Dynamic Android Pentesting

(By: Jaivardhan Khanna)

This is a dynamic application analysis for 'DIVA', which is a deliberately insecure Android application designed for educational and testing purposes.
Throughout this analysis, I went through a number of vulnerabilities, such as:

1. *Insecure Logging*
2. *Hardcoding Issues*
3. *Insecure Data Storage*
4. *Input Validation Issues*
5. *Access Control Issues*

Even though the application itself is pretty old and made deliberately vulnerable, it covers and showcases some of the most common made mistakes during app development and packaging. This allows developers, security professionals, and students to practice identifying and exploiting these issues to enhance their mobile application security skills.
Using a variety of tools for this case analysis, my toolkit included:

1. *Genymotion*
   (an Android emulator for Windows, Mac and Linux).
2. *Oracle Virtual Box*
   (a tool for virtualizing x86 and AMD64/Intel64 computing architecture).
3. *SDK Platform Tools*
   (include tools that interface with the Android platform, primarily adb and fastboot).
4. *Jadx-GUI*
   (a GUI for decompiling a .apk file into its respective source code).
5. *SQLite Viewer*
   (to view the embedded data in SQL queries that was being stored in the application).

Initial Setup

The steps for the setting up of Android machine, viewing the source code and connecting the android machine to windows terminal are as follow:

1. *Download all the required applications from the links provided.*
2. *Set-Up Genymotion:*
   1. *You will need to Create and Account on genymotion in order to use the app.*
   2. *After logging in the app, while creating your 'android device', you will be given options to select the model of your Android Device, the Android Version you want to use and the memory you want to assign to the device.*
   3. *Now, you just need to press the ▶ Button to start the device.*
   4. *Now, drag and drop the 'DIVA' apk file into your device and it will be installed as a app.*
3. *Using SDK tools:*
   1. *Download the 'platform-tools' zip file and extract it.*
   2. *Right Click on the 'platform-tools' folder and open it in Terminal.*
4. *Connecting Android Device to Terminal:*
   1. *Look at the top of the Genymotion box and you will find an IP address.*
   2. *In the windows terminal**, type the command*

      adb connect <ip-address>
   3. *Now verify if you have connected to the device using*

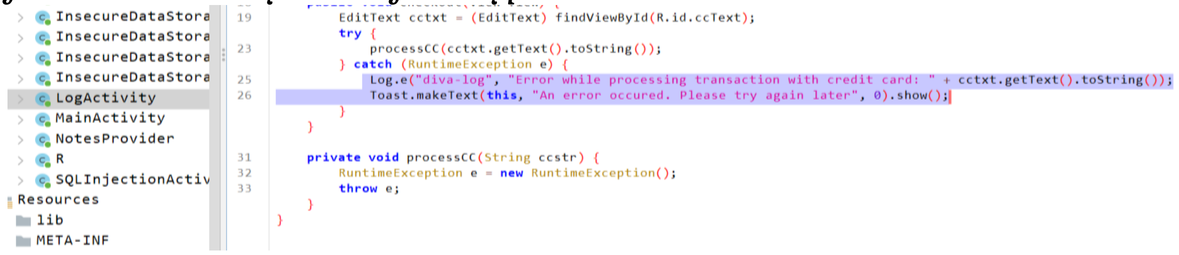
      adb devices
5. *Viewing the source code:*
   1. *Open the 'jadx-gui' folder and click on ' jadx-gui.exe '.*
   2. *After the application opens, click on Open File and select the 'DIVA' apk file and you will be able to see all of the components of the code of the apk file.*

Now our environment is all set, so let's explore all the vulnerabilities.

***this specific windows terminal will be referred to as shell.*

I. Insecure Login

The goal is to find where the user data is being stored.
On code inspection, we find that the data is being stored in the log files without any kind of encryption.

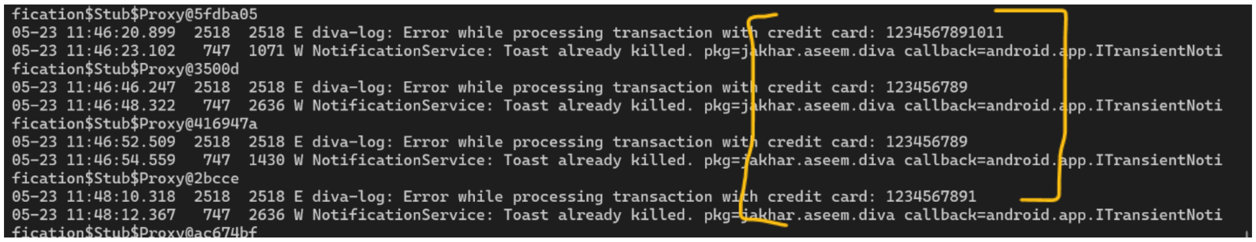


```
InsecureDataStora
InsecureDataStora
InsecureDataStora
InsecureDataStora
LogActivity
MainActivity
NotesProvider
R
SQLInjectionActiv
Resources
lib
META-INF
```

```java
        EditText cctxt = (EditText) findViewById(R.id.ccText);
        try {
            processCC(cctxt.getText().toString());
        } catch (RuntimeException e) {
            Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toString());
            Toast.makeText(this, "An error occured. Please try again later", 0).show();
        }
    }

    private void processCC(String ccstr) {
        RuntimeException e = new RuntimeException();
        throw e;
    }
}
```

So, to inspect it, we go to the terminal and type

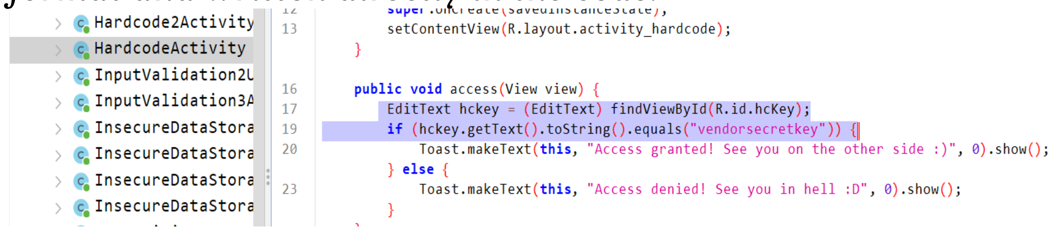
adb shell logcat | grep diva



```
fication$Stub$Proxy@5fdba05
05-23 11:46:20.899  2518  2518 E diva-log: Error while processing transaction with credit card: 1234567891011
05-23 11:46:23.102   747  1071 W NotificationService: Toast already killed. pkg=jakhar.aseem.diva callback=android.app.ITransientNoti
fication$Stub$Proxy@3500d
05-23 11:46:46.247  2518  2518 E diva-log: Error while processing transaction with credit card: 123456789
05-23 11:46:48.322   747  2636 W NotificationService: Toast already killed. pkg=jakhar.aseem.diva callback=android.app.ITransientNoti
fication$Stub$Proxy@416947a
05-23 11:46:52.509  2518  2518 E diva-log: Error while processing transaction with credit card: 123456789
05-23 11:46:54.559   747  1430 W NotificationService: Toast already killed. pkg=jakhar.aseem.diva callback=android.app.ITransientNoti
fication$Stub$Proxy@2bcce
05-23 11:48:10.318  2518  2518 E diva-log: Error while processing transaction with credit card: 1234567891
05-23 11:48:12.367   747  2636 W NotificationService: Toast already killed. pkg=jakhar.aseem.diva callback=android.app.ITransientNoti
fication$Stub$Proxy@ac674bf
```

II. Hardcoded Issue 1

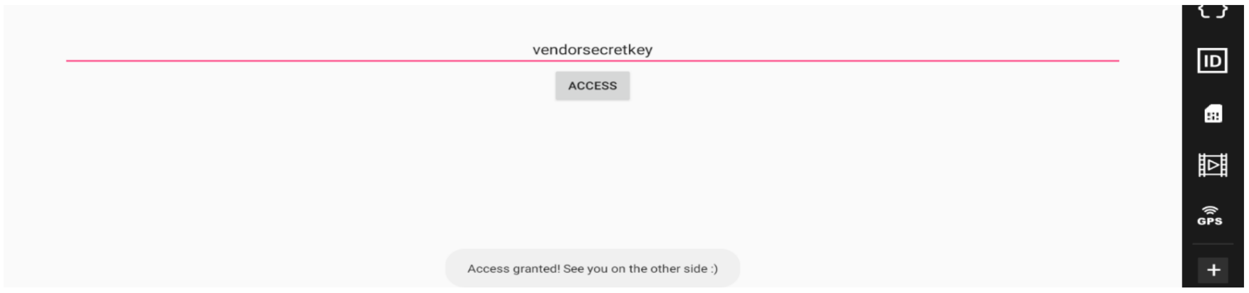
The goal is to find the directly embedded value in the source code that allows authentication.
Upon code inspection, we find that secret key is given in plain text format and written directly in the code.



```
Hardcode2Activity
HardcodeActivity
InputValidation2U
InputValidation3A
InsecureDataStora
InsecureDataStora
InsecureDataStora
InsecureDataStora
```

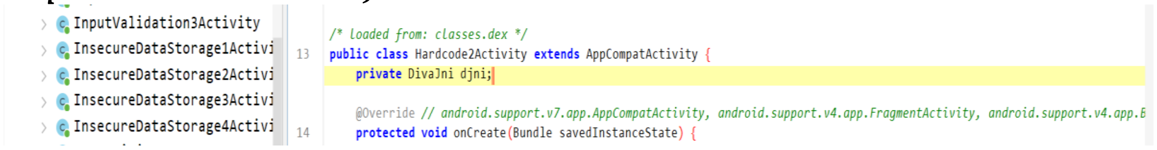
```java
        setContentView(R.layout.activity_hardcode);
    }

    public void access(View view) {
        EditText hckey = (EditText) findViewById(R.id.hcKey);
        if (hckey.getText().toString().equals("vendorsecretkey")) {
            Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
        } else {
            Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
        }
```

Therefore, when we enter "vendorsecretkey" as the input, we are granted access.



III. Hardcoded Issue 2

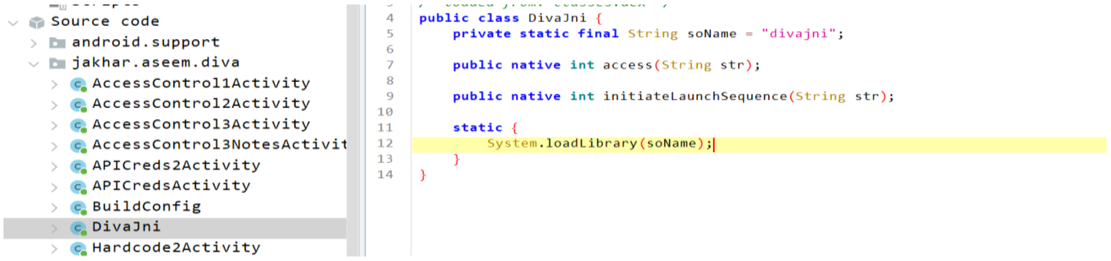
The goal is to find the directly embedded value in the source code that allows authentication.
Upon code inspection of this part, I noticed that there is a class being imported called "DivaJni".



```
> InputValidation3Activity
> InsecureDataStorage1Activi
> InsecureDataStorage2Activi
> InsecureDataStorage3Activi
> InsecureDataStorage4Activi
```

```java
/* loaded from: classes.dex */
public class Hardcode2Activity extends AppCompatActivity {
    private DivaJni djni;

    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.B
    protected void onCreate(Bundle savedInstanceState) {
```

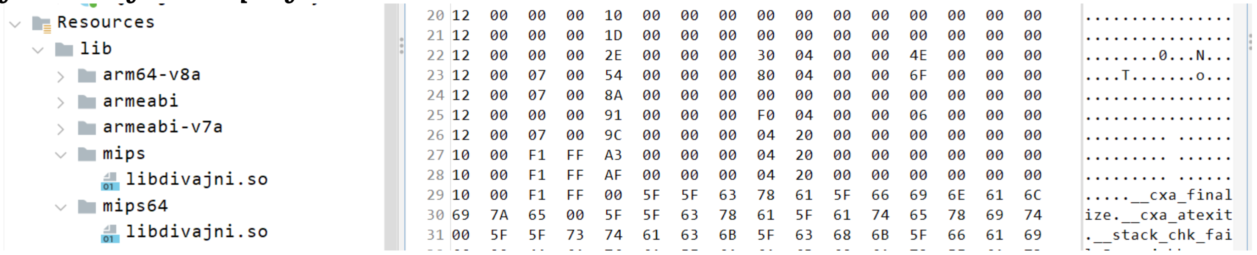
Now, if we see the "DivaJni" class, we find that another class's value is being called and used as the output of the "DivaJni" class.



```
Source code
  > android.support
  v jakhar.aseem.diva
    > AccessControl1Activity
    > AccessControl2Activity
    > AccessControl3Activity
    > AccessControl3NotesActivit
    > APICreds2Activity
    > APICredsActivity
    > BuildConfig
    > DivaJni
    > Hardcode2Activity
```

```java
public class DivaJni {
    private static final String soName = "divajni";

    public native int access(String str);

    public native int initiateLaunchSequence(String str);

    static {
        System.loadLibrary(soName);
    }
}
```

Since this is a library file, our first place to look for should be the "lib" folder of our apk file.



```
Resources
  v lib
    > arm64-v8a
    > armeabi
    > armeabi-v7a
    v mips
        libdivajni.so
    v mips64
        libdivajni.so
```

```
20 12  00  00  00  10  00  00  00  00  00  00  00  00  00  00  00   ................
21 12  00  00  00  1D  00  00  00  00  00  00  00  00  00  00  00   ................
22 12  00  00  00  2E  00  00  00  30  04  00  00  4E  00  00  00   ........0...N...
23 12  00  07  00  54  00  00  00  80  04  00  00  6F  00  00  00   ....T.......o...
24 12  00  07  00  8A  00  00  00  00  00  00  00  00  00  00  00   ................
25 12  00  00  00  91  00  00  00  F0  04  00  00  06  00  00  00   ................
26 12  00  07  00  9C  00  00  00  04  20  00  00  00  00  00  00   .........  ......
27 10  00  F1  FF  A3  00  00  00  04  20  00  00  00  00  00  00   .........  ......
28 10  00  F1  FF  AF  00  00  00  04  20  00  00  00  00  00  00   .........  ......
29 10  00  F1  FF  00  5F  5F  63  78  61  5F  66  69  6E  61  6C   .....__cxa_final
30 69  7A  65  00  5F  5F  63  78  61  5F  61  74  65  78  69  74   ize.__cxa_atexit
31 00  5F  5F  73  74  61  63  6B  5F  63  68  6B  5F  66  61  69   .__stack_chk_fai
```

In order to read these files, the steps are:

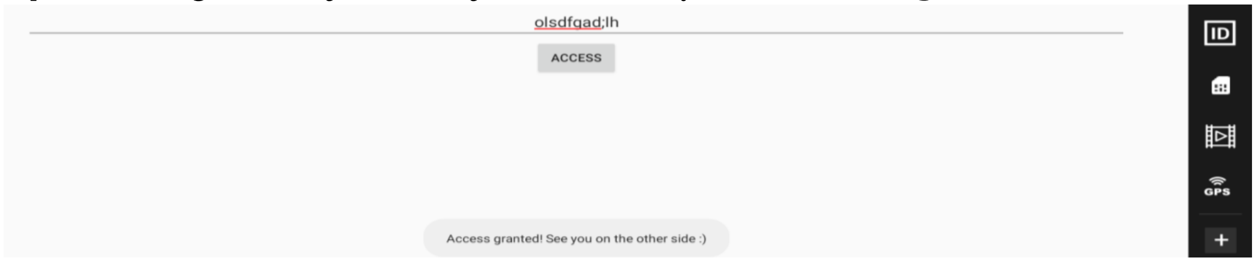
i. *Right click on "libdivajni.so" and select 'Export' to save the file on your desktop.*

ii. *Now open a new terminal on the desktop and use command.*

```
strings libdivajni.so
```

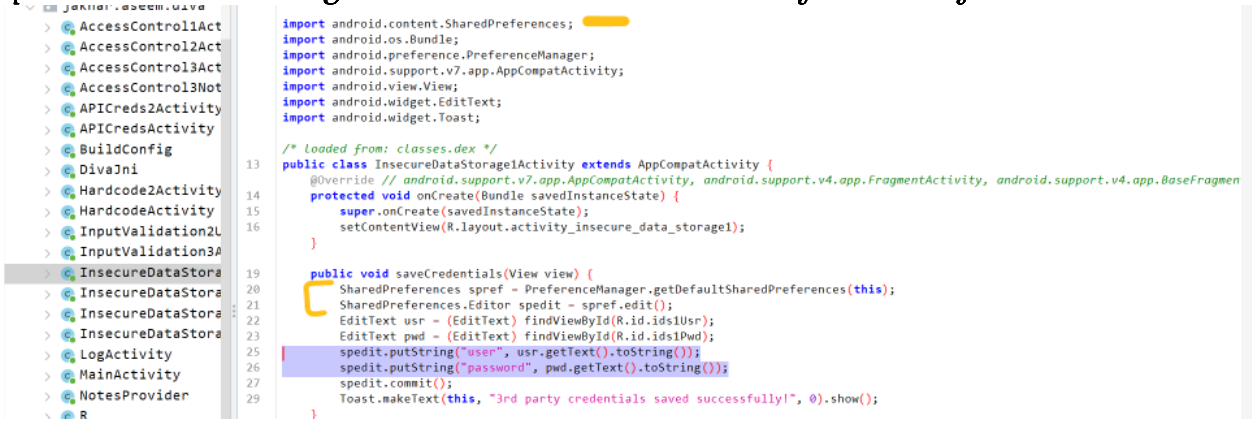
iii. *This will give us a list of strings from the file.*

Upon testing each of them, I found the key to be: olsdfgad;lh



IV. Insecure data Storage 1

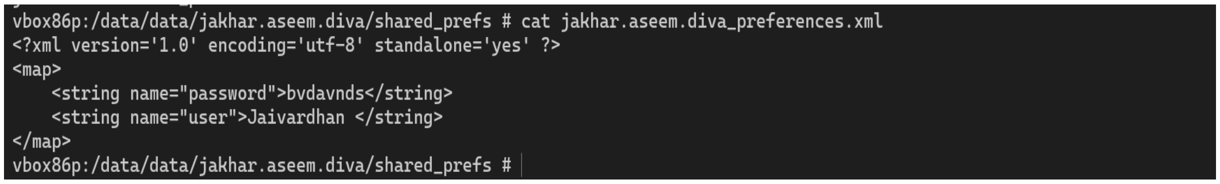
The goal is to find where/how the credentials are being stored.
Upon code inspection of this part, I noticed that the 'username' and 'password' are being stored as in the "SharedPreferences" folder.



```
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* loaded from: classes.dex */
public class InsecureDataStorage1Activity extends AppCompatActivity {
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmen
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage1);
    }

    public void saveCredentials(View view) {
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor spedit = spref.edit();
        EditText usr = (EditText) findViewById(R.id.ids1Usr);
        EditText pwd = (EditText) findViewById(R.id.ids1Pwd);
        spedit.putString("user", usr.getText().toString());
        spedit.putString("password", pwd.getText().toString());
        spedit.commit();
        Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    }
```

So, I navigated to the 'DIVA' application folder in my android device :

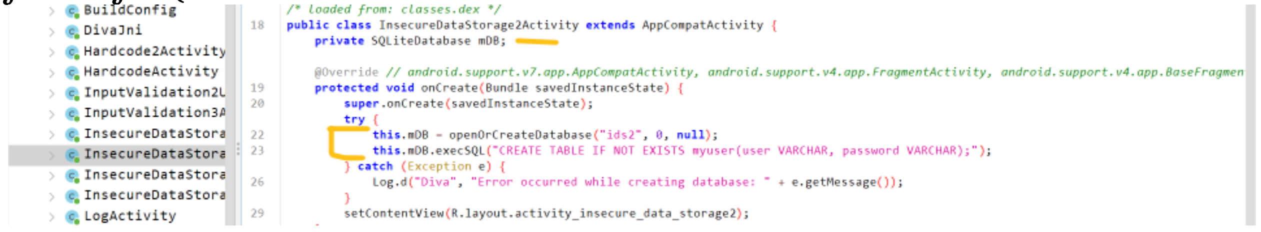
i. *In the shell, I typed*: cd /data/data
ii. *I then type* 'ls' *to check what folders are there.*
iii. *Then, I moved to "jakhar.aseem.diva" directory and searched for folders using* 'ls'. *I could see a folder named "shared_prefs" which had a file named "jakhar.aseem.diva_preferences.xml".*



```
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs # cat jakhar.aseem.diva_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">bvdavnds</string>
    <string name="user">Jaivardhan </string>
</map>
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs #
```

V. Insecure Data storage 2

The goal is to find where/how the credentials are being stored.
Upon code inspection of this part, I noticed data being stored in the form of 'SQL'.

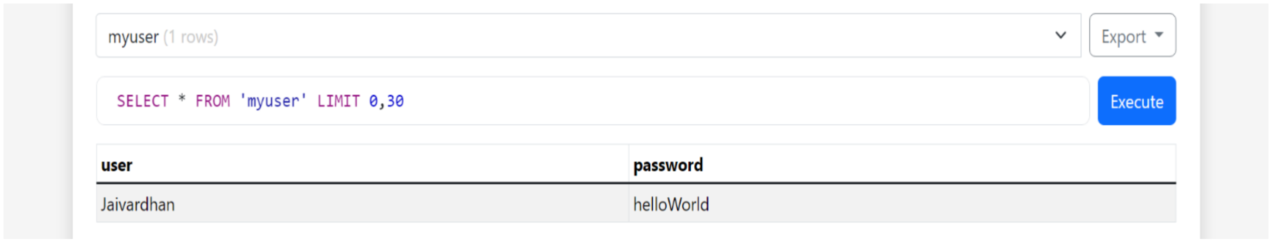


```
/* loaded from: classes.dex */
public class InsecureDataStorage2Activity extends AppCompatActivity {
    private SQLiteDatabase mDB;

    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmen
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try {
            this.mDB = openOrCreateDatabase("ids2", 0, null);
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
        } catch (Exception e) {
            Log.d("Diva", "Error occurred while creating database: " + e.getMessage());
        }
        setContentView(R.layout.activity_insecure_data_storage2);
```

So, I navigated to the 'DIVA' application folder in my android device using the same steps as mentioned in ([IV]). I noticed a folder named "databases" which had files with '.db' extension. So, I downloaded the whole folder using the command:

adb pull /data/data/Jakhar.aseem.diva/databases

After this, I used the SQLite Viewer to view each file and I found:

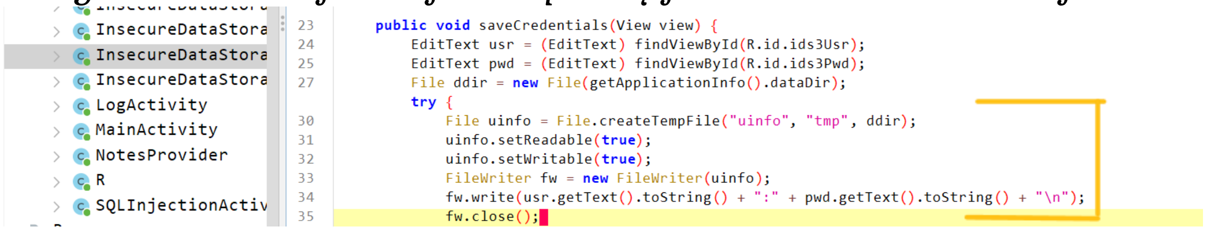


```
SELECT * FROM 'myuser' LIMIT 0,30
```

user	password
Jaivardhan	helloWorld

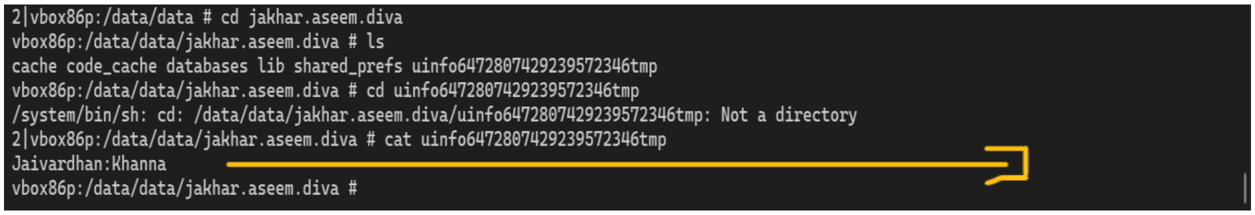
VI. Insecure Data Storage 3

The goal is to find where/how the credentials are being stored. Upon code inspection of this part, I noticed that the credentials are being stored in the form of a temporary file with the initials "uinfo".



```java
    public void saveCredentials(View view) {
        EditText usr = (EditText) findViewById(R.id.ids3Usr);
        EditText pwd = (EditText) findViewById(R.id.ids3Pwd);
        File ddir = new File(getApplicationInfo().dataDir);
        try {
            File uinfo = File.createTempFile("uinfo", "tmp", ddir);
            uinfo.setReadable(true);
            uinfo.setWritable(true);
            FileWriter fw = new FileWriter(uinfo);
            fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
            fw.close();
```

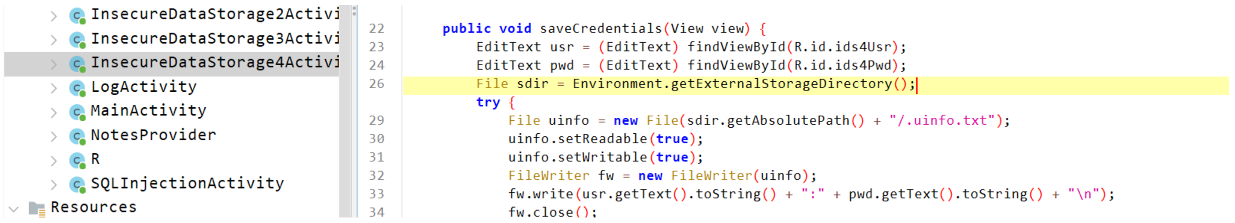
So, I navigated to the 'DIVA' application folder in my android device using the same steps as mentioned in ([IV]). I noticed a file with the initials "uinfo" and upon looking at its contents:



```
2|vbox86p:/data/data # cd jakhar.aseem.diva
vbox86p:/data/data/jakhar.aseem.diva # ls
cache code_cache databases lib shared_prefs uinfo6472807429239572346tmp
vbox86p:/data/data/jakhar.aseem.diva # cd uinfo6472807429239572346tmp
/system/bin/sh: cd: /data/data/jakhar.aseem.diva/uinfo6472807429239572346tmp: Not a directory
2|vbox86p:/data/data/jakhar.aseem.diva # cat uinfo6472807429239572346tmp
Jaivardhan:Khanna
vbox86p:/data/data/jakhar.aseem.diva #
```

VII. Insecure Data Storage 4

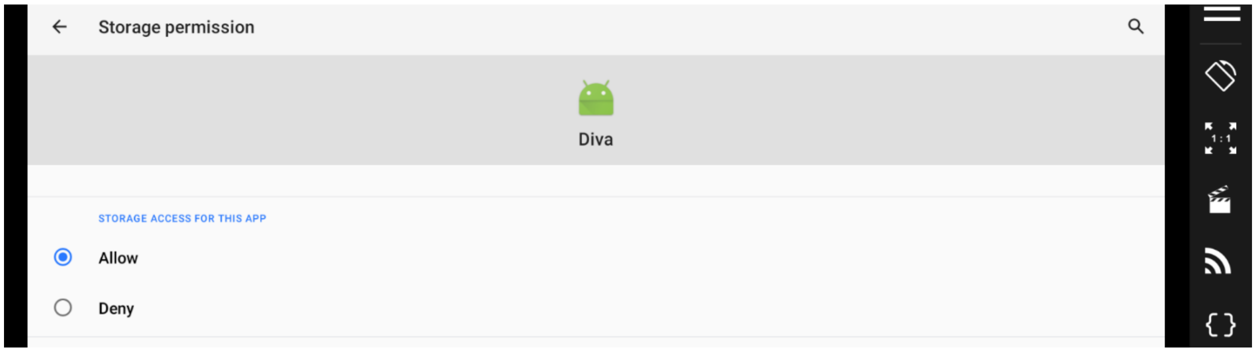
The goal is to find where/how the credentials are being stored.
Upon code inspection of this part, I noticed that the data was being stored in an external SDCard.



```
InsecureDataStorage2Activi
InsecureDataStorage3Activi
InsecureDataStorage4Activi
LogActivity
MainActivity
NotesProvider
R
SQLInjectionActivity
Resources
```

```java
    public void saveCredentials(View view) {
        EditText usr = (EditText) findViewById(R.id.ids4Usr);
        EditText pwd = (EditText) findViewById(R.id.ids4Pwd);
        File sdir = Environment.getExternalStorageDirectory();
        try {
            File uinfo = new File(sdir.getAbsolutePath() + "/.uinfo.txt");
            uinfo.setReadable(true);
            uinfo.setWritable(true);
            FileWriter fw = new FileWriter(uinfo);
            fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
            fw.close();
```

However, in my case, the application was not given permission to access external storage. So, in order to check out this vulnerability, I had to give permission to the 'DIVA' app from settings.



Storage permission

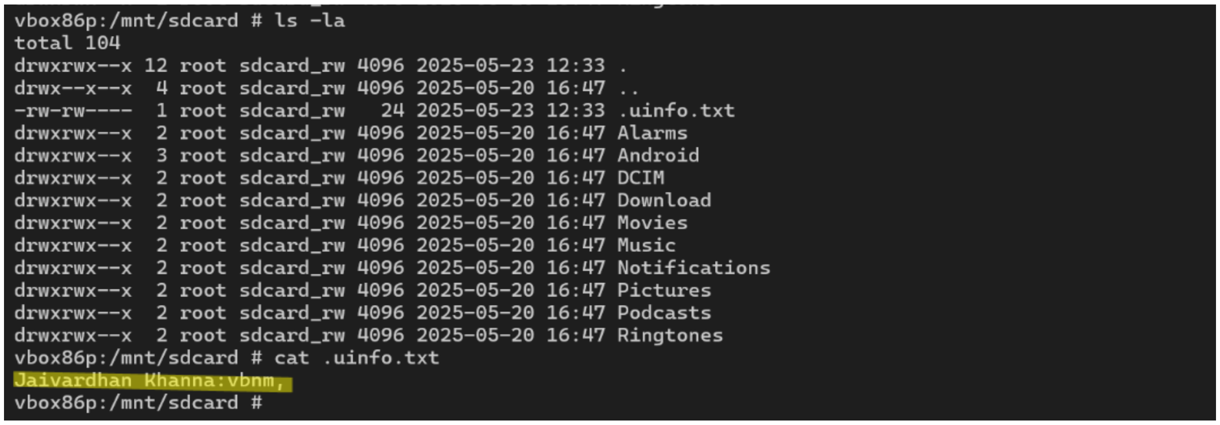
Diva

STORAGE ACCESS FOR THIS APP

[✓ Allow]

Deny

After giving the permission, the user credentials were being stored without error. So, from the shell, I opened the "mnt" directory inside of which was the "sdcard" directory.
What the catch here was, that a 'ls' command would only show visible directories, so, I ran the 'ls -la' and found:

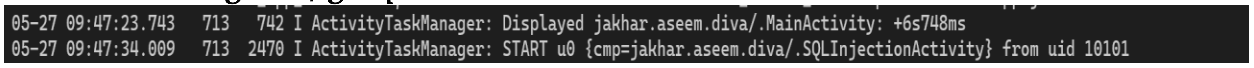


```
vbox86p:/mnt/sdcard # ls -la
total 104
drwxrwx--x 12 root sdcard_rw 4096 2025-05-23 12:33 .
drwx--x--x  4 root sdcard_rw 4096 2025-05-20 16:47 ..
-rw-rw----  1 root sdcard_rw   24 2025-05-23 12:33 .uinfo.txt
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Alarms
drwxrwx--x  3 root sdcard_rw 4096 2025-05-20 16:47 Android
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 DCIM
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Download
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Movies
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Music
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Notifications
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Pictures
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Podcasts
drwxrwx--x  2 root sdcard_rw 4096 2025-05-20 16:47 Ringtones
vbox86p:/mnt/sdcard # cat .uinfo.txt
Jaivardhan Khanna:vbnm,
vbox86p:/mnt/sdcard #
```

VIII. Input Data Validation Issue 1

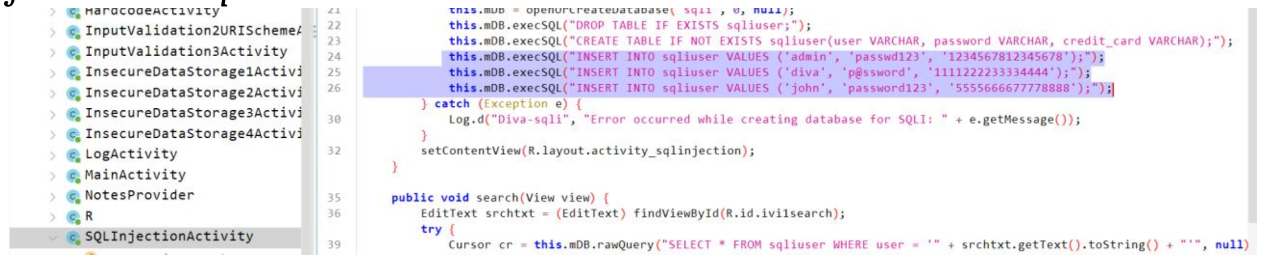
The goal is to find all user data without knowing any usernames. Upon inspection of the source code, I found that this time, there wasn't a section named as "InputValidation1" or something, so to identify my issue, I checked the log of the app using logcat in the shell.

logcat | grep diva



```
05-27 09:47:23.743   713   742 I ActivityTaskManager: Displayed jakhar.aseem.diva/.MainActivity: +6s748ms
05-27 09:47:34.009   713  2470 I ActivityTaskManager: START u0 {cmp=jakhar.aseem.diva/.SQLInjectionActivity} from uid 10101
```

This shows that when an input is being entered, the application calls for the file named "SQLInjectionActivity", upon looking at the file, I found the required user credentials.

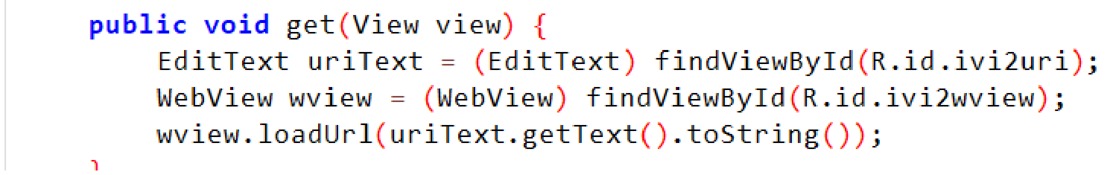


```
HardcodeActivity
InputValidation2URISchemeA
InputValidation3Activity
InsecureDataStorage1Activi
InsecureDataStorage2Activi
InsecureDataStorage3Activi
InsecureDataStorage4Activi
LogActivity
MainActivity
NotesProvider
R
SQLInjectionActivity
```

```
            this.mDB = openOrCreateDatabase("sqli", 0, null);
            this.mDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'passwd123', '1234567812345678');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('diva', 'p@ssword', '1111222233334444');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('john', 'password123', '5555666677778888');");
        } catch (Exception e) {
            Log.d("Diva-sqli", "Error occurred while creating database for SQLI: " + e.getMessage());
        }
        setContentView(R.layout.activity_sqlinjection);
    }

    public void search(View view) {
        EditText srchtxt = (EditText) findViewById(R.id.ivi1search);
        try {
            Cursor cr = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null)
```

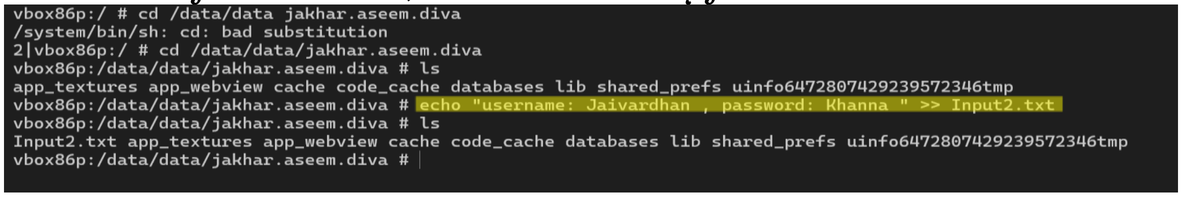
IX. Input Data Validation Issue 2

The goal is to access any sensitive information apart from a web URL. Upon looking at the source code, I could tell that the input section will take a parameter without validating it or checking any system restrictions.



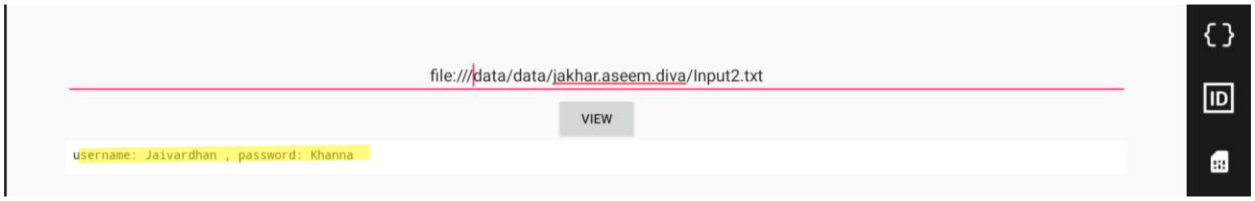
```
    public void get(View view) {
        EditText uriText = (EditText) findViewById(R.id.ivi2uri);
        WebView wview = (WebView) findViewById(R.id.ivi2wview);
        wview.loadUrl(uriText.getText().toString());
```

So, that means that if I enter a file url into the input section, I can access that file. To test it, I made a dummy file.



```
vbox86p:/ # cd /data/data jakhar.aseem.diva
/system/bin/sh: cd: bad substitution
2|vbox86p:/ # cd /data/data/jakhar.aseem.diva
vbox86p:/data/data/jakhar.aseem.diva # ls
app_textures app_webview cache code_cache databases lib shared_prefs uinfo6472807429239572346tmp
vbox86p:/data/data/jakhar.aseem.diva # echo "username: Jaivardhan , password: Khanna " >> Input2.txt
vbox86p:/data/data/jakhar.aseem.diva # ls
Input2.txt app_textures app_webview cache code_cache databases lib shared_prefs uinfo6472807429239572346tmp
vbox86p:/data/data/jakhar.aseem.diva #
```

Now, If I enter the file path as the input parameter, I get:



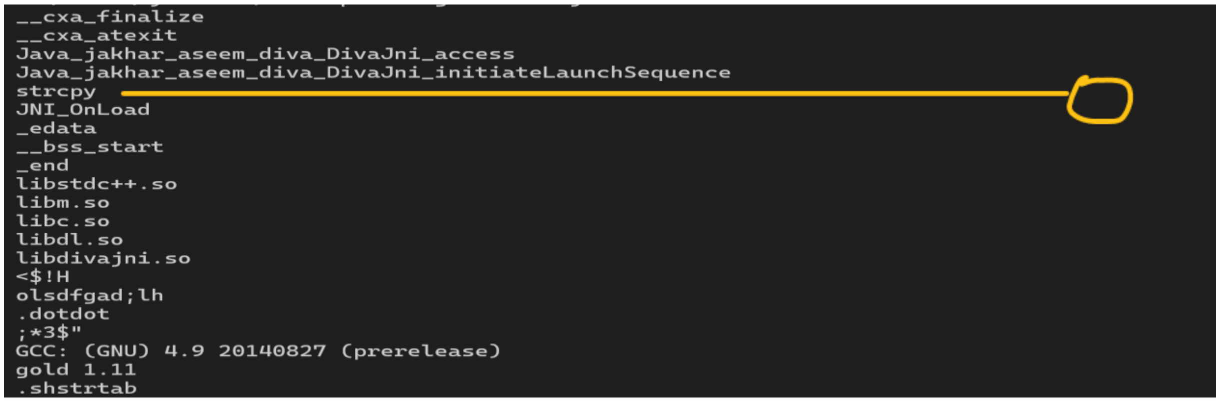
```
username: Jaivardhan , password: Khanna
```

X. Input Data Validation Issue 3

This time the goal is to NOT to find the hidden code, but to crash the app.

Upon code inspection of this part, I noticed that there is a class being imported called "DivaJni", like it did in ([III](#)).

So, I inspected the library file "libdivajni.so"*, I found in ([III](#)):*



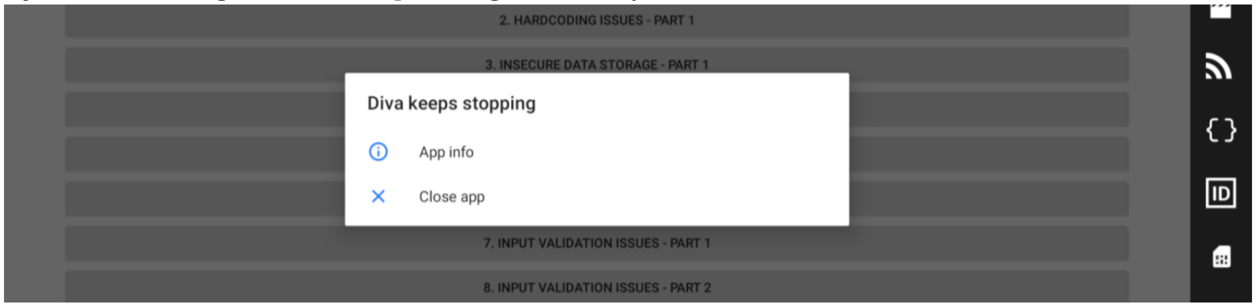
```
__cxa_finalize
__cxa_atexit
Java_jakhar_aseem_diva_DivaJni_access
Java_jakhar_aseem_diva_DivaJni_initiateLaunchSequence
strcpy
JNI_OnLoad
_edata
__bss_start
_end
libstdc++.so
libm.so
libc.so
libdl.so
libdivajni.so
<$!H
olsdfgad;lh
.dotdot
;*3$"
GCC: (GNU) 4.9 20140827 (prerelease)
gold 1.11
.shstrtab
```

The possibility of a usage of 'strcpy' *makes the app vulnerable to* '*Buffer-Overflow*'.

To put this to test, I started spamming random characters until the string was long enough, which came to be:

"HelloFROM(JAIVARDHANKHANNA)3K#9y&#&41470744#$FJr3f;rGJgjt2t9gyyoy69uy9yu-6u-y86u9h689-h82gh5-8gh59g5989gh5-h94hg994hg84g9h95hg"

After entering this as input, I got the system error:


2. HARDCODING ISSUES - PART 1
3. INSECURE DATA STORAGE - PART 1
7. INPUT VALIDATION ISSUES - PART 1
8. INPUT VALIDATION ISSUES - PART 2

Diva keeps stopping

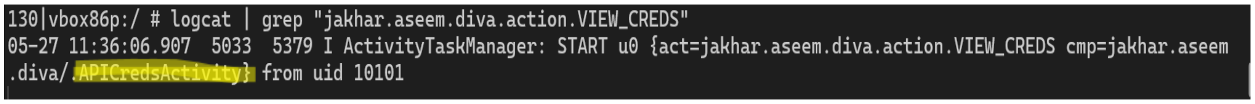
App info

Close app

XI. Access Control Issue 1

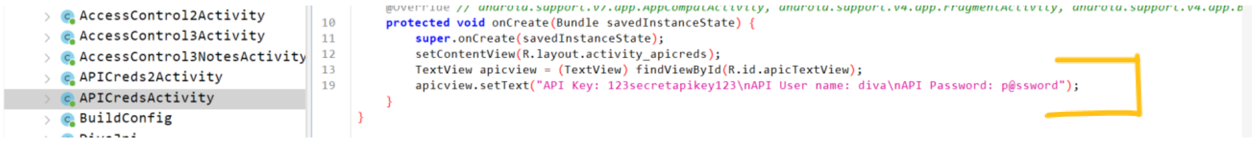
The goal is to access the API credentials outside from the app.
Upon inspection of the code, I could see, that the button started an activity "jakhar.aseem.diva.action.VIEW_CREDS". Therefore, upon using:

logcat | grep "jakhar.aseem.diva.action.VIEW_CREDS"



```
130|vbox86p:/ # logcat | grep "jakhar.aseem.diva.action.VIEW_CREDS"
05-27 11:36:06.907  5033  5379 I ActivityTaskManager: START u0 {act=jakhar.aseem.diva.action.VIEW_CREDS cmp=jakhar.aseem
.diva/.APICredsActivity} from uid 10101
```

When I inspected the "APICredsActivity", I could see the hardcoded credentials.



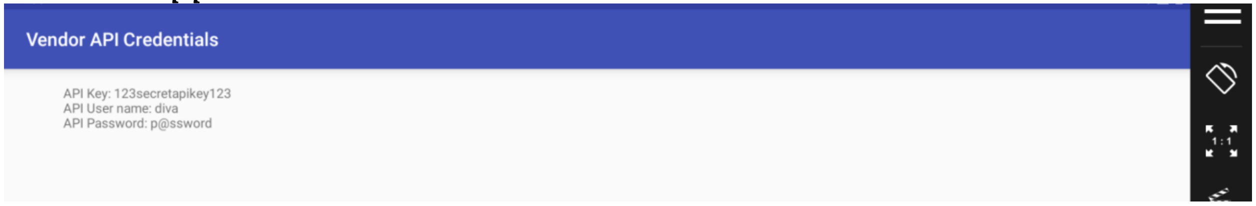
```
AccessControl2Activity
AccessControl3Activity
AccessControl3NotesActivity
APICreds2Activity
APICredsActivity
BuildConfig
```

```
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.B
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_apicreds);
        TextView apicview = (TextView) findViewById(R.id.apicTextView);
        apicview.setText("API Key: 123secretapikey123\nAPI User name: diva\nAPI Password: p@ssword");
    }
}
```

Now that I knew which activity was used to store the credentials, I could bypass the button in the app from the shell, using:

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

This causes the screen to show credentials without having to interact with the app.



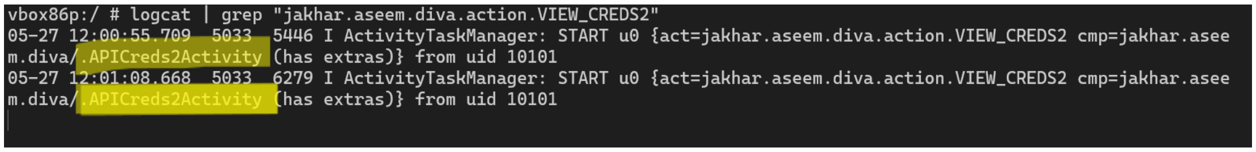
Vendor API Credentials

API Key: 123secretapikey123
API User name: diva
API Password: p@ssword

XII. Access Control Issue 2

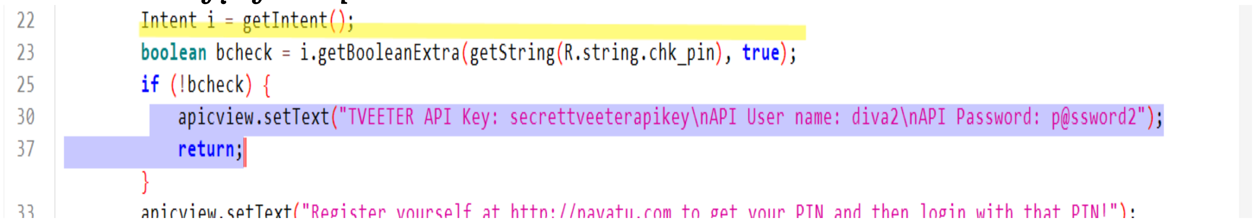
The goal is to find access the api credentials for 'Tveeter' without the pin and outside the app.
Upon inspection of the code, I could see, that the button started an activity "jakhar.aseem.diva.action.VIEW_CREDS2". Therefore, upon using:

logcat | grep "jakhar.aseem.diva.action.VIEW_CREDS2"



```
vbox86p:/ # logcat | grep "jakhar.aseem.diva.action.VIEW_CREDS2"
05-27 12:00:55.709  5033  5446 I ActivityTaskManager: START u0 {act=jakhar.aseem.diva.action.VIEW_CREDS2 cmp=jakhar.asee
m.diva/.APICreds2Activity (has extras)} from uid 10101
05-27 12:01:08.668  5033  6279 I ActivityTaskManager: START u0 {act=jakhar.aseem.diva.action.VIEW_CREDS2 cmp=jakhar.asee
m.diva/.APICreds2Activity (has extras)} from uid 10101
```

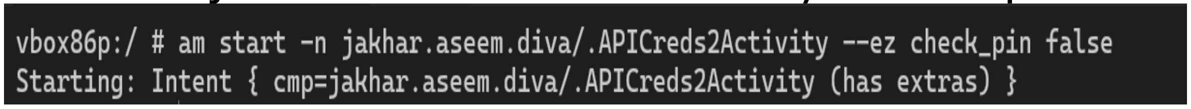
When I inspected the "APICredsActivity", I could see the hardcoded credentials. However, in this case, I also noticed a 'boolean-check' was there to verify if the pin was there or not.



```
        Intent i = getIntent();
        boolean bcheck = i.getBooleanExtra(getString(R.string.chk_pin), true);
        if (!bcheck) {
            apicview.setText("TVEETER API Key: secrettveeterapikey\nAPI User name: diva2\nAPI Password: p@ssword2");
            return;
        }
        apicview.setText("Register yourself at http://payatu.com to get your PIN and then login with that PIN!");
```

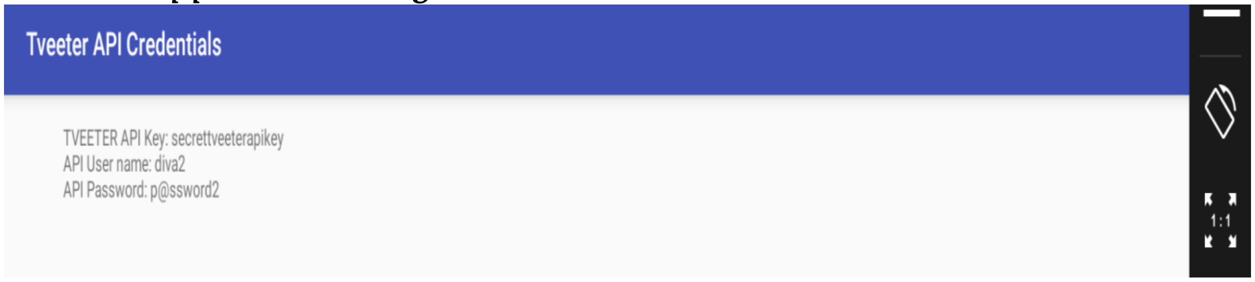
Now that I knew which activity was used to store the credentials, I could bypass the button in the app from the shell, using:

am start -n jakhar.aseem.diva/.APICreds2Activity --ez check_pin false



```
vbox86p:/ # am start -n jakhar.aseem.diva/.APICreds2Activity --ez check_pin false
Starting: Intent { cmp=jakhar.aseem.diva/.APICreds2Activity (has extras) }
```

This causes the screen to show credentials without having to interact with the app and entering the Pin.

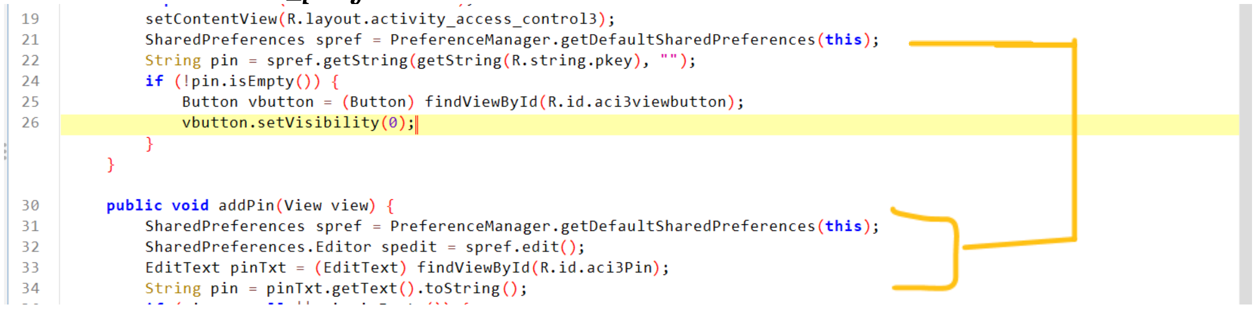


Tveeter API Credentials

TVEETER API Key: secrettveeterapikey
API User name: diva2
API Password: p@ssword2

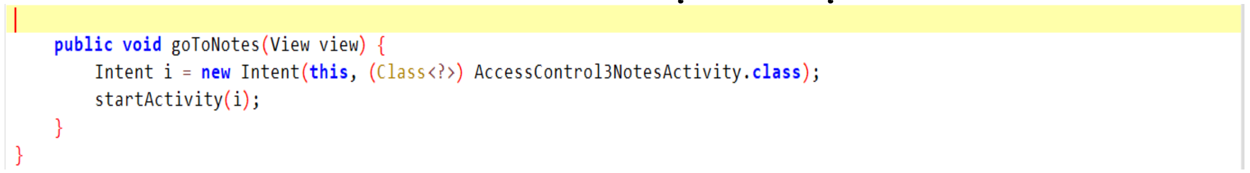
XIII. Access Control Issue 3

The goal is to access the notes of the app outside of the app.
Upon inspection of the code, I noticed that registered pins are getting stored in "shared_prefs":



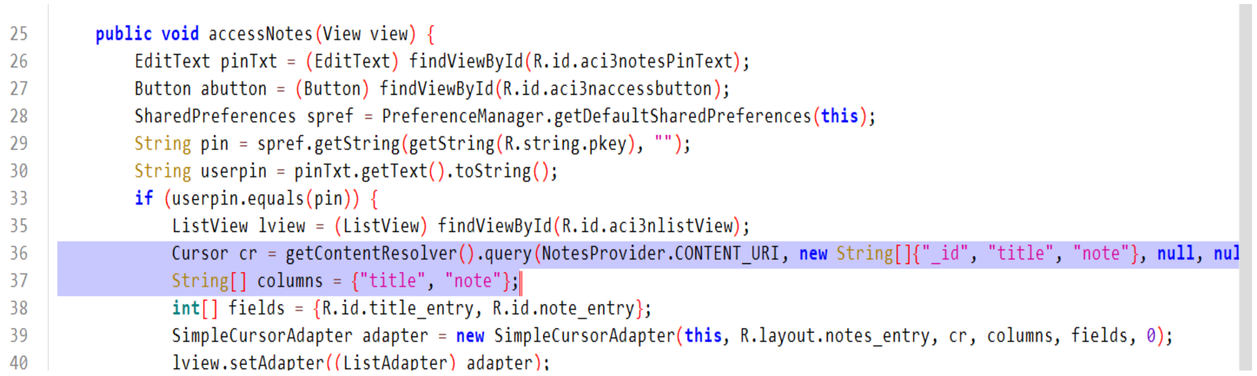
```
        setContentView(R.layout.activity_access_control3);
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        String pin = spref.getString(getString(R.string.pkey), "");
        if (!pin.isEmpty()) {
            Button vbutton = (Button) findViewById(R.id.aci3viewbutton);
            vbutton.setVisibility(0);
        }
    }

    public void addPin(View view) {
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor spedit = spref.edit();
        EditText pinTxt = (EditText) findViewById(R.id.aci3Pin);
        String pin = pinTxt.getText().toString();
```

Therefore, when I entered the pin that was saved in "shared_prefs", it launches the "AccessControl3NotesActivity" activity:



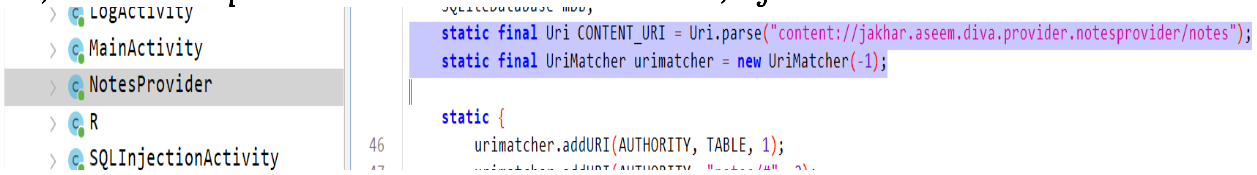
```
    public void goToNotes(View view) {
        Intent i = new Intent(this, (Class<?>) AccessControl3NotesActivity.class);
        startActivity(i);
    }
}
```

Upon inspecting the "AccessControl3NotesActivity", I noticed that after validating the pin, the notes are shown via a content query with the "NotesProvider" class:



```
    public void accessNotes(View view) {
        EditText pinTxt = (EditText) findViewById(R.id.aci3notesPinText);
        Button abutton = (Button) findViewById(R.id.aci3naccessbutton);
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        String pin = spref.getString(getString(R.string.pkey), "");
        String userpin = pinTxt.getText().toString();
        if (userpin.equals(pin)) {
            ListView lview = (ListView) findViewById(R.id.aci3nlistView);
            Cursor cr = getContentResolver().query(NotesProvider.CONTENT_URI, new String[]{"_id", "title", "note"}, null, nul
            String[] columns = {"title", "note"};
            int[] fields = {R.id.title_entry, R.id.note_entry};
            SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.notes_entry, cr, columns, fields, 0);
            lview.setAdapter((ListAdapter) adapter);
```

So, when I inspect the "NotesProvider" class, I find the URI:

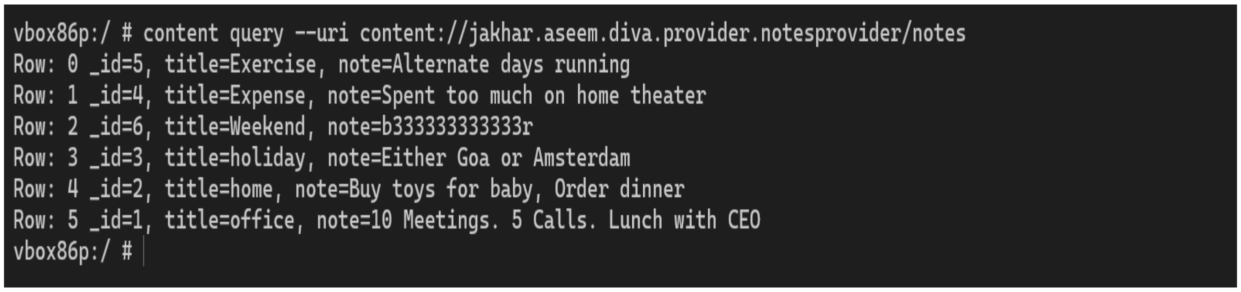


```
LogActivity
MainActivity
NotesProvider
R
SQLInjectionActivity
```

```
    static final Uri CONTENT_URI = Uri.parse("content://jakhar.aseem.diva.provider.notesprovider/notes");
    static final UriMatcher urimatcher = new UriMatcher(-1);

    static {
        urimatcher.addURI(AUTHORITY, TABLE, 1);
```

We can access the notes without the app using the command:

content query --uricontent://jakhar.aseem.diva.provider.notesprovider/notes



```
vbox86p:/ # content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes
Row: 0 _id=5, title=Exercise, note=Alternate days running
Row: 1 _id=4, title=Expense, note=Spent too much on home theater
Row: 2 _id=6, title=Weekend, note=b333333333333r
Row: 3 _id=3, title=holiday, note=Either Goa or Amsterdam
Row: 4 _id=2, title=home, note=Buy toys for baby, Order dinner
Row: 5 _id=1, title=office, note=10 Meetings. 5 Calls. Lunch with CEO
vbox86p:/ #
```

Conclusion

This dynamic analysis of the '[DIVA]' Android application has provided valuable hands-on experience in identifying and exploiting common mobile security vulnerabilities, such as insecure logging, hardcoded secrets, and improper access controls. While '[DIVA]' is intentionally vulnerable for educational purposes, the issues uncovered closely mirror real-world threats faced by production Android apps.

After my analysis, I was also interested in how can one prevent such vulnerabilities. I can tell you briefly but here are the references if you want to read the full articles:

1. *Insecure Logging*
   Never log sensitive information such as credentials or personal data. Use log levels appropriately and sanitize logs before release.
2. *Hardcoding Issues*
   Avoid embedding secrets, API keys, or credentials directly in the source code. Use secure storage solutions and environment variables.
3. *Insecure Data Storage*
   Store sensitive data using encrypted storage mechanisms (e.g., Android Keystore, EncryptedSharedPreferences). Avoid storing sensitive data in plain text or external storage.
4. *Input Validation Issues*
   Always validate and sanitize user input on both client and server sides to prevent injection attacks and unintended behavior.
5. *Access Control Issues*
   Implement proper authentication and authorization checks for all sensitive actions and data. Avoid relying solely on client-side controls.

References

- ➢ *https://cheatsheetseries.owasp.org*

  [OWASP Cheat Sheets for secure coding practices]
- ➢ *https://developer.android.com*
- ➢ [official Android Developers portal for documentation]
- ➢ *https://medium.com*

  [Medium hosts community-written articles]
- ➢ *https://www.invicti.com/blog*

  [Invicti's blog covers web and application security topics]
- ➢ *https://www.perplexity.ai*

  [AI-powered search and research assistant useful for quickly gathering information]
- ➢ *https://www.github.com*

  [reference code samples, tools, or community discussions]

(Jaivardhan, 2025)