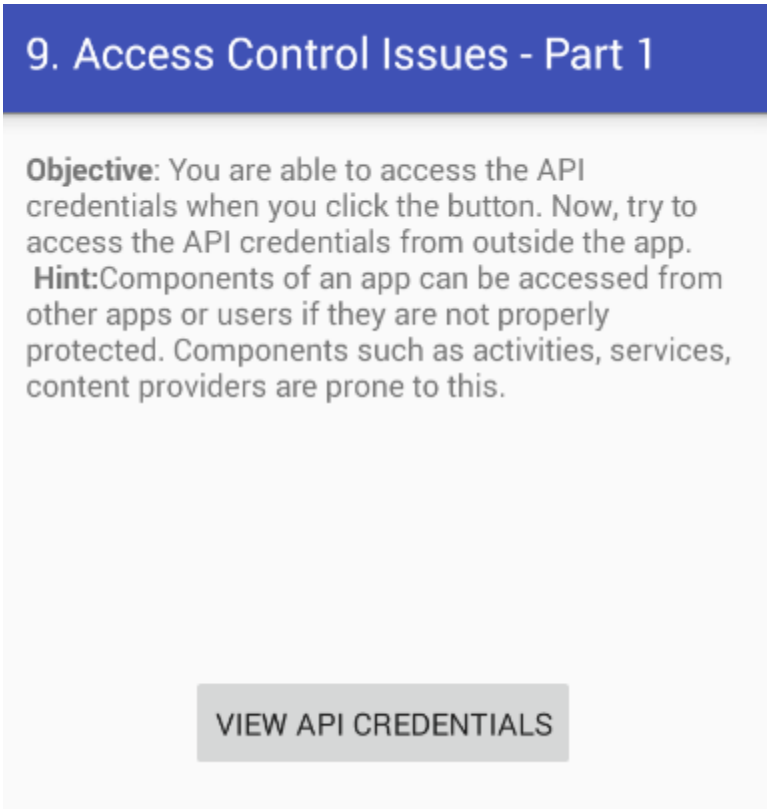


Figure 1.34

Accessing credentials from "View API Credentials" Button is completely legal. There is no issue in it. We need to check that can we directly access credentials without going through this activity or this checkpoint. In order to do that first we have to get the name of activity which will appear after it, for that we take the help of logcat which is discussed earlier.

Run logcat command from android shell then click on "View API Credentials" Button a log will generated related to this which gives us name of next activity as shown in figure 1.35 below:

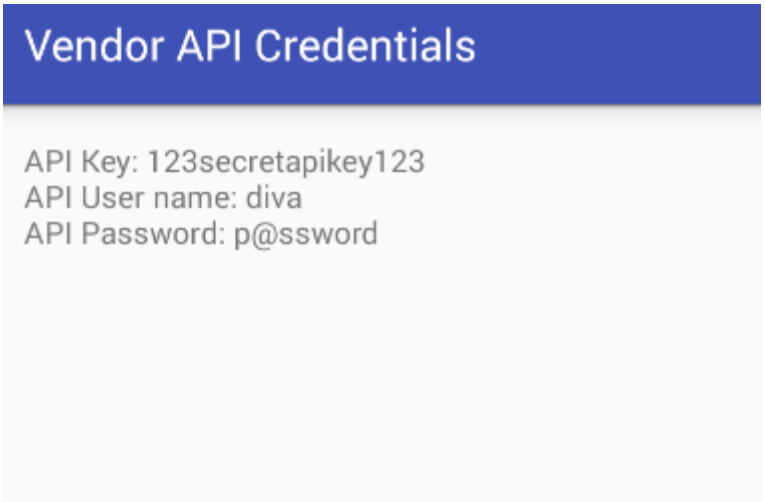


Figure 1.35

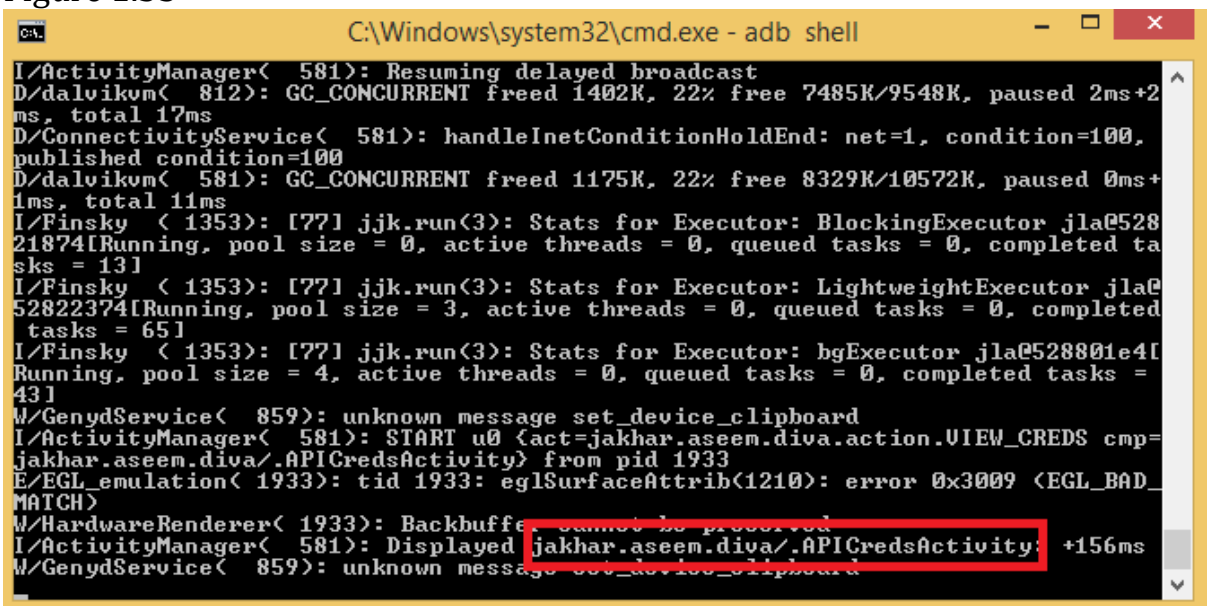


Figure 1.36

As we got the activity name. Now let's try to access it from adb activity manager directly.

adb shell am start -n jakhar.aseem.diva/.APICredsActivity

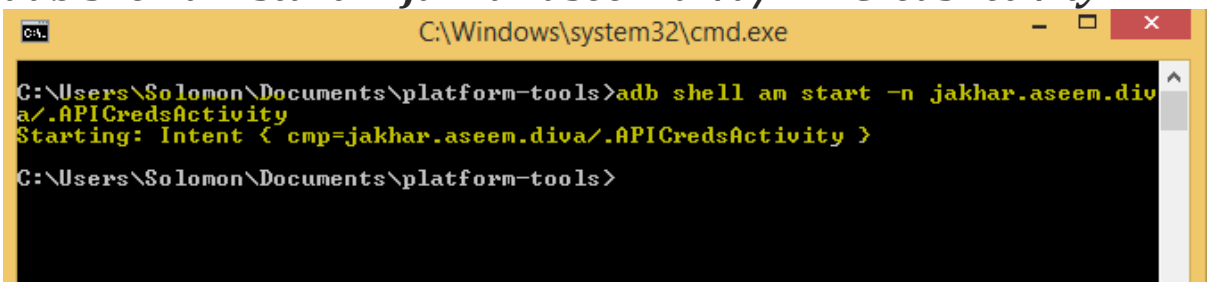


Figure 1.37

As you can see we got access of API Credentials without any restriction or authentication. This also means that other apps can also access these credentials.

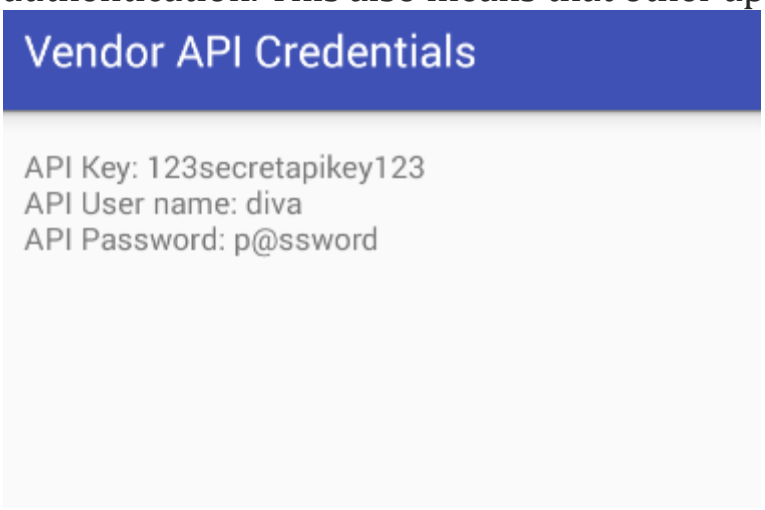


Figure 1.38

Here Access Control Issues - Part 1 challenge is completed.

ACCESS CONTROL ISSUES - PART 2:

First we have to de compile the application. We use APKTOOL for it.

On Command Line type the following command:

apktool_2.4.1.jar d diva-beta.apk

```
C:\Users\Solomon\Desktop\Android Labs>apktool_2.4.1.jar d diva-beta.apk
C:\Users\Solomon\Desktop\Android Labs>
```

Figure 1.39

A folder of application name will be created in same directory of apk. From there open AndroidManifest.XML file:

AndroidManifest.xml - Notepad

```
        <activity android:label="@string/d5" android:name="jakhar.aseem.diva.InsecureDataStorage3Activity"/>
        <activity android:label="@string/d6" android:name="jakhar.aseem.diva.InsecureDataStorage4Activity"/>
        <activity android:label="@string/d7" android:name="jakhar.aseem.diva.SQLInjectionActivity"/>
        <activity android:label="@string/d8" android:name="jakhar.aseem.diva.InputValidation2URISchemeActivity"/>
        <activity android:label="@string/d9" android:name="jakhar.aseem.diva.AccessControl1Activity"/>
        <activity android:label="@string/apic_label" android:name="jakhar.aseem.diva.APICredsActivity">
            <intent-filter>
                <action android:name="jakhar.aseem.diva.action.VIEW_CREDS"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
        <activity android:label="@string/d10" android:name="jakhar.aseem.diva.AccessControl2Activity"/>
        <activity android:label="@string/apic2_label" android:name="jakhar.aseem.diva.APICreds2Activity">
            <intent-filter>
                <action android:name="jakhar.aseem.diva.action.VIEW_CREDS2"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
        <provider android:authorities="jakhar.aseem.diva.provider.notesprovider" android:enabled="true" android:exported="
        <activity android:label="@string/d11" android:name="jakhar.aseem.diva.AccessControl3Activity"/>
        <activity android:label="@string/d12" android:name="jakhar.aseem.diva.Hardcode2Activity"/>
        <activity android:label="@string/pnotes" android:name="jakhar.aseem.diva.AccessControl3NotesActivity"/>
        <activity android:label="@string/d13" android:name="jakhar.aseem.diva.InputValidation3Activity"/>
    </application>
</manifest>
```

Figure 1.40

Let's open the Java code file and inspect there about any new thing.

APICreds2Activity.class - Java Decompiler

```
package jakhar.aseem.diva;

import android.content.Context;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class APICreds2Activity extends AppCompatActivity {
  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130968606);
    TextView textView = (TextView)findViewById(2131492983);
    EditText editText = (EditText)findViewById(2131492984);
    Button button = (Button)findViewById(2131492985);
    if (!getIntent().getBooleanExtra(getString(2131099686), true)) {
      textView.setText("TVEETER API key: secrettveeterapikey\nAPI User name: diva2\nAPI Password: p@ssu
      return;
    }
    textView.setText("Register yourself at http://payatu.com to get your PIN and then login with that P
    editText.setVisibility(0);
    button.setVisibility(0);
  }

  public void viewCreds(View paramView) {
    Toast.makeText((Context)this, "Invalid PIN. Please try again", 0).show();
  }
}
```

Figure 1.41

Search this highlighted parameter in R.class.

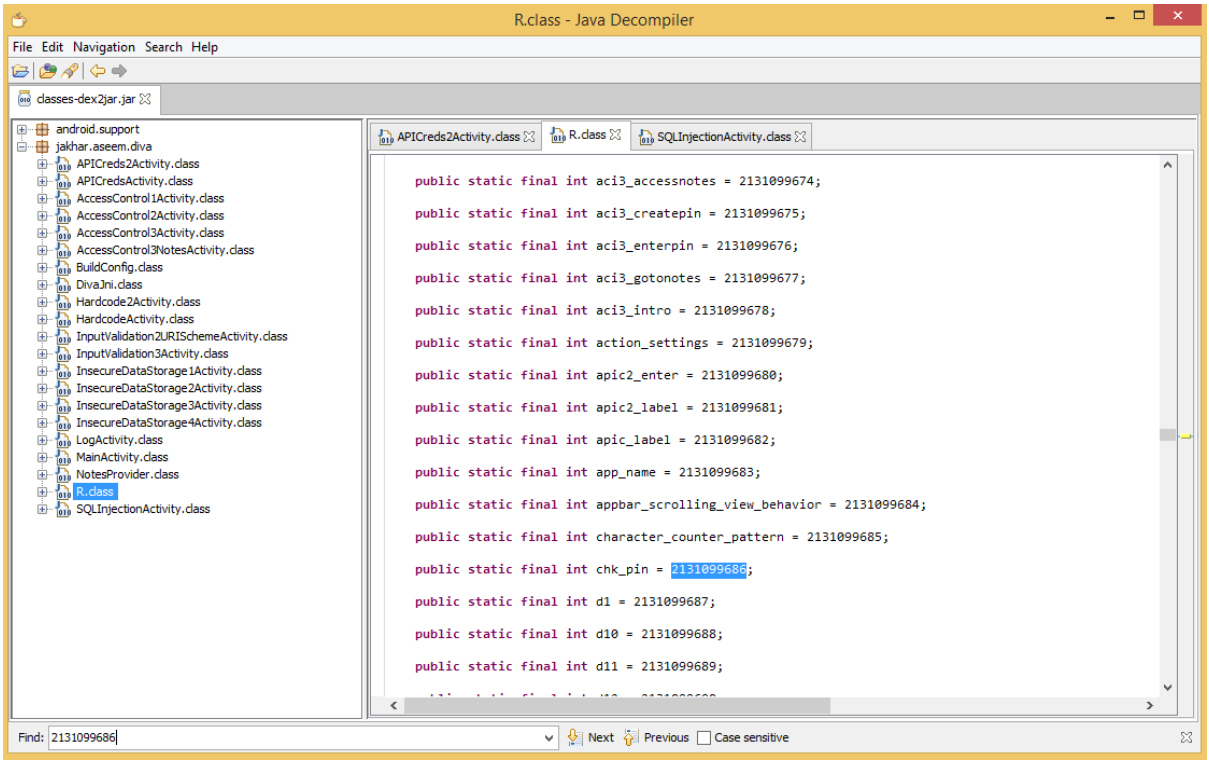


Figure 1.42

In order to get the value of this chk_pin we have to inspect the strings.XML file which is located in application decompiled folder /res/values/string.xml

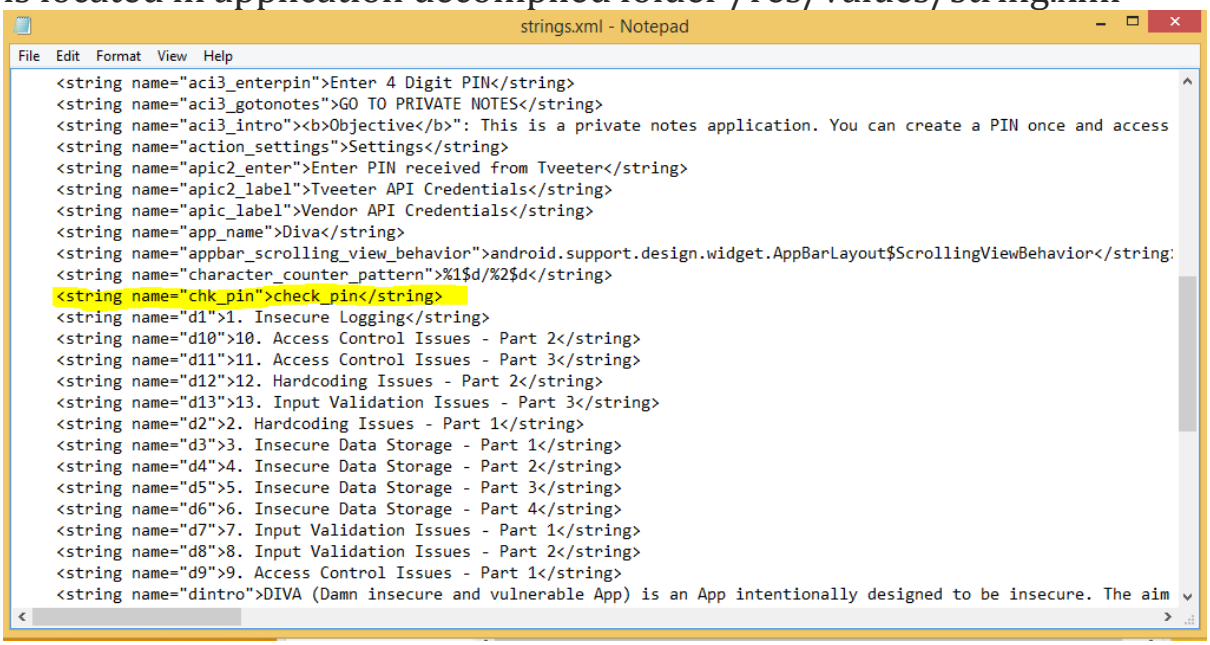


Figure 1.43

Let's try to access the credentials with these details we have found and disabling check pin.

adb shell am start -n jakhar.aseem.diva/.APICreds2Activity -a jakhar.aseem.diva.action.View_CREDS2 --ez check_pin false

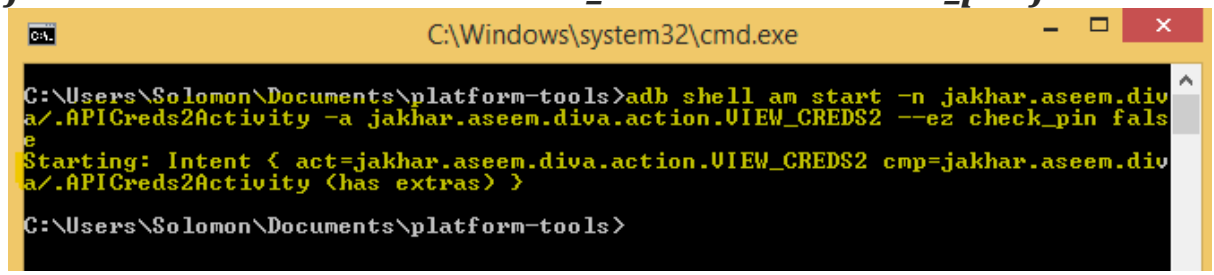


Figure 1.44

Now you will see in your VM Credentials appeared.

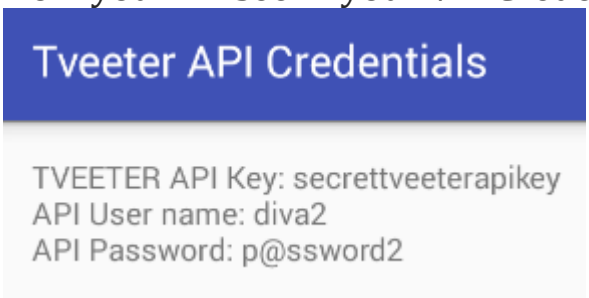


Figure 1.45

Here Access Control Issues - Part 2 challenge is completed.

ACCESS CONTROL ISSUES - PART 3:

Apparently we cannot access notes without pin. Let's try to access activity using content provider.

adb shell content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes/

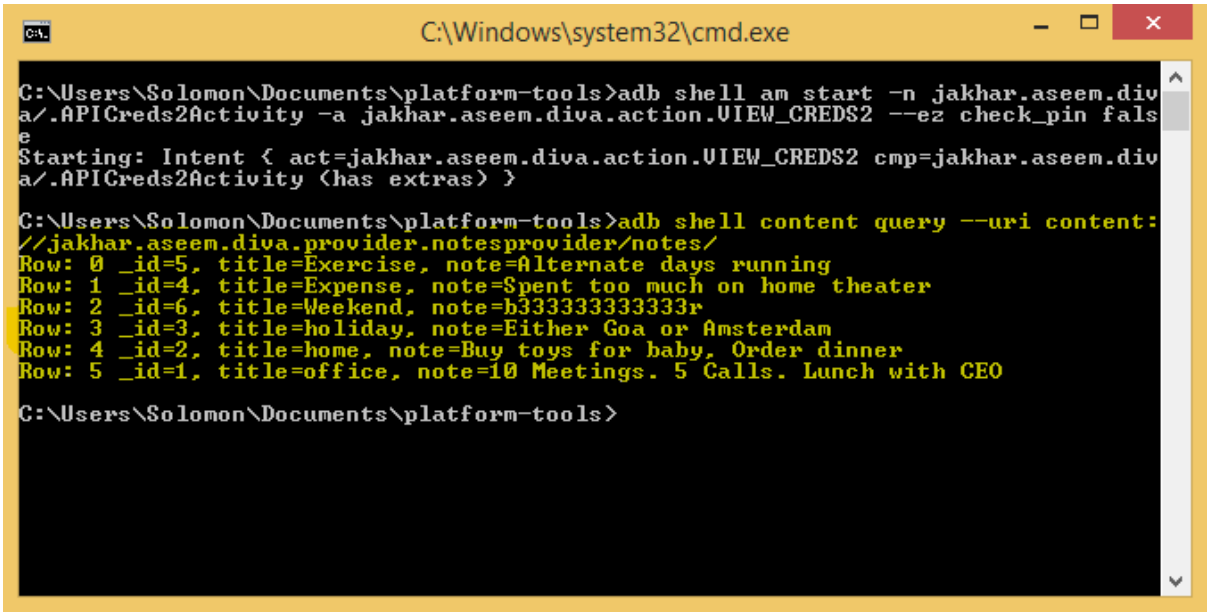


Figure 1.46

Here Access Control Issues - Part 3 challenge is completed.

HARDCODING ISSUES - PART 2:

Pull libdivajni.so file from VM using adb.

adb pull /data/data/jakhar.aseem.diva/lib/libdivajni.so

Use strings tool to get strings from libdivajni.so file.

strings libdivajni.so

After trying different strings finally highlighted string worked.

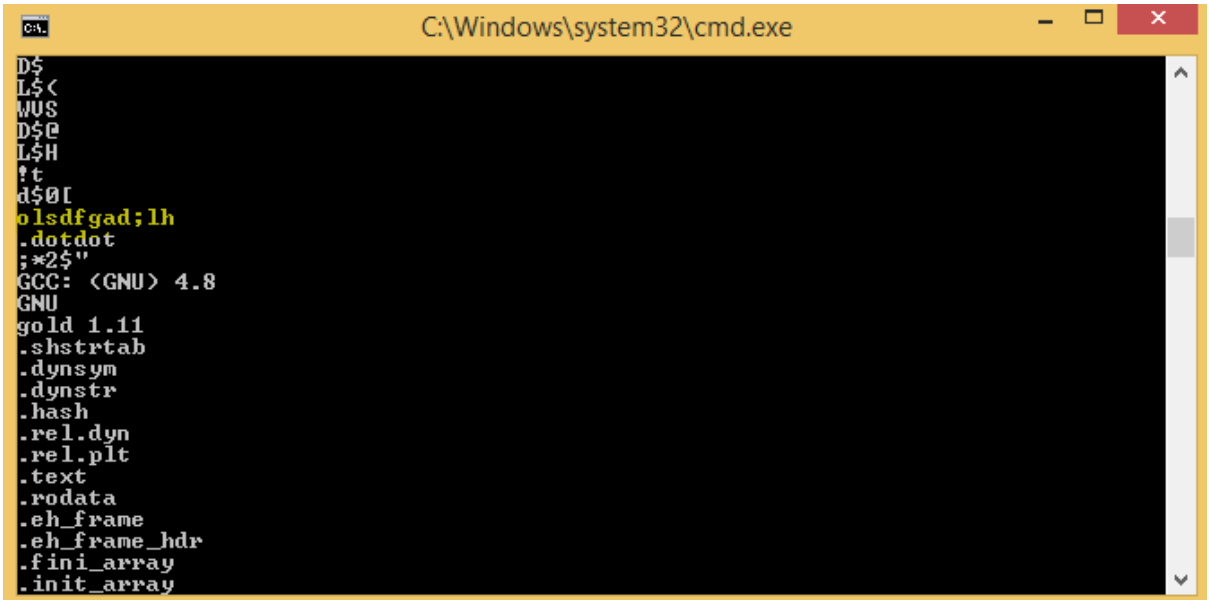

C:\Windows\system32\cmd.exe
D$
L$(
WVS
D$@
L$H
!t
d$0[
olsdfgad;lh
.dotdot
;*2$"
GCC: (GNU) 4.8
GNU
gold 1.11
.shstrtab
.dynsym
.dynstr
.hash
.rel.dyn
.rel.plt
.text
.rodata
.eh_frame
.eh_frame_hdr
.fini_array
.init_array

Figure 1.47

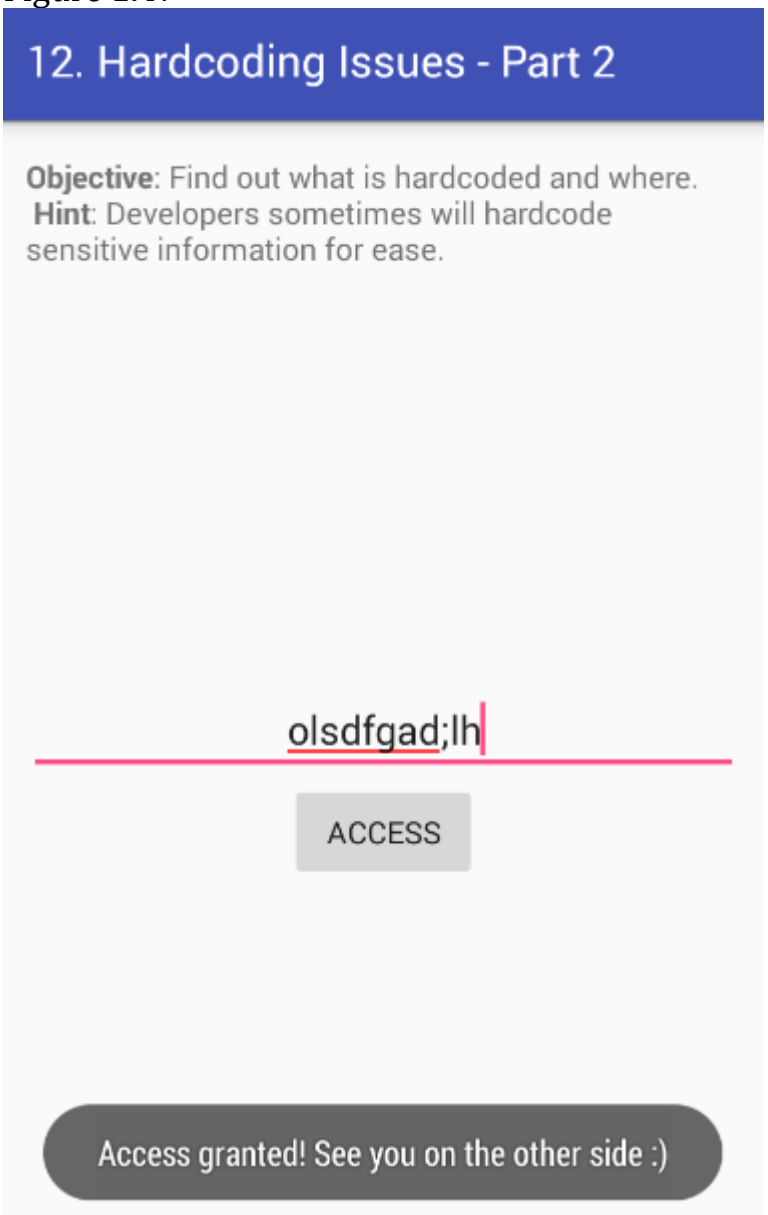

12. Hardcoding Issues - Part 2
Objective: Find out what is hardcoded and where.
Hint: Developers sometimes will hardcode sensitive information for ease.
olsdfgad;lh
ACCESS
Access granted! See you on the other side :)

Figure 1.48

Here Hardcoding Issues - Part 2 challenge is completed.

INPUT VALIDATION ISSUES - PART 3:

In this challenge goal is to crash the application.

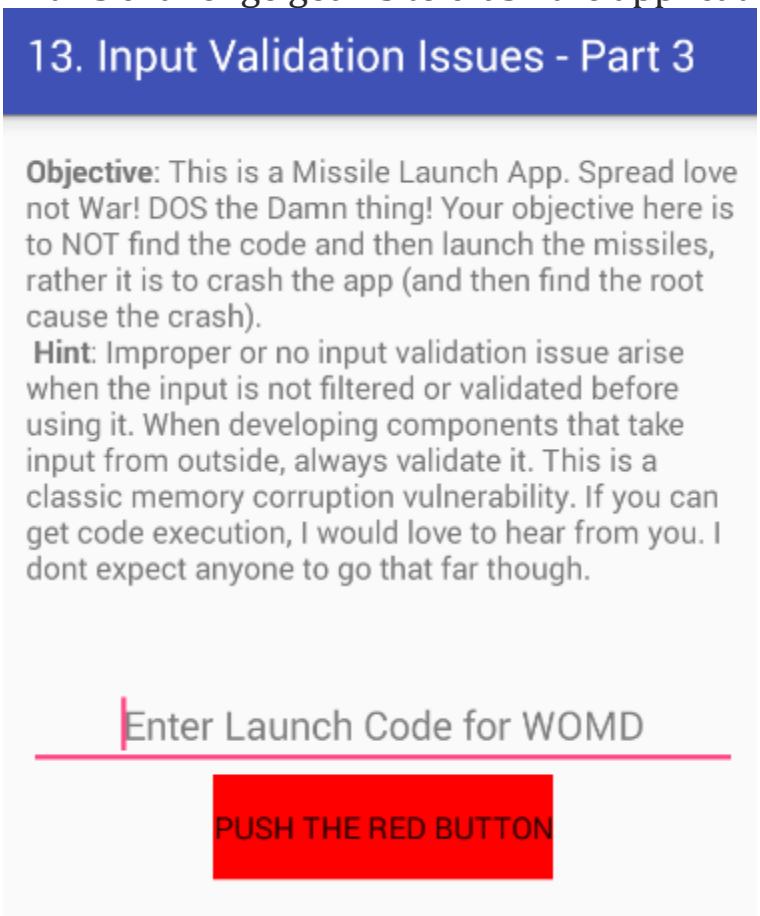


Figure 1.49

Enter any random but long string, that string will lead to crash the application.

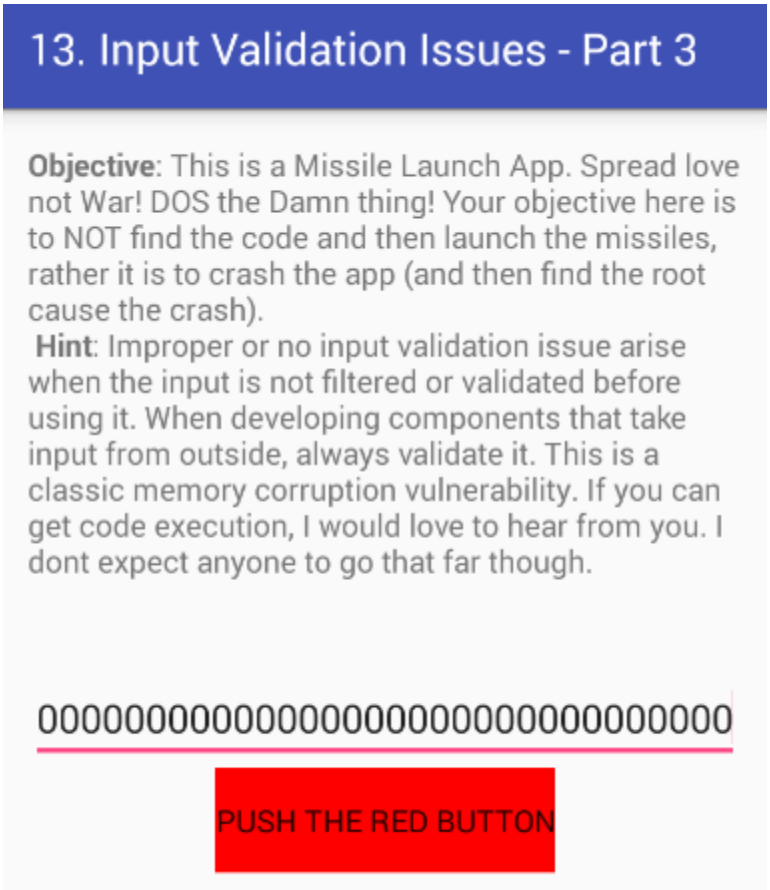

13. Input Validation Issues - Part 3
Objective: This is a Missile Launch App. Spread love not War! DOS the Damn thing! Your objective here is to NOT find the code and then launch the missiles, rather it is to crash the app (and then find the root cause the crash).
Hint: Improper or no input validation issue arise when the input is not filtered or validated before using it. When developing components that take input from outside, always validate it. This is a classic memory corruption vulnerability. If you can get code execution, I would love to hear from you. I dont expect anyone to go that far though.
0000000000000000000000000000000
PUSH THE RED BUTTON

Figure 1.50

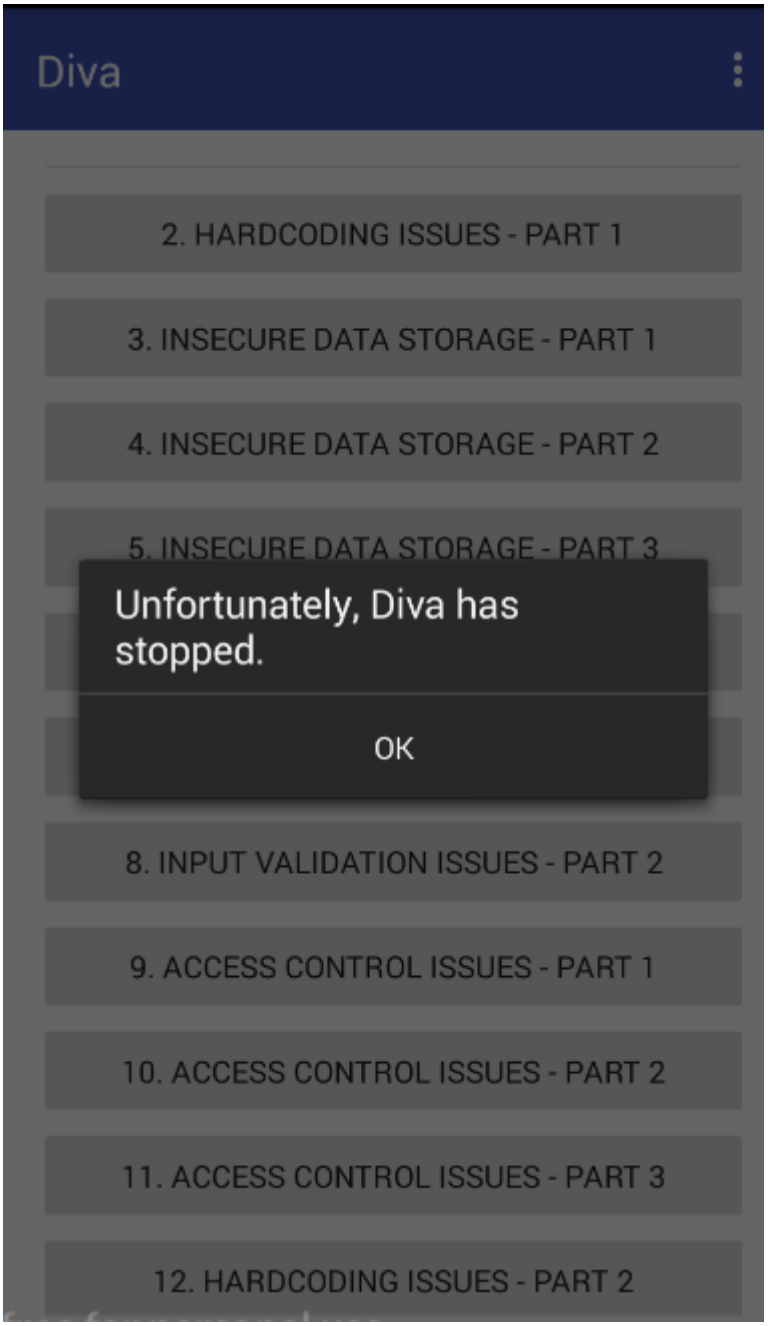


Figure 1.51

We have successfully crashed this application.

Here Input Validation Issues - Part 3 challenge is completed.

We have successfully cracked the full DIVA application and completed all challenges.