ABU DHABI POLYTECHNIC

INFORMATION SECURITY ENGINEERING TECHNOLOGY

# OCT- Secure Mobile.App Dev Lab 2

# DIVA Android App

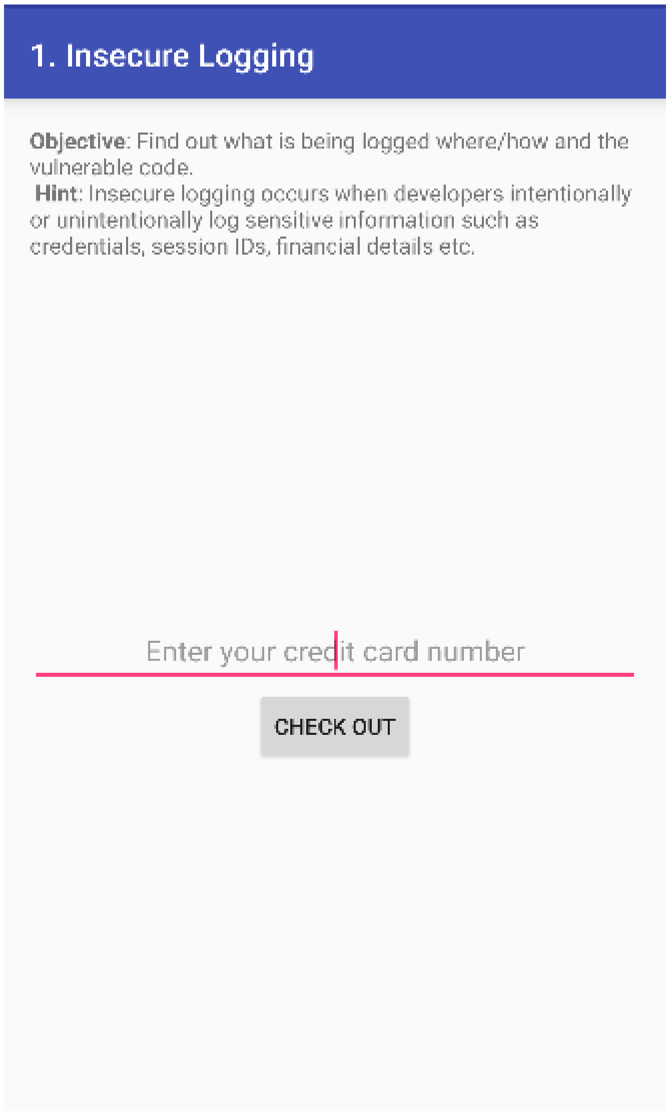
Prepared By:

Dua’a Abuhamdi

In this Lab article we are going to crack DIVA Android Application.

## INSECURE LOGGING:

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

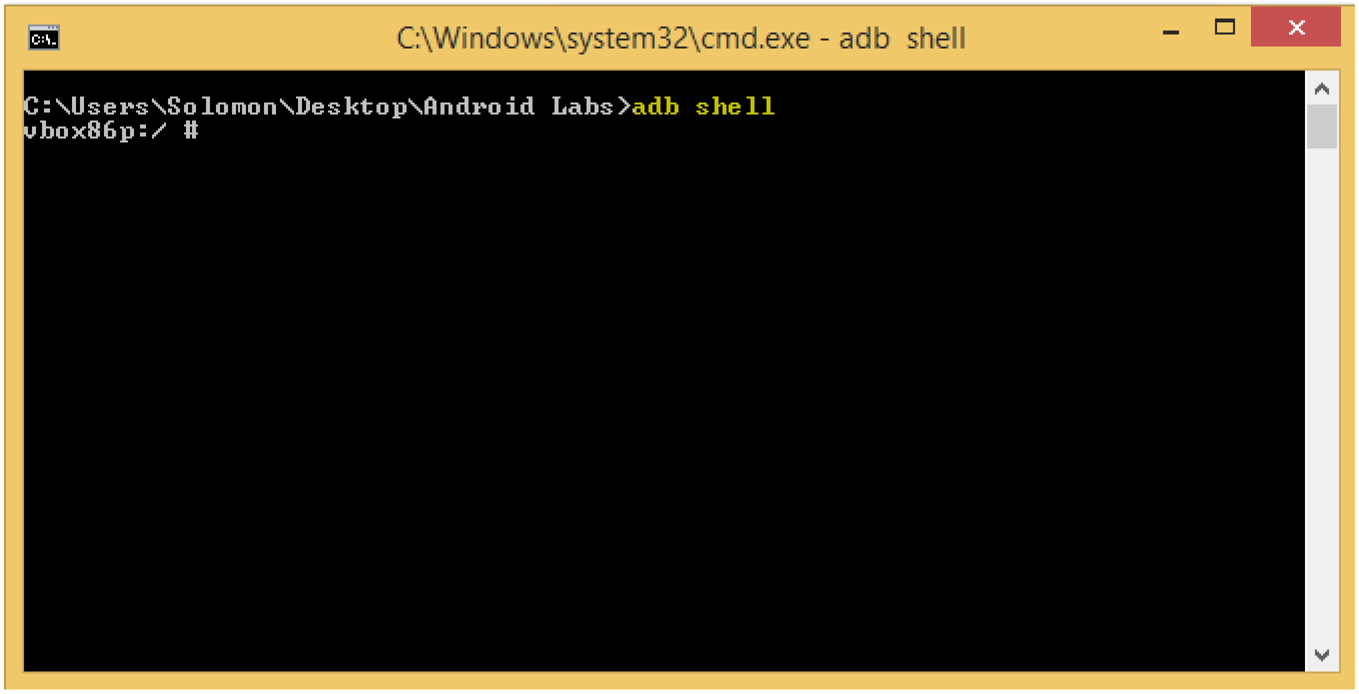

1. Insecure Logging

**Objective**: Find out what is being logged where/how and the vulnerable code.
**Hint**: Insecure logging occurs when developers intentionally or unintentionally log sensitive information such as credentials, session IDs, financial details etc.

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**



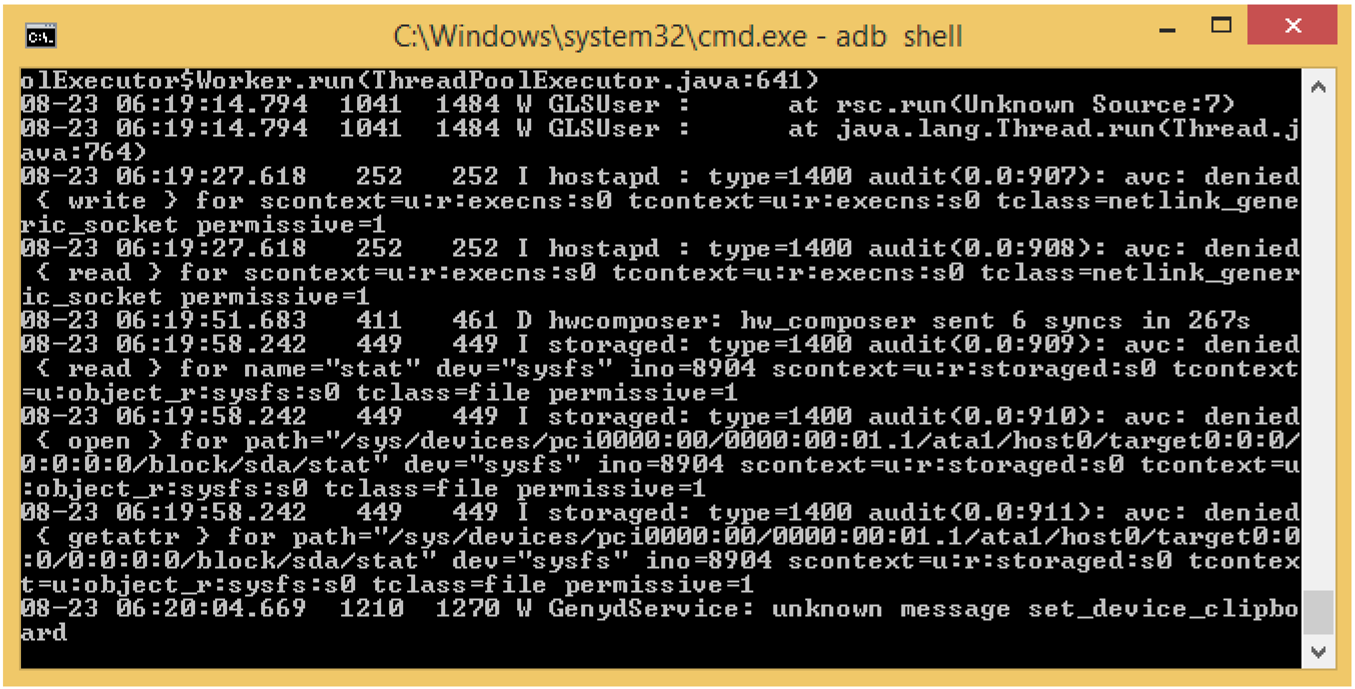
```
C:\Users\Solomon\Desktop\Android Labs>adb shell
vbox86p:/ #
```

Figure 1.8

Shell will open there type the command:

**logcat**

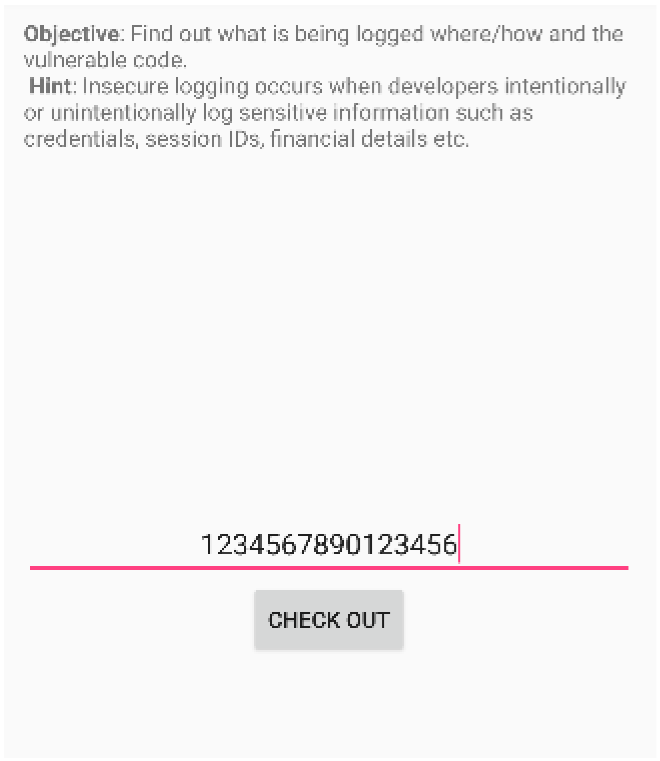
Once you enter command logs will start appearing in front of you.



```
olExecutor$Worker.run(ThreadPoolExecutor.java:641)
08-23 06:19:14.794  1041  1484 W GLSUser :       at rsc.run(Unknown Source:7)
08-23 06:19:14.794  1041  1484 W GLSUser :       at java.lang.Thread.run(Thread.j
ava:764)
08-23 06:19:27.618   252   252 I hostapd : type=1400 audit(0.0:907): avc: denied
 { write } for scontext=u:r:execns:s0 tcontext=u:r:execns:s0 tclass=netlink_gene
ric_socket permissive=1
08-23 06:19:27.618   252   252 I hostapd : type=1400 audit(0.0:908): avc: denied
 { read } for scontext=u:r:execns:s0 tcontext=u:r:execns:s0 tclass=netlink_gener
ic_socket permissive=1
08-23 06:19:51.683   411   461 D hwcomposer: hw_composer sent 6 syncs in 267s
08-23 06:19:58.242   449   449 I storaged: type=1400 audit(0.0:909): avc: denied
 { read } for name="stat" dev="sysfs" ino=8904 scontext=u:r:storaged:s0 tcontext
=u:object_r:sysfs:s0 tclass=file permissive=1
08-23 06:19:58.242   449   449 I storaged: type=1400 audit(0.0:910): avc: denied
 { open } for path="/sys/devices/pci0000:00/0000:00:01.1/ata1/host0/target0:0:0/
0:0:0:0/block/sda/stat" dev="sysfs" ino=8904 scontext=u:r:storaged:s0 tcontext=u
:object_r:sysfs:s0 tclass=file permissive=1
08-23 06:19:58.242   449   449 I storaged: type=1400 audit(0.0:911): avc: denied
 { getattr } for path="/sys/devices/pci0000:00/0000:00:01.1/ata1/host0/target0:0
:0/0:0:0:0/block/sda/stat" dev="sysfs" ino=8904 scontext=u:r:storaged:s0 tcontex
t=u:object_r:sysfs:s0 tclass=file permissive=1
08-23 06:20:04.669  1210  1270 W GenydService: unknown message set_device_clipbo
ard
```

Figure 1.9

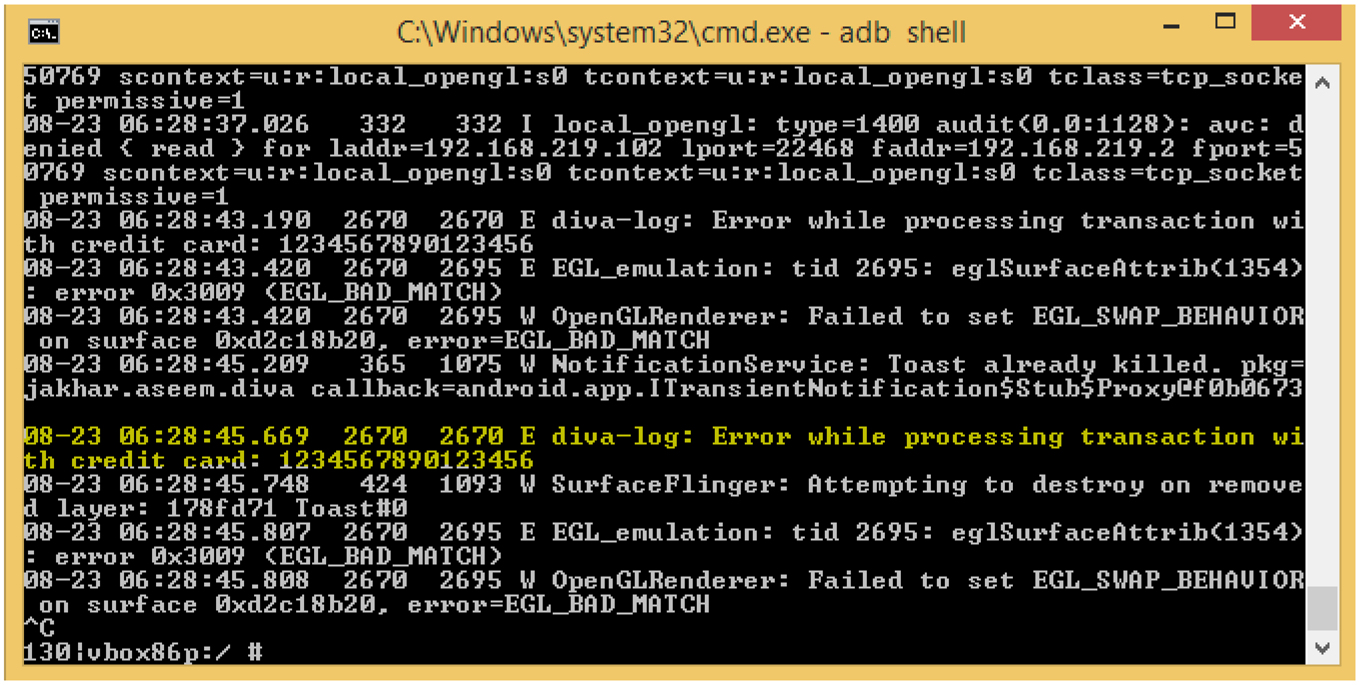
Now go to the Android VM and there enter credit card number


**Objective**: Find out what is being logged where/how and the vulnerable code.
**Hint**: Insecure logging occurs when developers intentionally or unintentionally log sensitive information such as credentials, session IDs, financial details etc.

1234567890123456

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:



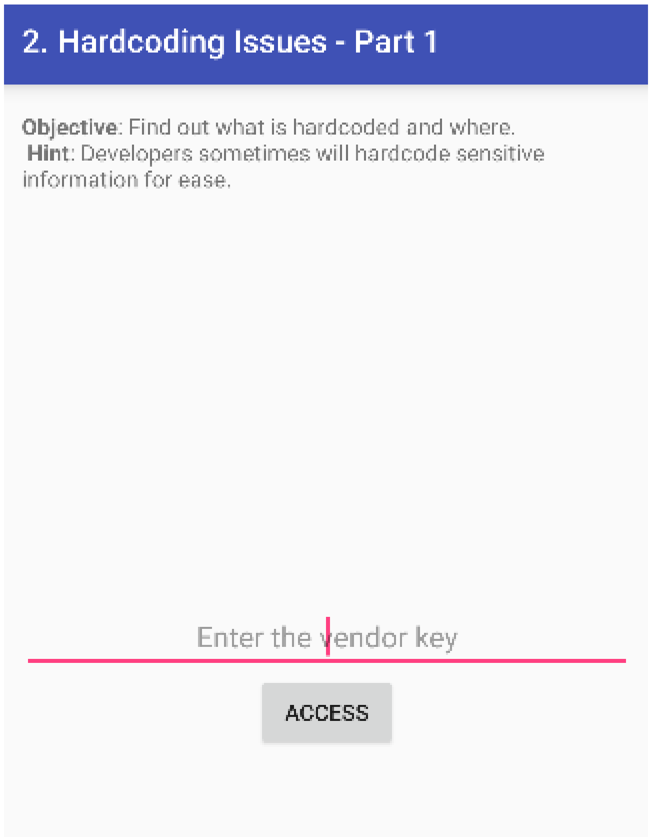
```
50769 scontext=u:r:local_opengl:s0 tcontext=u:r:local_opengl:s0 tclass=tcp_socke
t permissive=1
08-23 06:28:37.026   332   332 I local_opengl: type=1400 audit(0.0:1128): avc: d
enied { read } for laddr=192.168.219.102 lport=22468 faddr=192.168.219.2 fport=5
0769 scontext=u:r:local_opengl:s0 tcontext=u:r:local_opengl:s0 tclass=tcp_socket
 permissive=1
08-23 06:28:43.190  2670  2670 E diva-log: Error while processing transaction wi
th credit card: 1234567890123456
08-23 06:28:43.420  2670  2695 E EGL_emulation: tid 2695: eglSurfaceAttrib(1354)
: error 0x3009 (EGL_BAD_MATCH)
08-23 06:28:43.420  2670  2695 W OpenGLRenderer: Failed to set EGL_SWAP_BEHAVIOR
 on surface 0xd2c18b20, error=EGL_BAD_MATCH
08-23 06:28:45.209   365  1075 W NotificationService: Toast already killed. pkg=
jakhar.aseem.diva callback=android.app.ITransientNotification$Stub$Proxy@f0b0673

08-23 06:28:45.669  2670  2670 E diva-log: Error while processing transaction wi
th credit card: 1234567890123456
08-23 06:28:45.748   424  1093 W SurfaceFlinger: Attempting to destroy on remove
d layer: 178fd71 Toast#0
08-23 06:28:45.807  2670  2695 E EGL_emulation: tid 2695: eglSurfaceAttrib(1354)
: error 0x3009 (EGL_BAD_MATCH)
08-23 06:28:45.808  2670  2695 W OpenGLRenderer: Failed to set EGL_SWAP_BEHAVIOR
 on surface 0xd2c18b20, error=EGL_BAD_MATCH
^C
130|vbox86p:/ #
```

Figure 1.11

Here Insecure Logging challenge is completed.

## HARDCODING ISSUES - PART 1:

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


2. Hardcoding Issues - Part 1

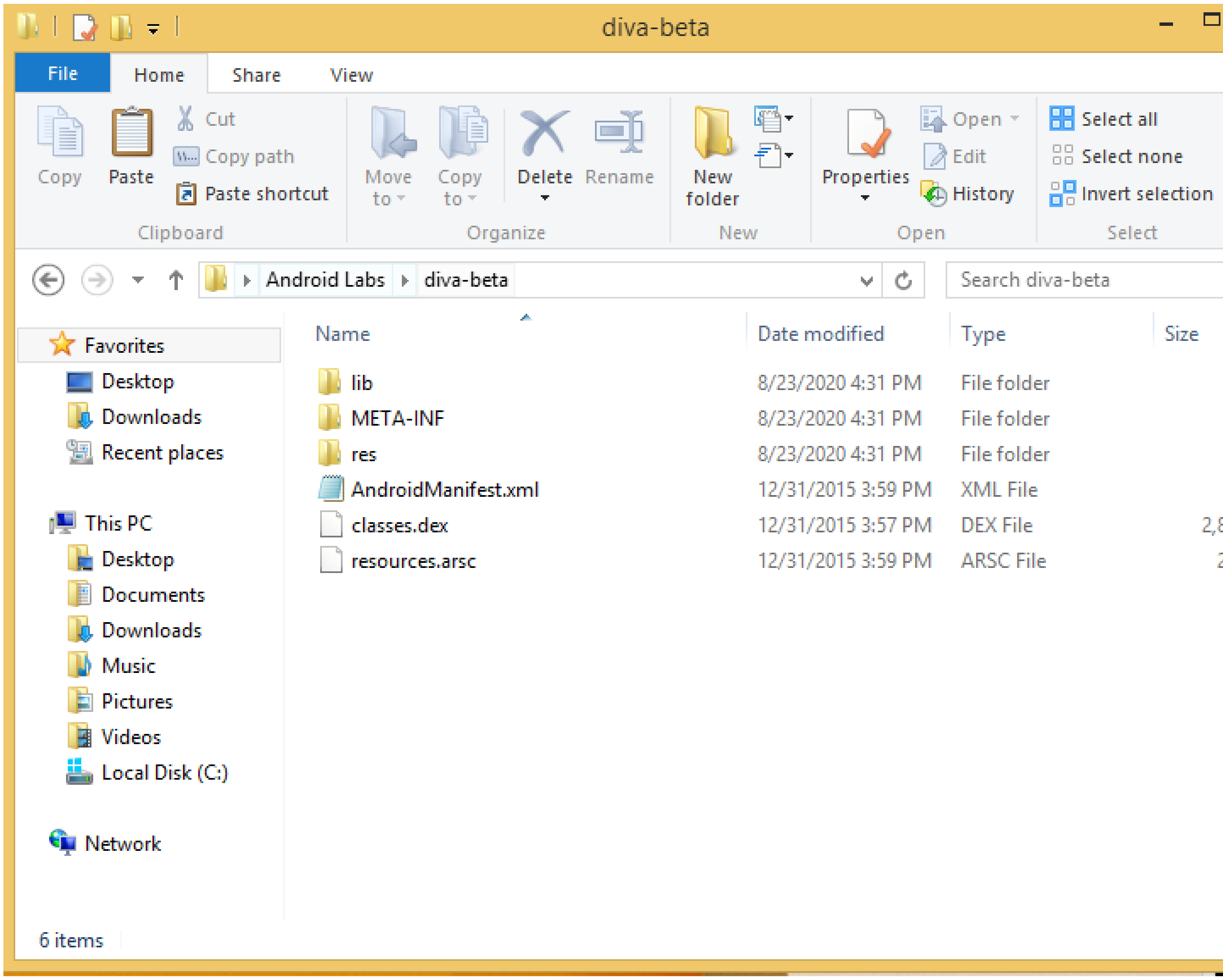
**Objective**: Find out what is hardcoded and where.
**Hint**: Developers sometimes will hardcode sensitive information for ease.

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.

First convert the APK file into RAR file by only changing the extension.

Then Extract the RAR file. You will get DEX file along with other files and folders as shown in figure 1.13 below:



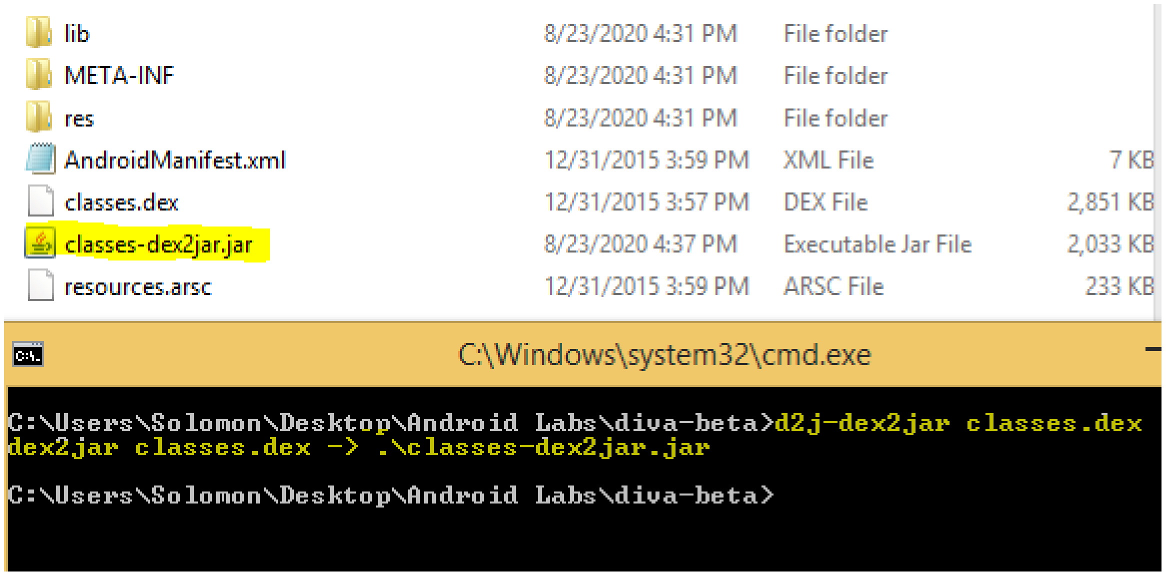
| Name | Date modified | Type | Size |
|---|---|---|---|
| lib | 8/23/2020 4:31 PM | File folder | |
| META-INF | 8/23/2020 4:31 PM | File folder | |
| res | 8/23/2020 4:31 PM | File folder | |
| AndroidManifest.xml | 12/31/2015 3:59 PM | XML File | |
| classes.dex | 12/31/2015 3:57 PM | DEX File | 2,8 |
| resources.arsc | 12/31/2015 3:59 PM | ARSC File | 2 |

Figure 1.13

Now convert this classes.dex file into jar file with Dex2Jar tool. Go to Command Prompt and enter following command:

**d2j-dex2jar classes.dex**

A converted jar file with same name as dex file will appear in your folder as shown in figure 1.14 below:

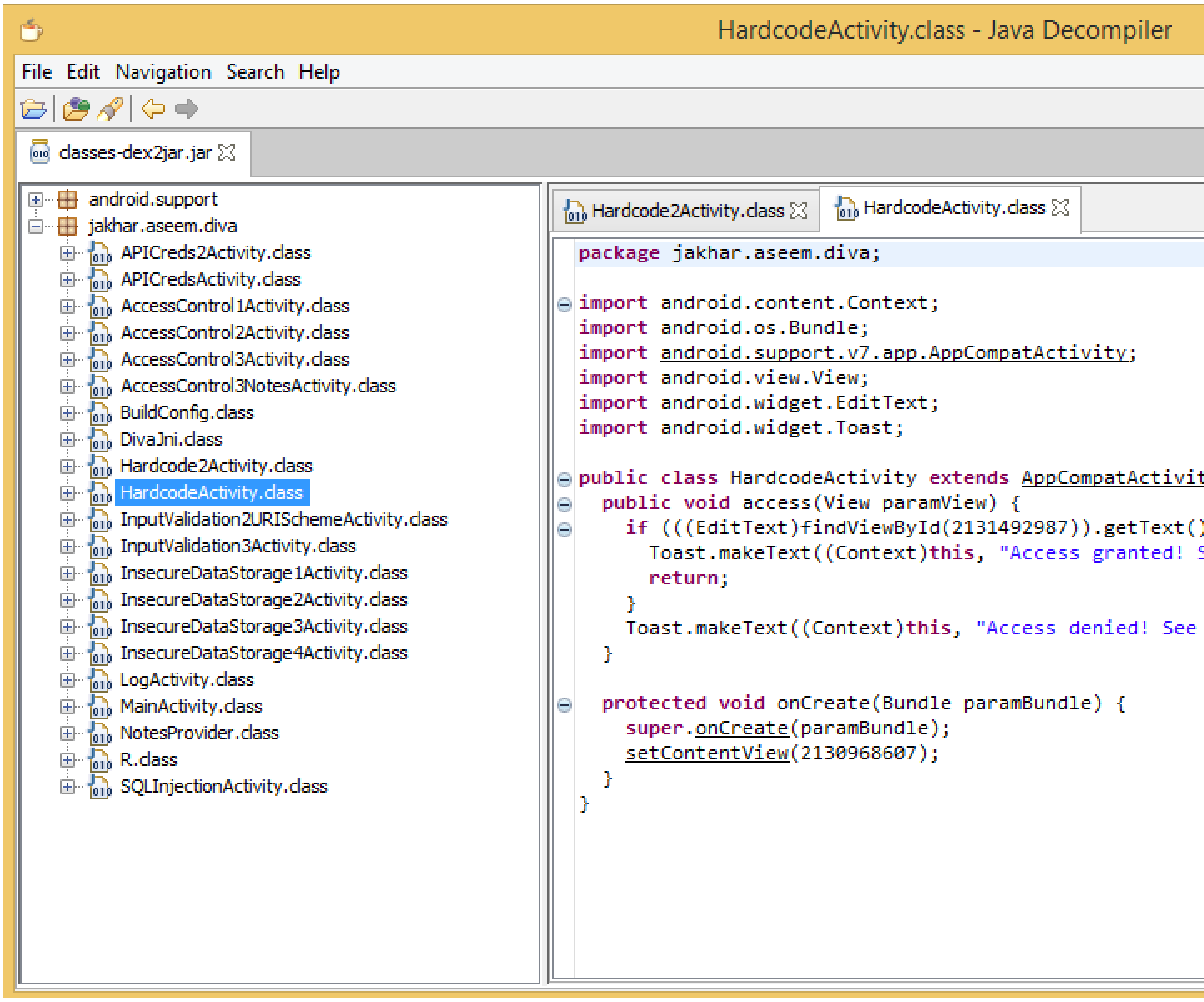


```
lib                         8/23/2020 4:31 PM    File folder
META-INF                    8/23/2020 4:31 PM    File folder
res                         8/23/2020 4:31 PM    File folder
AndroidManifest.xml         12/31/2015 3:59 PM   XML File                 7 KB
classes.dex                 12/31/2015 3:57 PM   DEX File             2,851 KB
classes-dex2jar.jar         8/23/2020 4:37 PM    Executable Jar File  2,033 KB
resources.arsc              12/31/2015 3:59 PM   ARSC File              233 KB
```

```
C:\Users\Solomon\Desktop\Android Labs\diva-beta>d2j-dex2jar classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar

C:\Users\Solomon\Desktop\Android Labs\diva-beta>
```

Figure 1.14

Now access this JAR file with JD-GUI which is Java Decompiler.

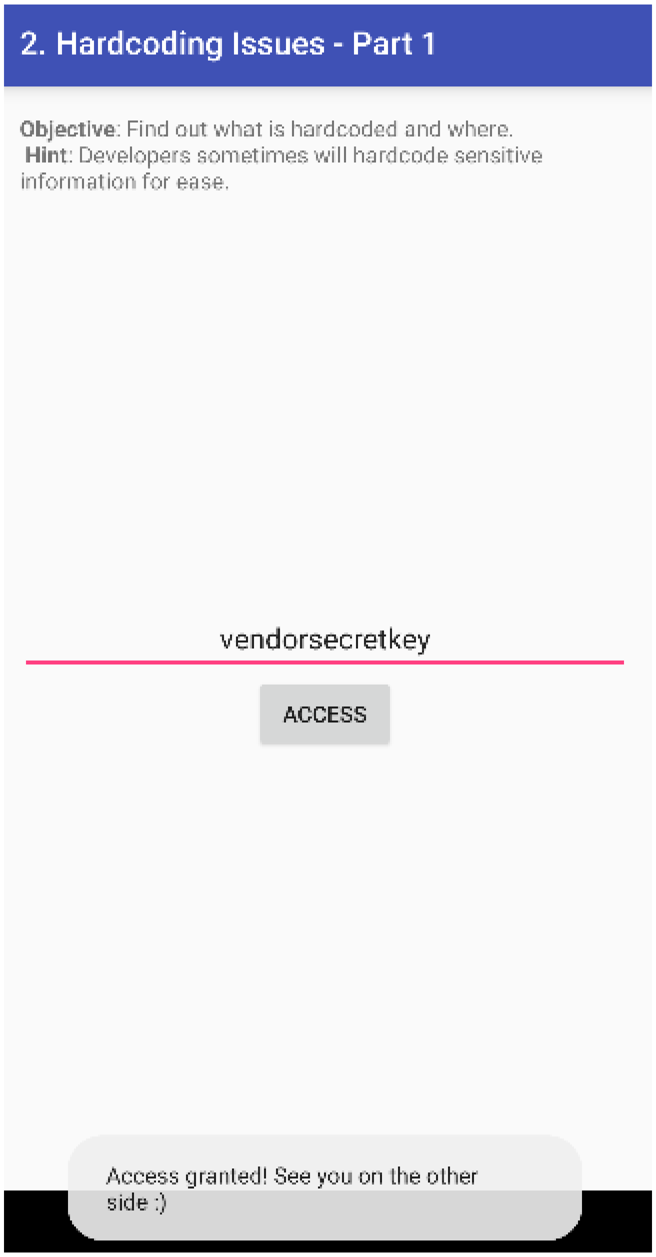

HardcodeActivity.class - Java Decompiler
File Edit Navigation Search Help
classes-dex2jar.jar
Hardcode2Activity.class
HardcodeActivity.class

```
android.support
jakhar.aseem.diva
    APICreds2Activity.class
    APICredsActivity.class
    AccessControl1Activity.class
    AccessControl2Activity.class
    AccessControl3Activity.class
    AccessControl3NotesActivity.class
    BuildConfig.class
    DivaJni.class
    Hardcode2Activity.class
    HardcodeActivity.class
    InputValidation2URISchemeActivity.class
    InputValidation3Activity.class
    InsecureDataStorage1Activity.class
    InsecureDataStorage2Activity.class
    InsecureDataStorage3Activity.class
    InsecureDataStorage4Activity.class
    LogActivity.class
    MainActivity.class
    NotesProvider.class
    R.class
    SQLInjectionActivity.class
```

```java
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.EditText;
import android.widget.Toast;

public class HardcodeActivity extends AppCompatActivit
  public void access(View paramView) {
    if (((EditText)findViewById(2131492987)).getText()
      Toast.makeText((Context)this, "Access granted! S
      return;
    }
    Toast.makeText((Context)this, "Access denied! See
  }

  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130968607);
  }
}
```

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:


2. Hardcoding Issues - Part 1

**Objective**: Find out what is hardcoded and where.
**Hint**: Developers sometimes will hardcode sensitive information for ease.

vendorsecretkey

Access granted! See you on the other side :)

Figure 1.16

Here Hardcoding Issue - Part 1 challenge is completed.