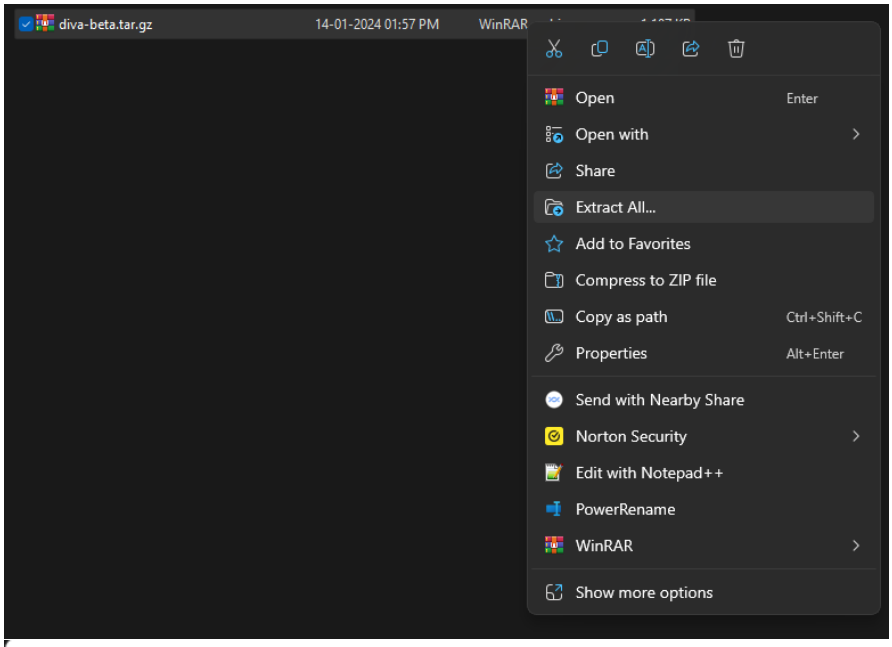
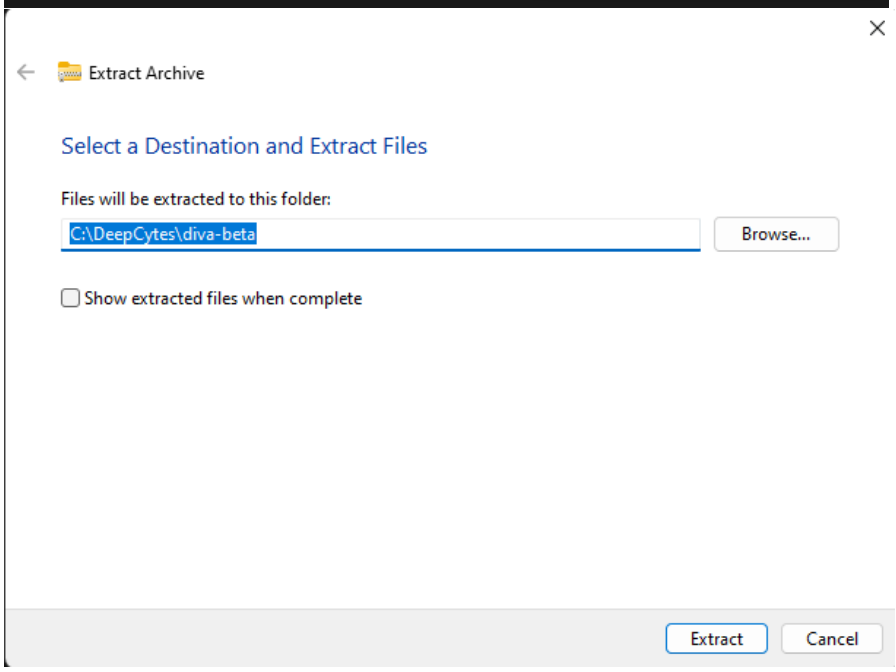
Installation and exploring all vulnerabilities of DIVA using JADX-gui

By Piyush Gayaki

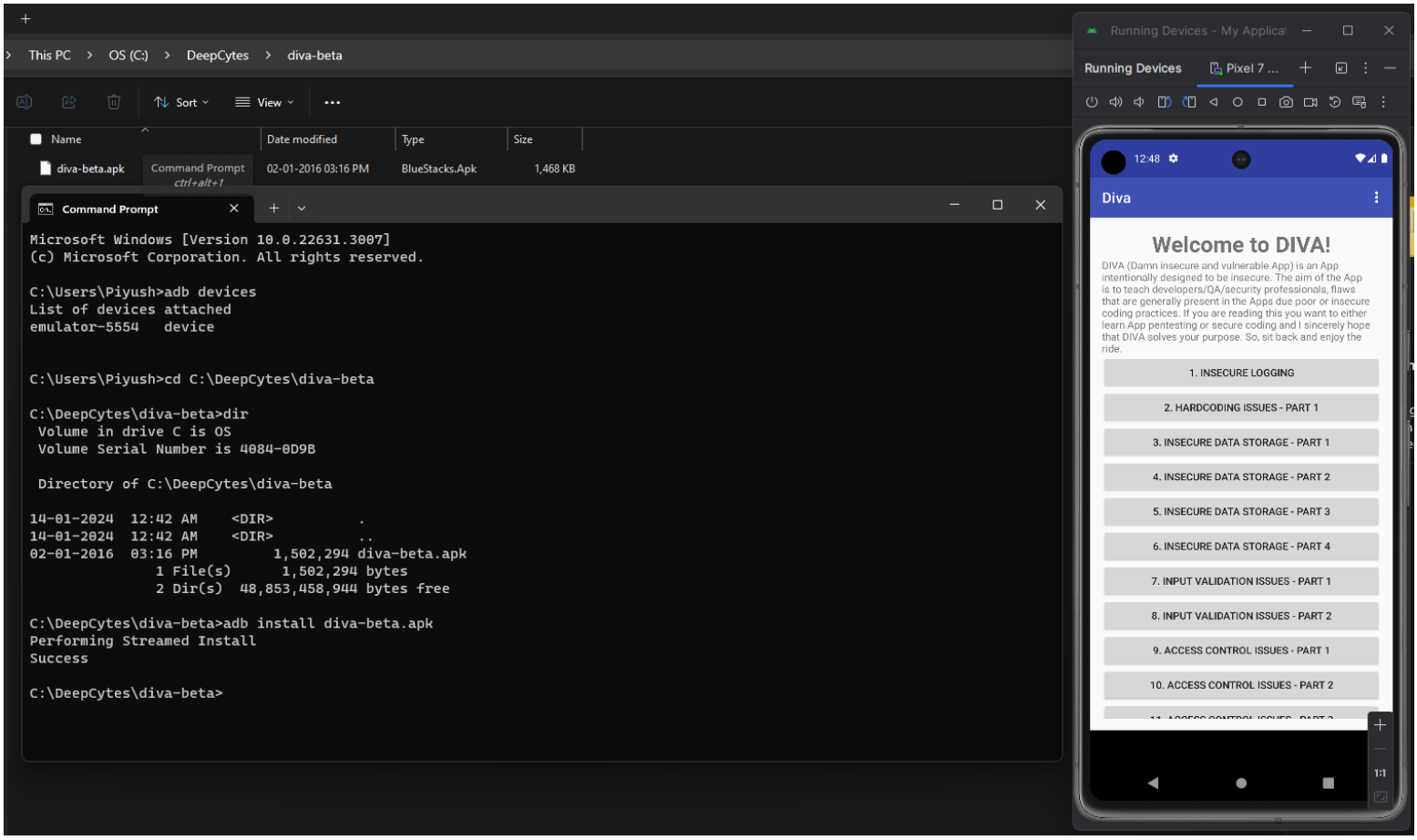
Download the **Diva** from https://payatu.com/wp-content/uploads/2016/01/diva-beta.tar.gz

1. Extract the downloaded file.





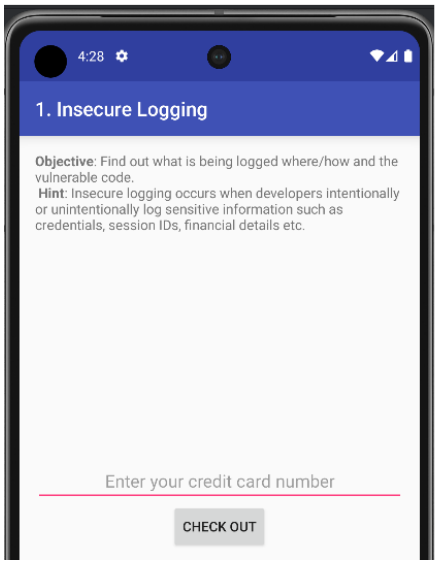
2. Open the folder diva-beta which contains *diva-beta.apk* file.
3. Start the Emulator
4. Check the attached devices using adb command
5. Navigate to the path where we have stored diva-beta.apk file
6. By using “adb install diva-beta.apk” install the application in our emulator.



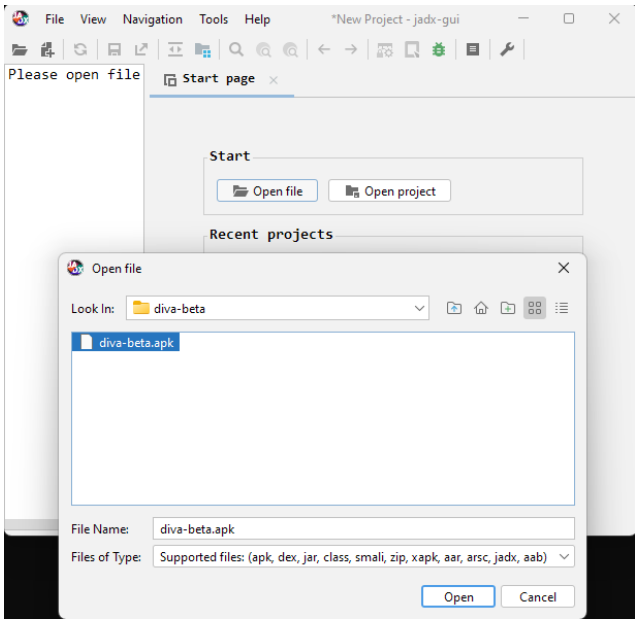
Now, let's Explore DIVA (Damn insecure and vulnerable App

The aim of the App is to teach developers/QA/security professionals, flaws that are generally present in the Apps due poor or insecure coding practices.

1. **Vulnerability is Insecure Logging:** Insecure logging refers to a security vulnerability where sensitive information is improperly handled and stored in log files without adequate protection. This can pose a significant risk as logs are often accessible to various system components and applications.



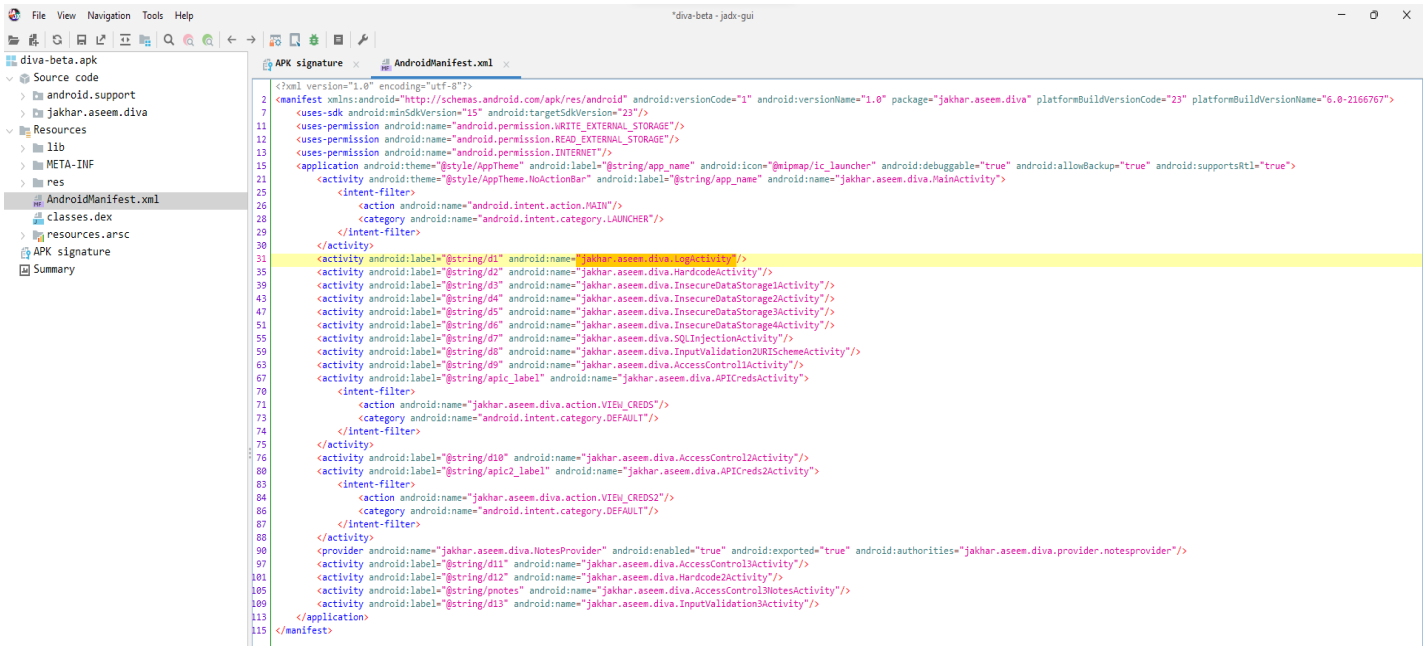
1.1. Open JADX-gui application and select out apk file.



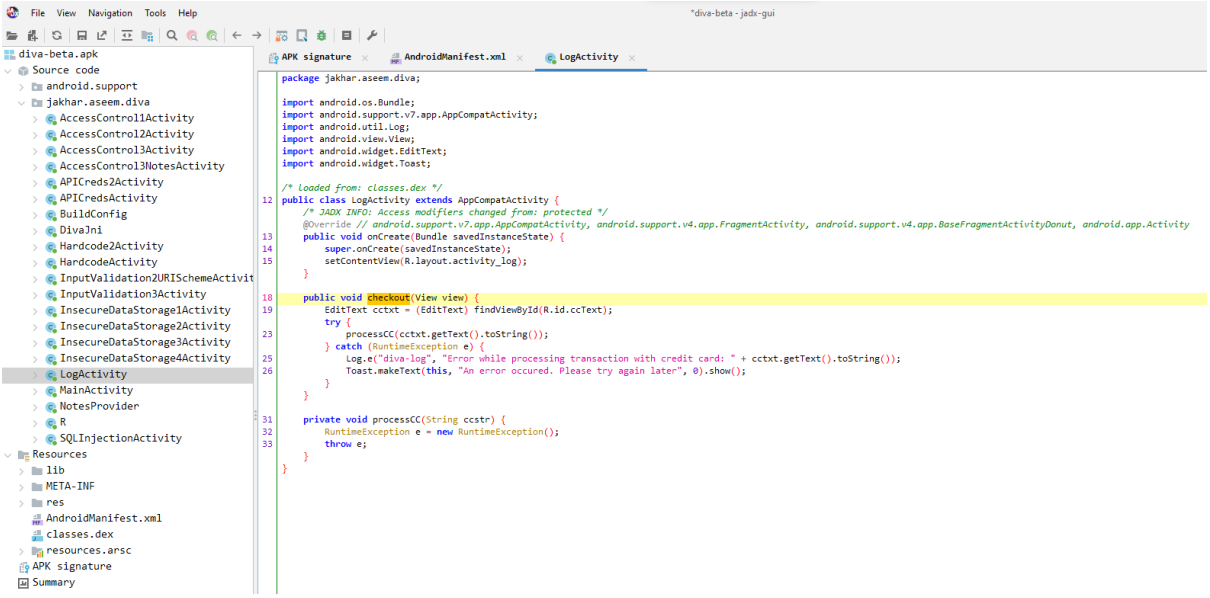
1.2. Go to *AndroidManifest.xml* file **located in Resources folder** as it contains important metadata about an Android app.

1.3. Then we will find the keywords like insecure, logging, logs for activity in AndroidManifest.xml file
(Here the name of Activity is LogActivity).

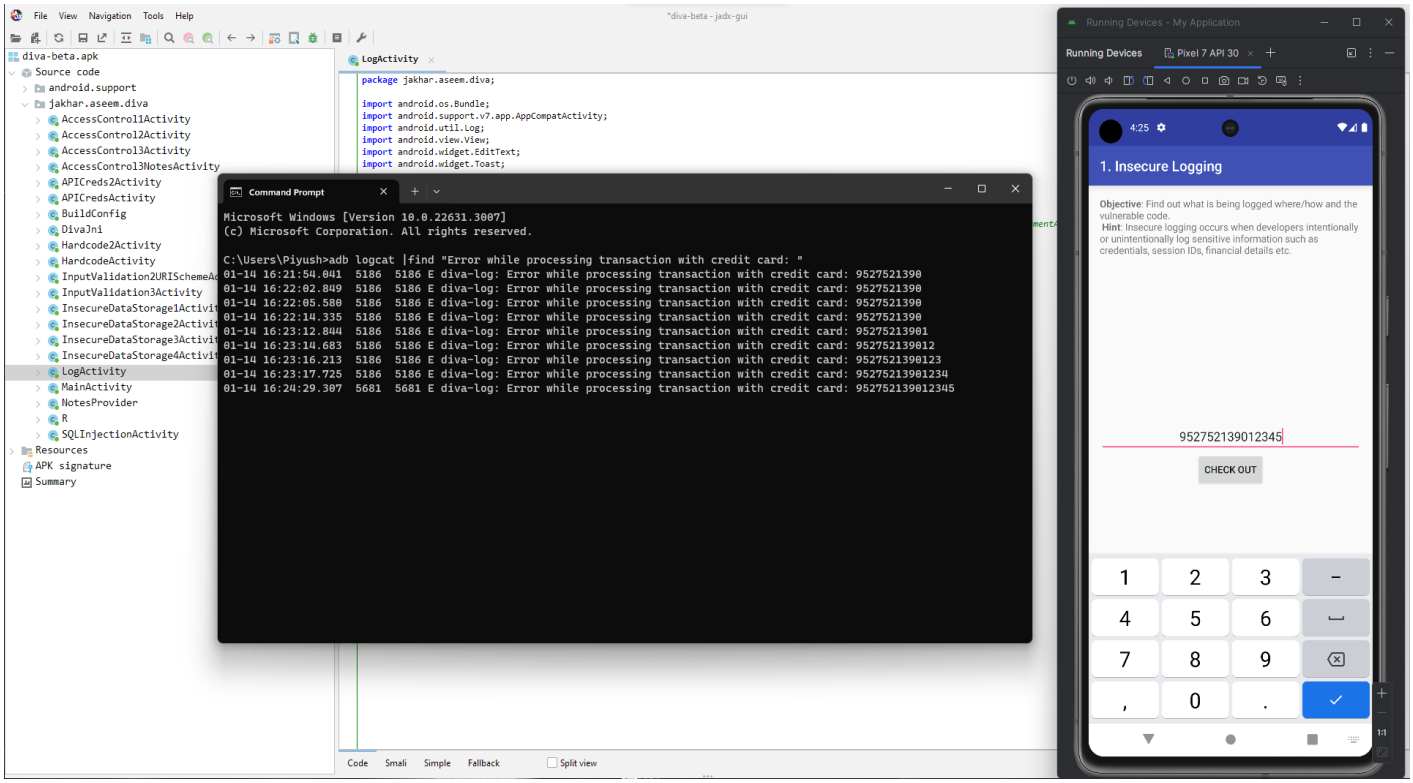
1.4. After finding, follow the hierarchical path of that activity in source code.
(Here the path is Jakhar.aseem.diva.LogActivity).



1.5. Now explore its source code check what methods are stored in particular class.
(Here checkout method is useful as the checkout method is responsible for getting credit card information from the user, attempting to process it, handling any errors that might occur during processing, logging those errors for developer awareness, and providing feedback to the user through a short message.)

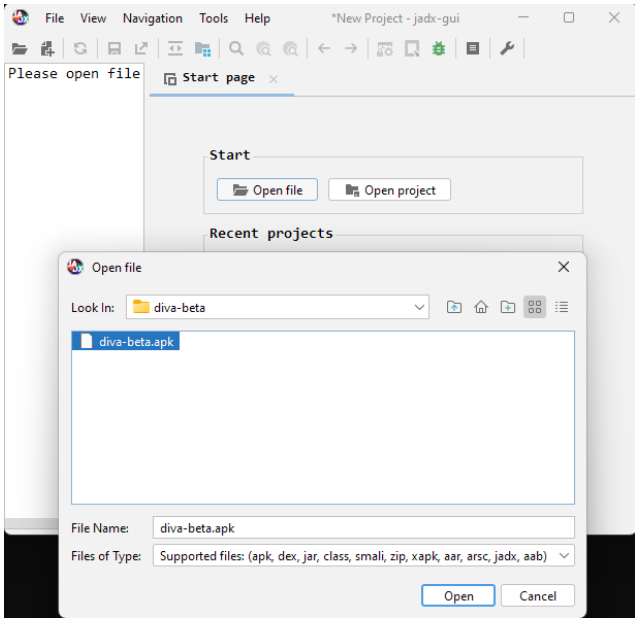


1.6. Open terminal/cmd Type "**adb logcat**" this will show us the log that are generated due to error,
And this log contains the sensitive information without any encryption.
1.7. For more clean result we will Type **adb logcat |find "Error while processing transaction with credit card:"**



2. **Hardcoding Part 1:** The goal is to understand and locate hardcoded elements, specifically the vendor key, which functions similarly to a license key, enabling access to the application. **So we have to get the vendor key.**

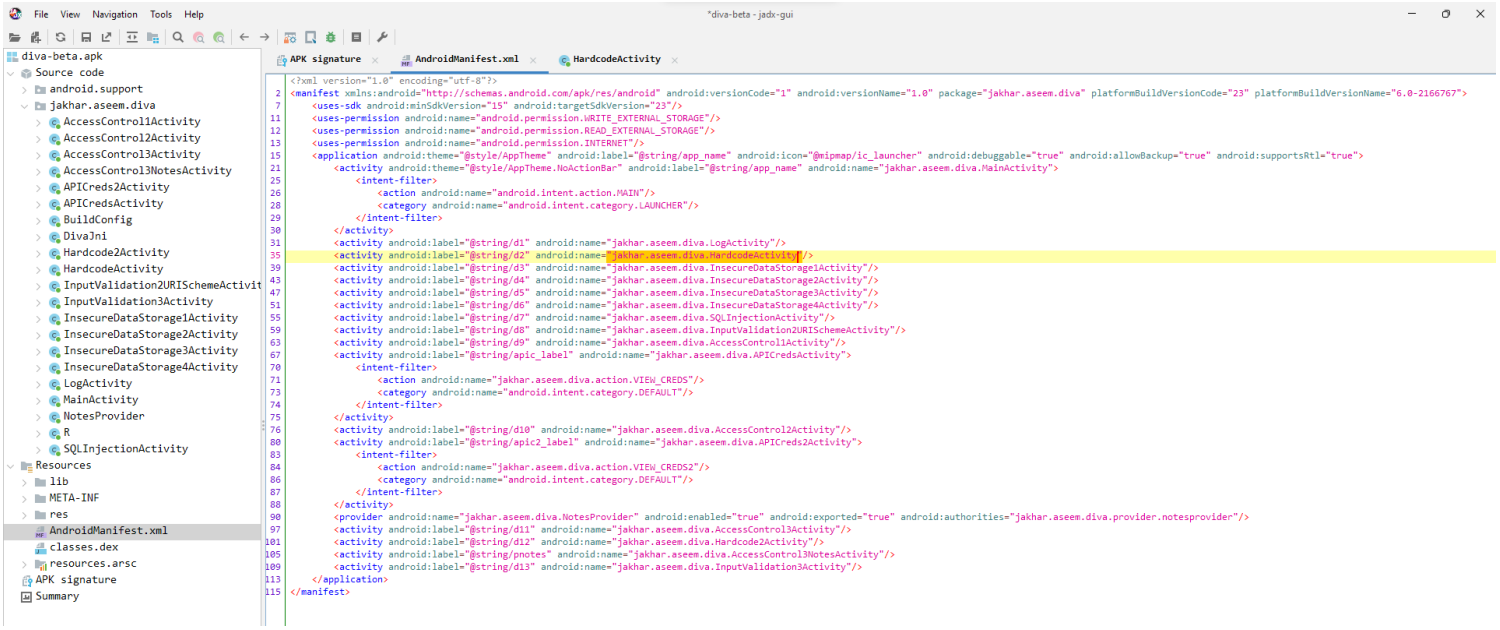
2.1. Open JADX-gui application and select out apk file.



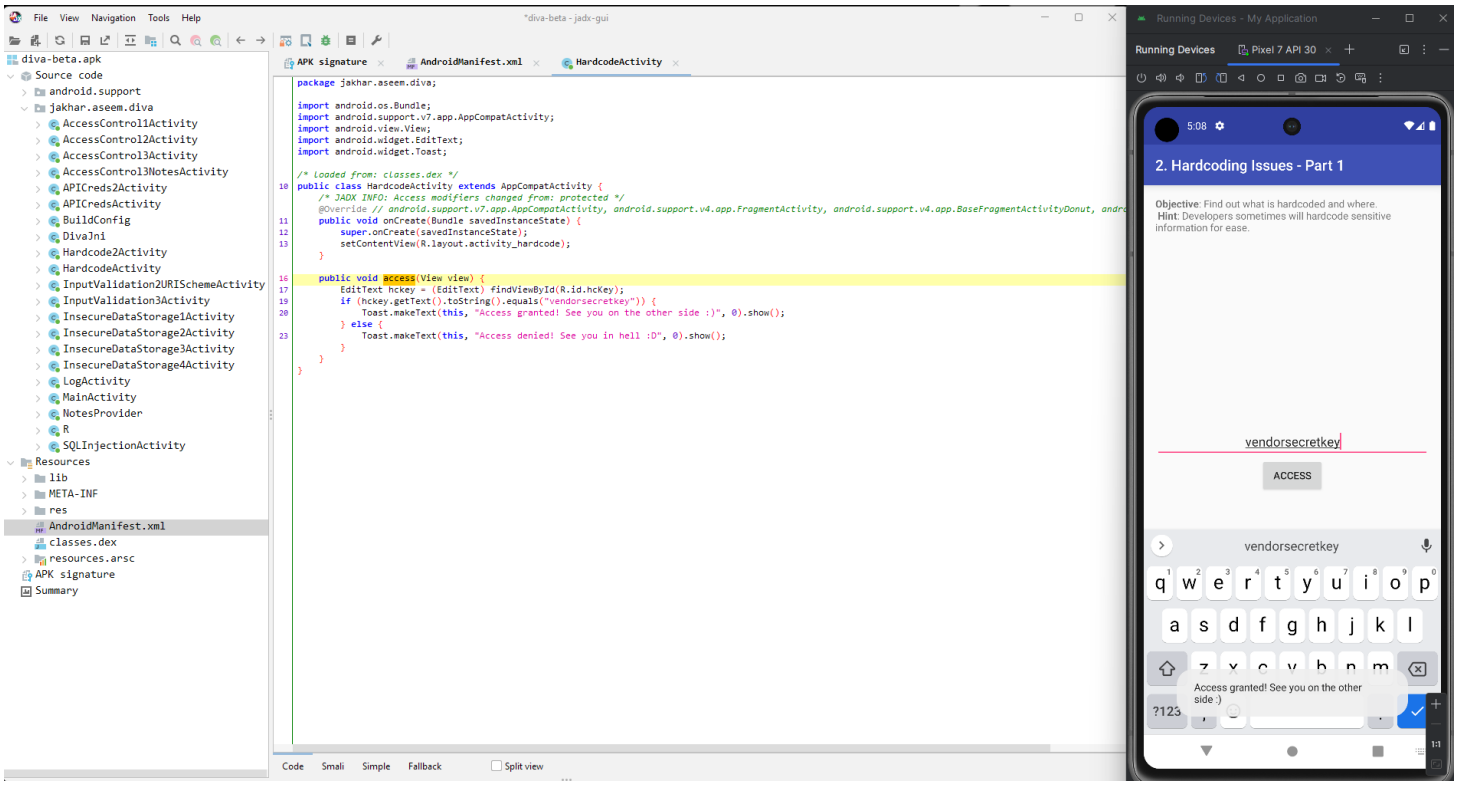
2.2. Go to *AndroidManifest.xml* file **located in Resources folder** as it contains important metadata about an Android app.

2.3. Then we will find the keywords like Hardcode for activity in AndroidManifest.xml file
(Here the name of Activity is HardcodeActivity).

2.4. After finding, follow the hierarchical path of that activity in source code.
(Here the path is Jakhar.aseem.diva.HardcodeActivity).



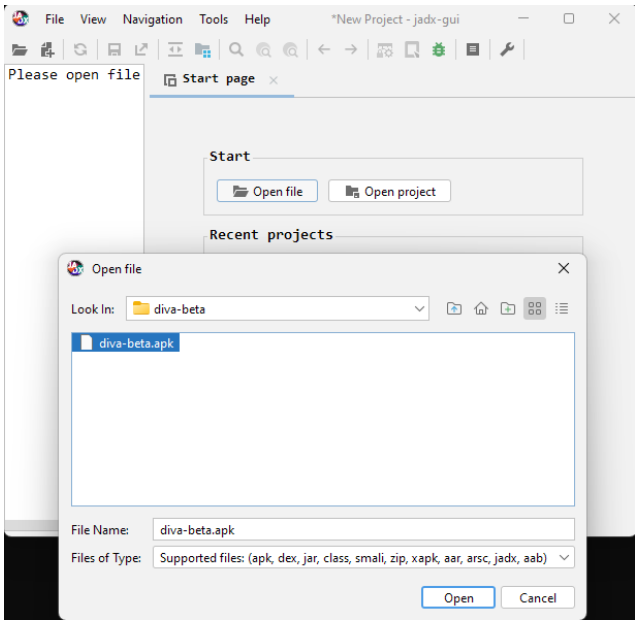
2.5. Now explore its source code check what methods are stored in *HardcodeActivity* class.
*(Here **access** method is useful as the access method checks if the user-entered key is equal to a predefined key ("vendorsecretkey"). If it matches, it shows a positive message; otherwise, it shows a negative message. The showToast method is used to display these messages to the user.)*



So here as we enter ***vendorsecretkey*** in the input filed we get the access.

3. **Hardcoding Part 2:** Same as Part1, here also we have to enter the vendor key.

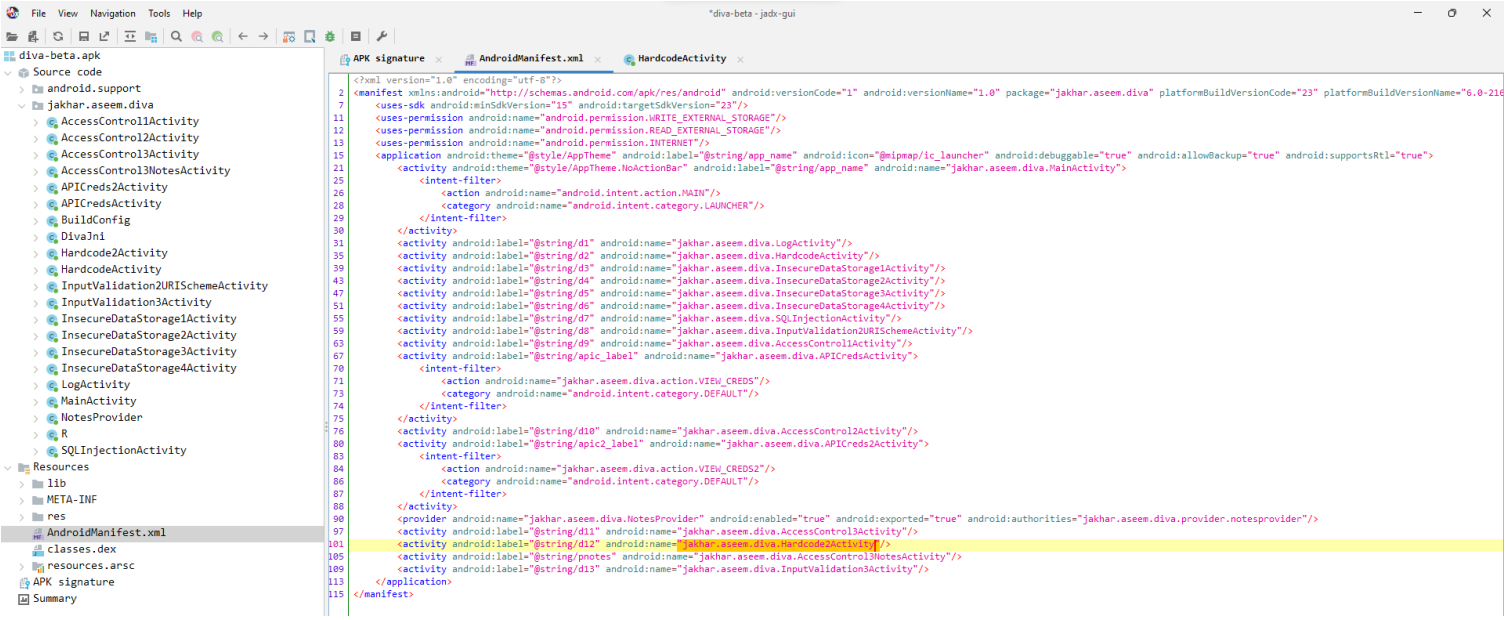
3.1. Open JADX-gui application and select out apk file.



3.2 Go to *AndroidManifest.xml* file **located in Resources folder** as it contains important metadata about an Android app.

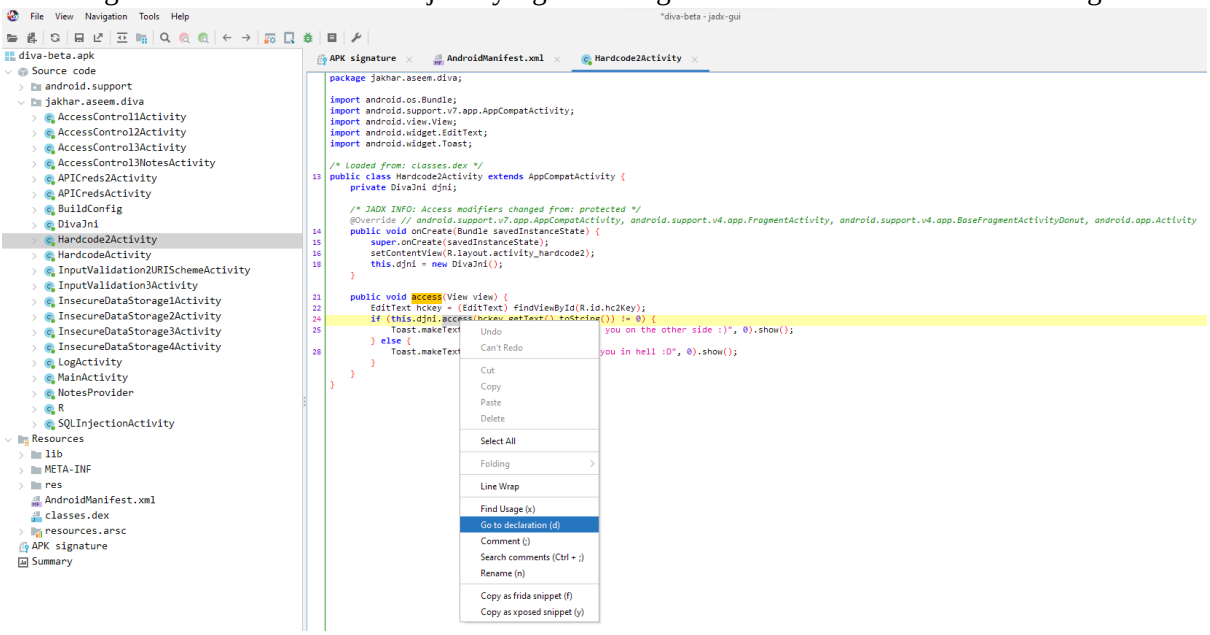
3.3 Then we will find the keywords like Hardcode2, Hardcoding2 for activity in AndroidManifest.xml file
(Here the name of Activity is Hardcode2Activity).

3.4 After finding, follow the hierarchical path of that activity in source code.
(Here the path is Jakhar.aseem.diva.Hardcode2Activity).

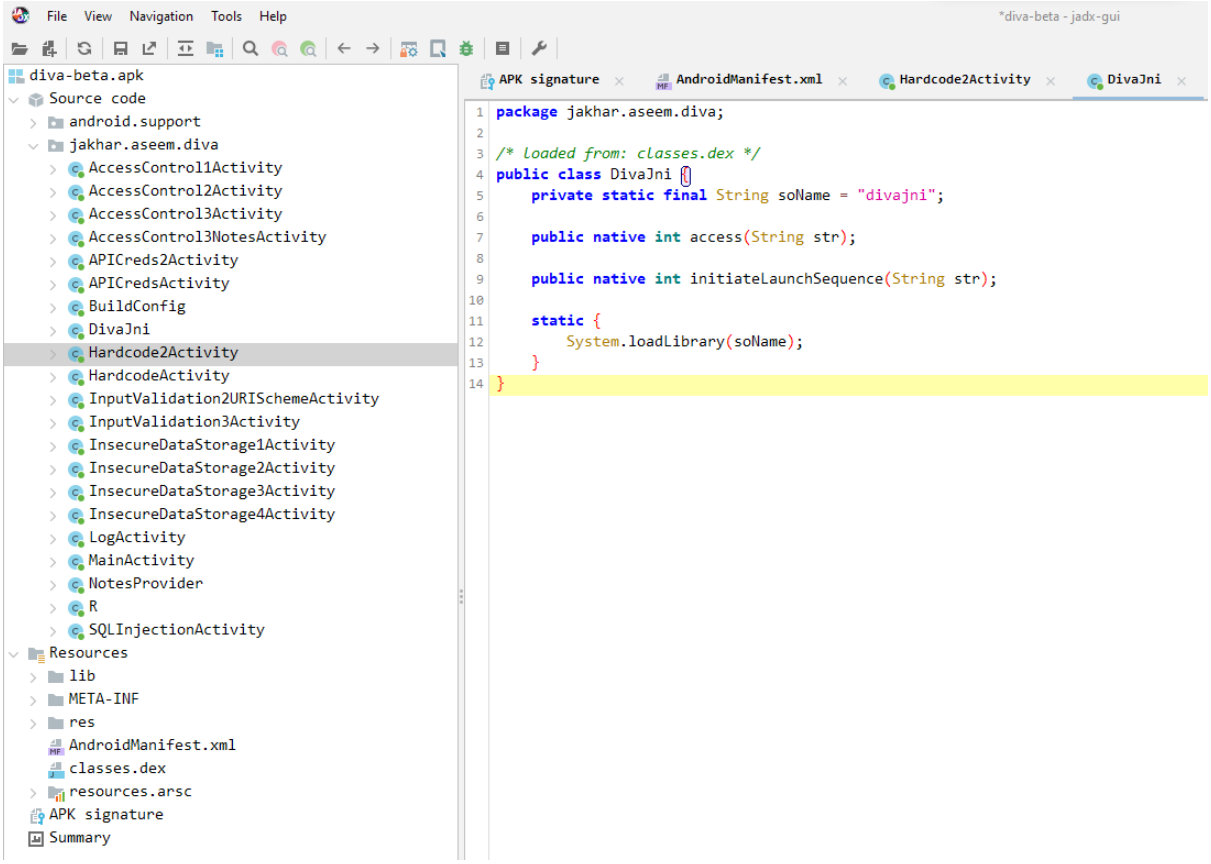


3.5. Now explore its source code check what methods are stored in *Hardcode2Activity* class.
*(Here **access** method is useful as the access method contains if else condition and inside this if condition we are calling the access method with the djni object here and we simply pass some input here as well as the attributes and the vendor key that we will enter here on the application will be sent to this access method.)*

3.6. We will go with this access method just by right clicking on the access method and click on go to declaration

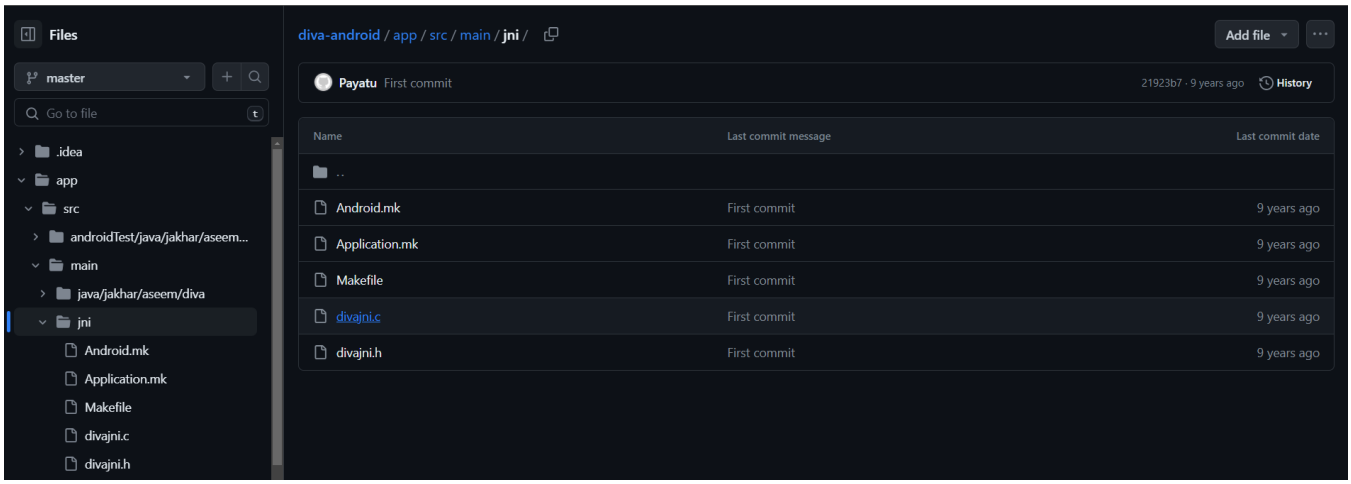


3.7. So now inside declaration we define another variable with the name soName = divajni and the access method will simply accept the string attributes and it will be called for divajni Library, in short, we simply pass our input to the library.

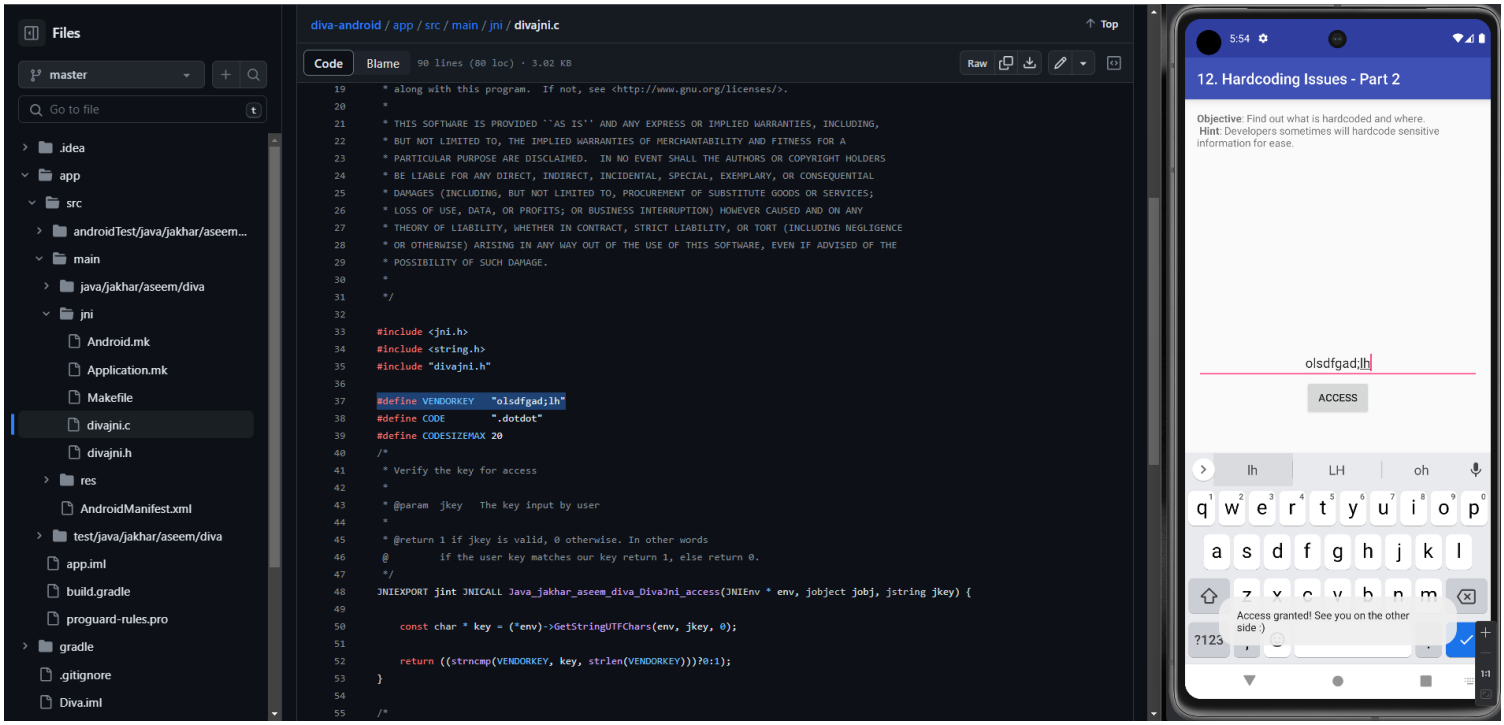


The point here is if we simply pass an input or if we simply want to process some data inside the libraries, we can process the part but we cannot directly extract the source code of the library unless we have the main part, we cannot directly decompile the library source code like we have done for this apk jadx will not do that part, but for this DIVA beta we application we have the source code for that we have to go to

https://github.com/payatu/diva-android/blob/master/app/src/main/jni/divajni.c



3.8. After opening the library and scroll down we can see we got the basic variable declaration here as vendor key this was the basic hard coded issue that was available in our libraries not inside our java source code not inside our strings.xml file not inside our manifest sile not somewhere else this vendor secret key was hidden or stored inside our libraries so if we simply type this “olsdfgad;lh” we get the access.



4. Rooting an emulator: Go to [https://github.com/newbit1/rootAVD?tab=readme-ov-file.](https://github.com/newbit1/rootAVD?tab=readme-ov-file)

4.1. After Download the file simply as zip or using git clone command in cmd
4.2. Unzip and open in cmd

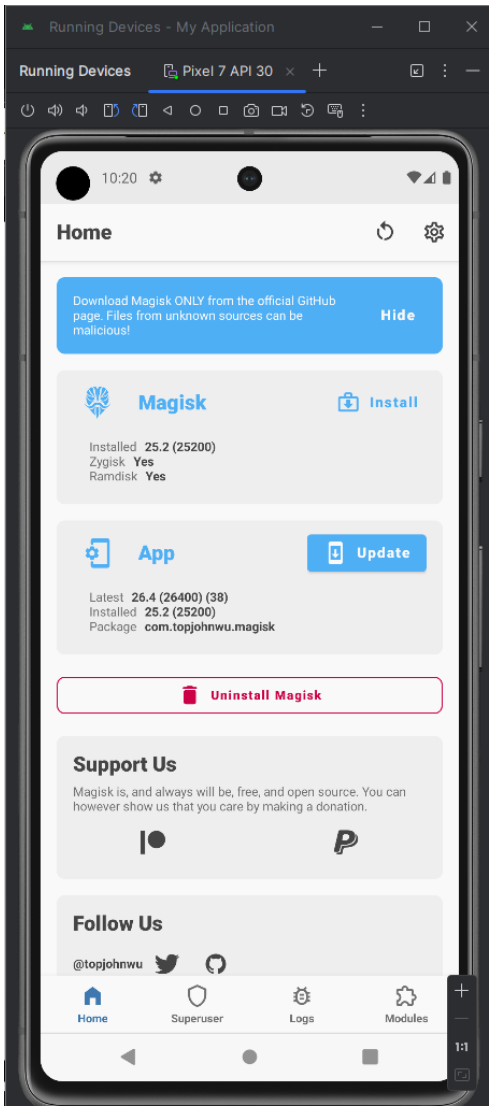
```
C:\Windows\System32\cmd.e

C:\DeepCytes\rootAVD>dir
 Volume in drive C is OS
 Volume Serial Number is 4084-0D9B

 Directory of C:\DeepCytes\rootAVD

14-01-2024  08:21 PM    <DIR>          .
14-01-2024  09:29 PM    <DIR>          ..
14-01-2024  06:43 PM                16 .gitattributes
14-01-2024  06:43 PM                67 .gitignore
14-01-2024  08:21 PM    <DIR>          Apps
14-01-2024  06:43 PM             6,644 CompatibilityChart.md
14-01-2024  06:43 PM            35,823 LICENSE
14-01-2024  06:43 PM        11,278,270 Magisk.zip
14-01-2024  06:43 PM            35,405 README.md
14-01-2024  06:43 PM            17,284 rootAVD.bat
14-01-2024  06:43 PM            82,110 rootAVD.sh
               8 File(s)     11,455,619 bytes
               3 Dir(s)  62,107,172,864 bytes free

C:\DeepCytes\rootAVD>
```

4.3. Then use command “rootAVD.bat”
4.4. Then use command “rootAVD.bat ListAllAVDs”
4.5. Select the path shown and past exactly as it is shown and enter.

```
C:\Windows\System32\cmd.e
- show a Menu, to choose the Magisk Version (Stable || Canary || Alpha), if the AVD is online
- make the choosen Magisk Version to its local
- install all APKs placed in the Apps folder
- use %LOCALAPPDATA%\Android\Sdk to search for AVD system images

Command Examples:
rootAVD.bat
rootAVD.bat ListAllAVDs
rootAVD.bat InstallApps

rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img FAKEBOOTIMG
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img DEBUG PATCHFSTAB GetUSBHPmodZ
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img restore
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img InstallKernelModules
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img InstallPrebuiltKernelModules
rootAVD.bat system-images\android-33\google_apis\x86_64\ramdisk.img InstallPrebuiltKernelModules GetUSBHPmodZ PATCHFSTAB
 DEBUG

rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img FAKEBOOTIMG
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img DEBUG PATCHFSTAB GetUSBHPmodZ
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img restore
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img InstallKernelModules
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img InstallPrebuiltKernelModules
rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img InstallPrebuiltKernelModules GetUSBHPmodZ PAT
CHFSTAB DEBUG


C:\DeepCytes\rootAVD>rootAVD.bat system-images\android-30\google_apis_playstore\x86\ramdisk.img
```

4.6. After this magisk application will get automatically saved in emulator.

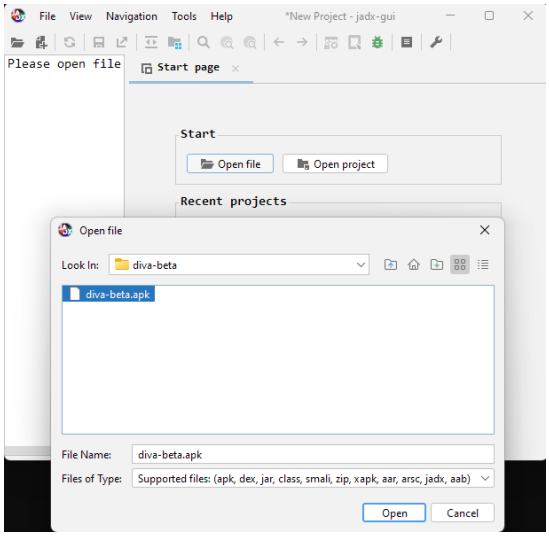
4.7. Open it emulator will get restart after restart open magisk again and click on Superuser option located at bottom centre and start Shell option.



5. **Insecure Data Storage Part 1:** Find out where/how the credentials are being stored and the vulnerable code. We can relate this as whenever we log in into any application like facebook, Instagram whenever we enter the username password, they will be compared on the server side and server will give us a session ID those sessions ID's will be stored somewhere. so here we have to figure out where the sensitive data like session ID's and tokens are stored.

5.1. Initially we will provide some random username and password.

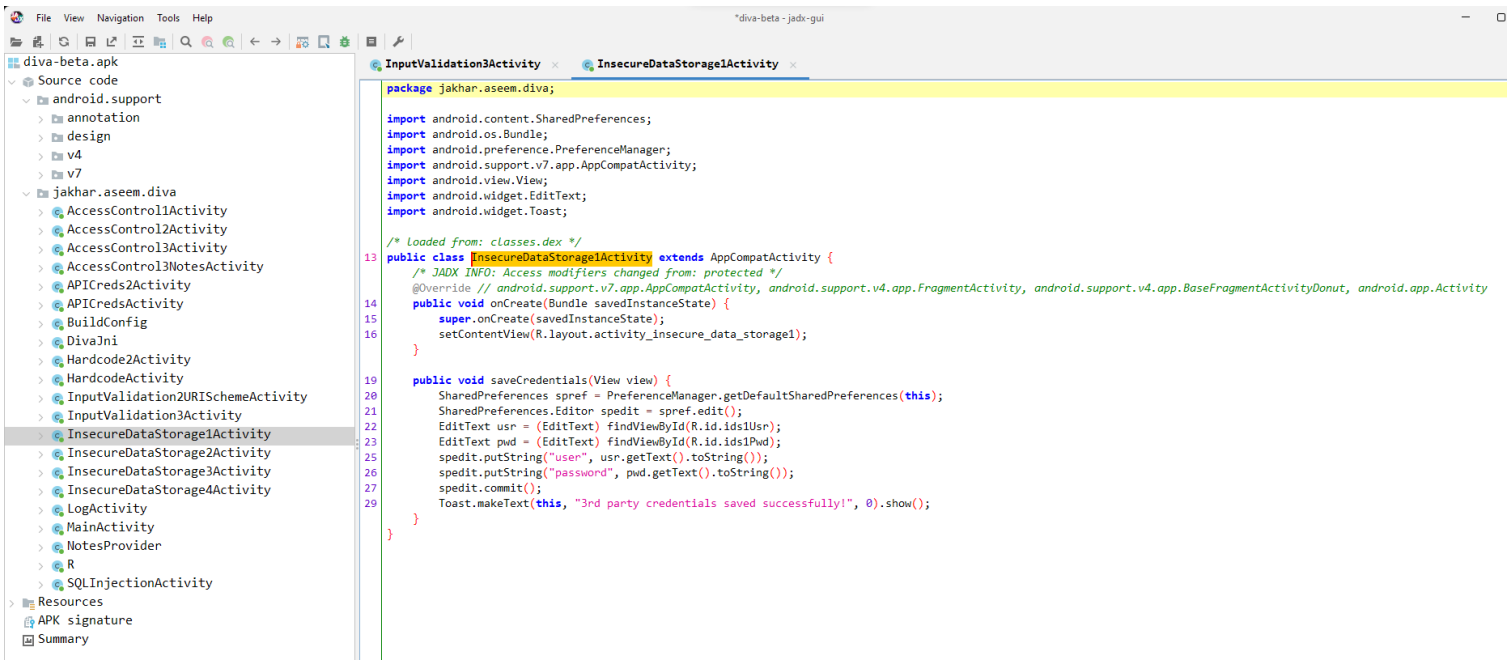
5.2. Start JADX-gui application and select out apk file.



5.3 Go to *AndroidManifest.xml* file **located in Resources folder** as it contains important metadata about an Android app.

5.4 Then we will find the keywords like DataStorage1, InsecureStorage1 for activity in AndroidManifest.xml file
(Here the name of Activity is InsecureDataStorage1Activity).

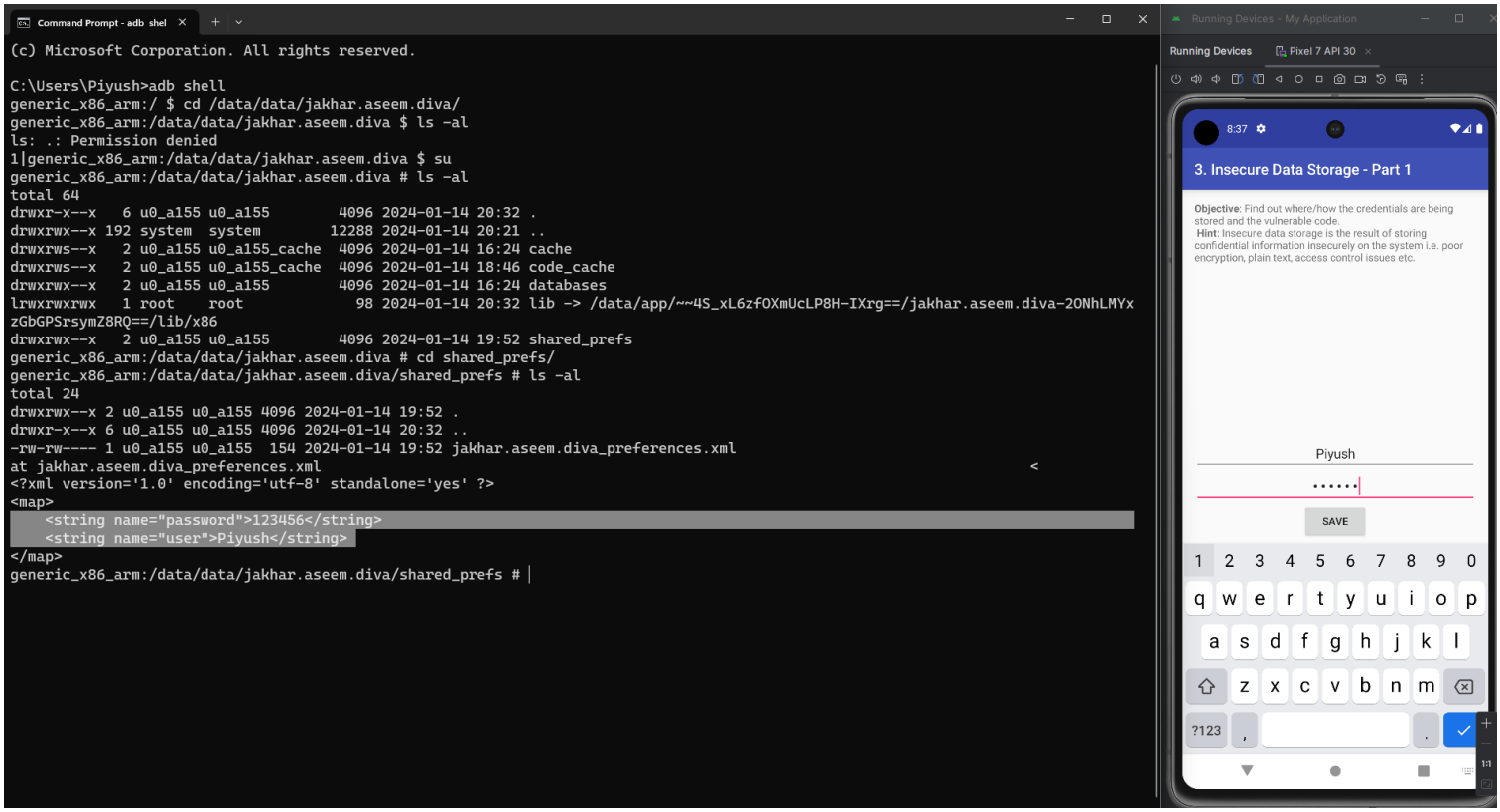
5.5 After finding, follow the hierarchical path of that activity in source code.
(Here the path is Jakhar.aseem.diva.InsecureDataStorage1Activity).



5.6 After Analysing the source code of *Jakhar.aseem.diva.InsecureDataStorage1Activity* file we came to know that the data will be stored inside shared preferences.
[In android we mainly have 4 locations where data is stored 1.Shared Preferences, 2. Database, 3.TempFiles, 4.External Storage]

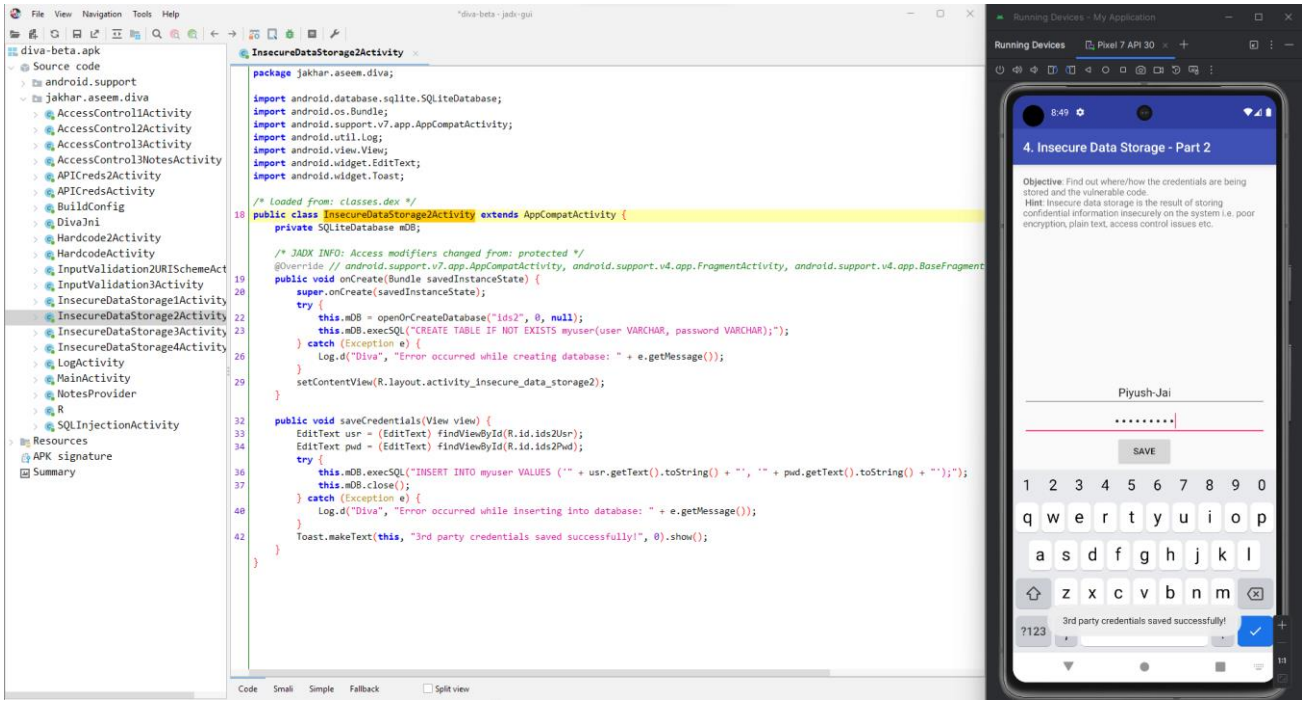
5.7 For Shared Preferences part if we want to see the data we have to go to the location /data/data folder and application package name here *Jakhar.aseem.diva so*, type command "**cd /data/data/jakhar.aseem.diva**/"
To get the folder shared_prefs we have to type command **"ls -al"**
Do **cd shared_prefs** then again do **ls -al** then we get .xml file so to read this file we use command "**cat filename**"

Here the Content such as username and password is directly visible means the stored credentials are in plain text which referred to as an insecure way of storing data.



6. **Insecure Data Storage Part 2:** Same as part 1 here also we have to find out where/how the credentials are being stored and the vulnerable code.

6.1-6.5 Follow same steps as 5.1-5.5

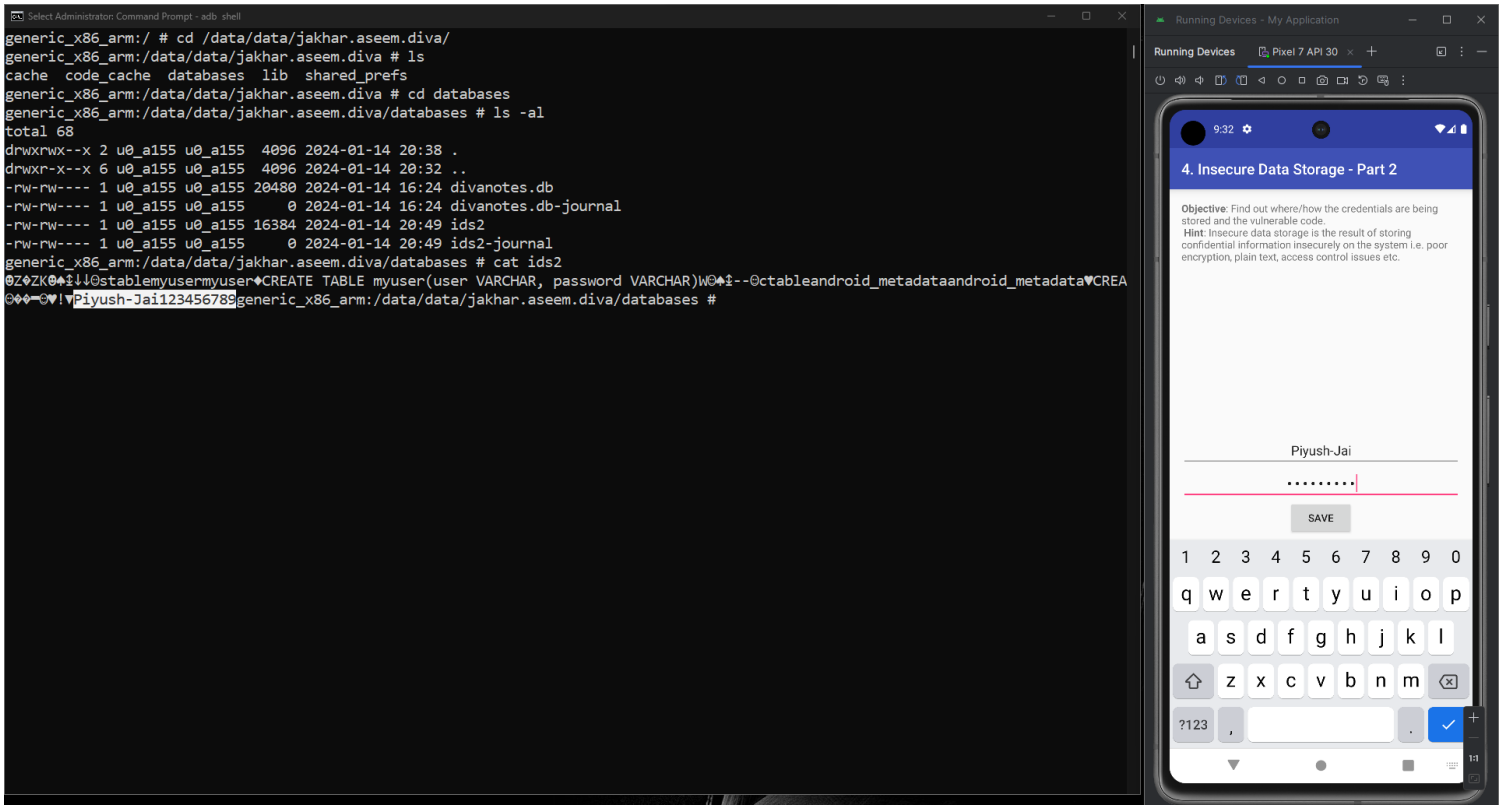


6.6 After Analysing the source code of *Jakhar.aseem.diva.InsecureDataStorage2Activity* we come to know that the data is getting stored in tables and database name is **ids2**.

6.7 Type adb shell command to connect emulator, as we know the path already we directly do cd /data/data/jakhr.aseem.diva/

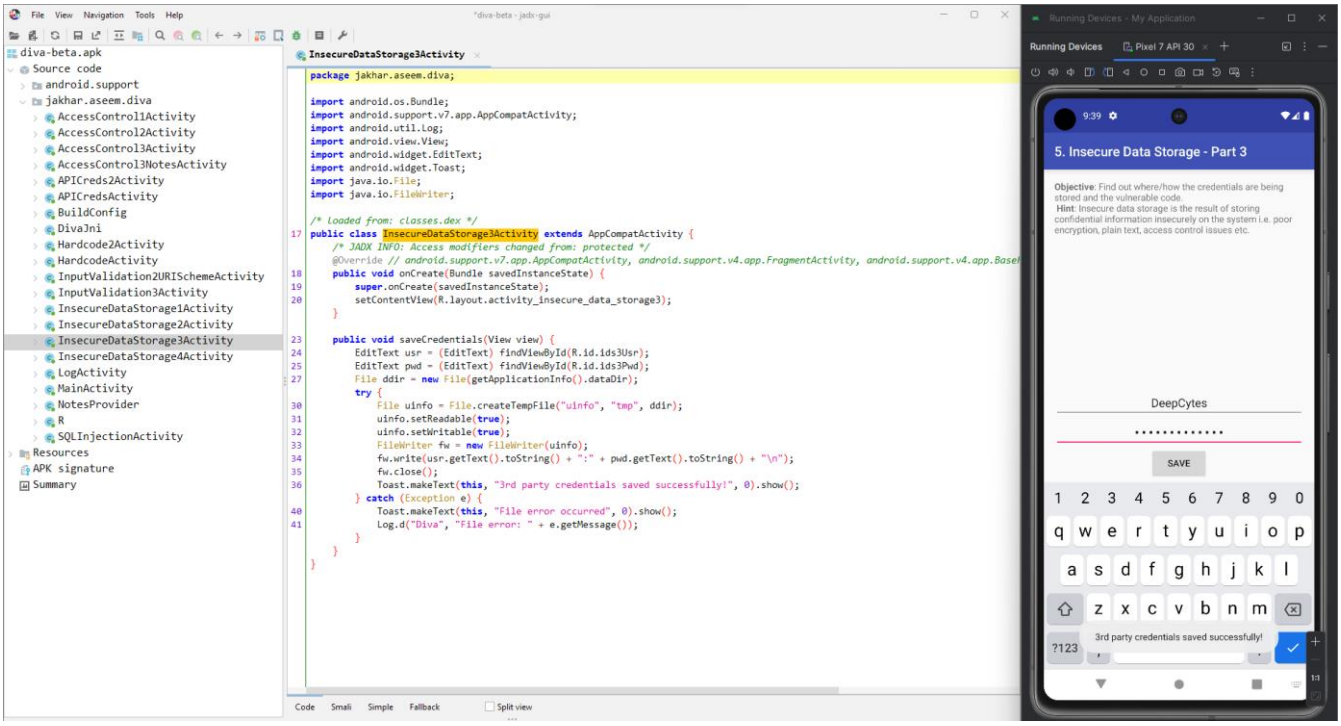
6.8 Type ls -al command we use this because it lists the contents of a directory in a detailed, long format. Includes hidden files and provides comprehensive information about each file or directory.

6.9 Then go to databases folder by cd databases and as we know database name is ids2, use cat ids2 command



7. **Insecure Data Storage Part 3:** Same as part 1,2 here also we have to find out where/how the credentials are being stored and the vulnerable code.

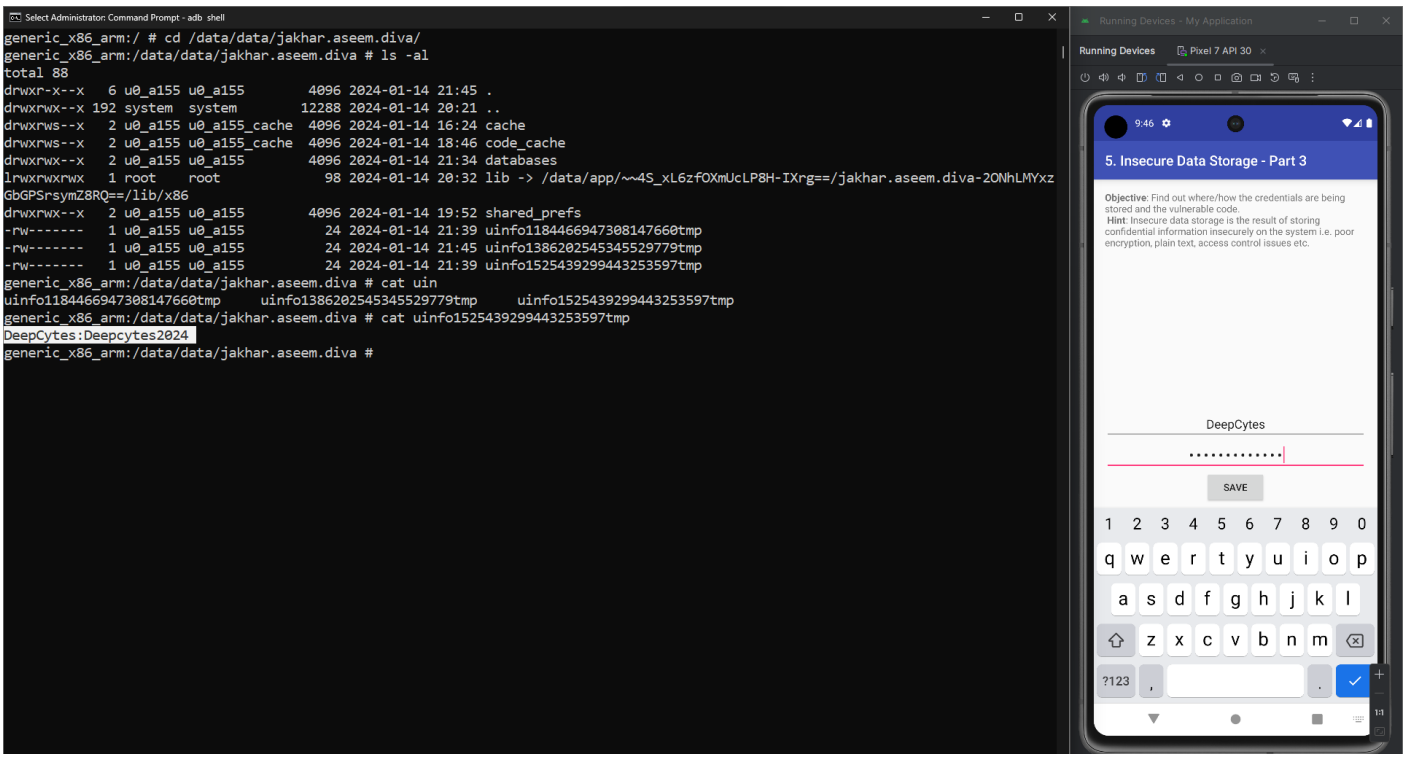
7.1-7.5 Follow same steps as 5.1-5.5



7.6 After Analysing the source code of *Jakhar.aseem.diva.InsecureDataStorage3Activity* we come to know that the data is getting stored in uinfo file type name starts with uinfo and ends with tmp and we has read and write permission.

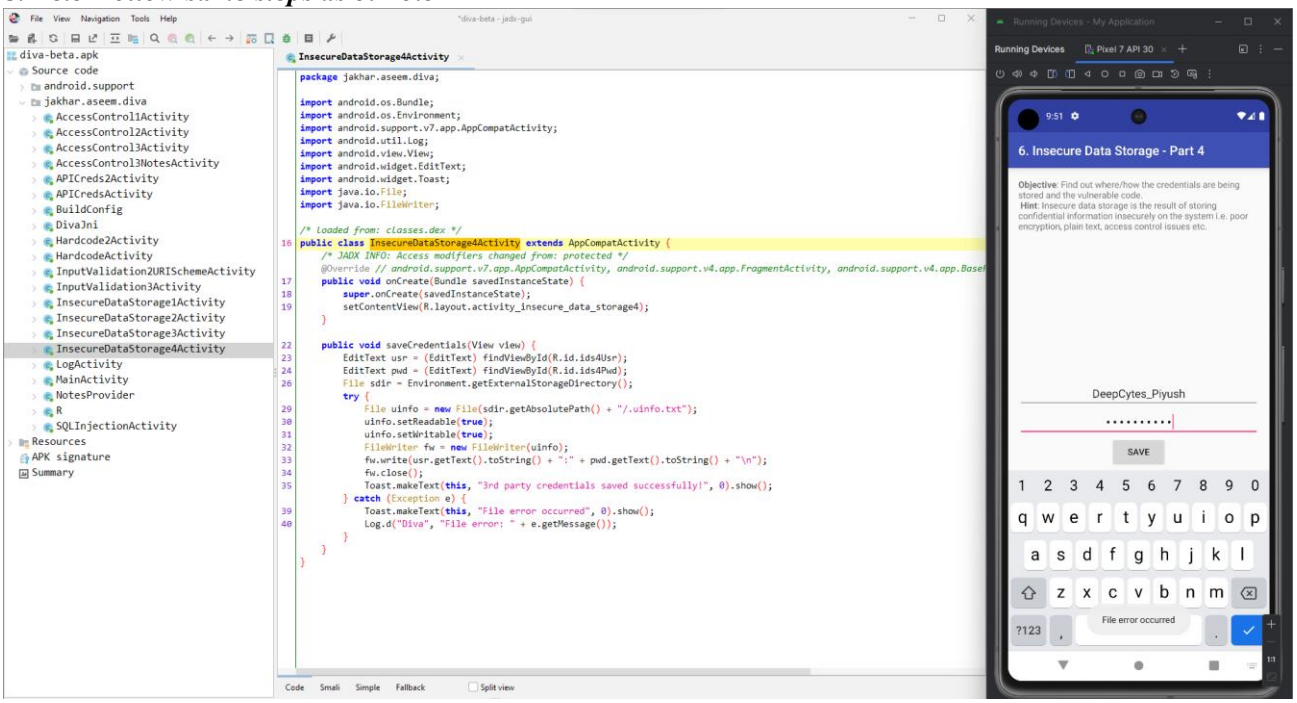
7.7 Same follow steps of 6.7 and 6.8

7.8 Here we get the file starting with unifo and ending with tmp just use cat command and get the contents.

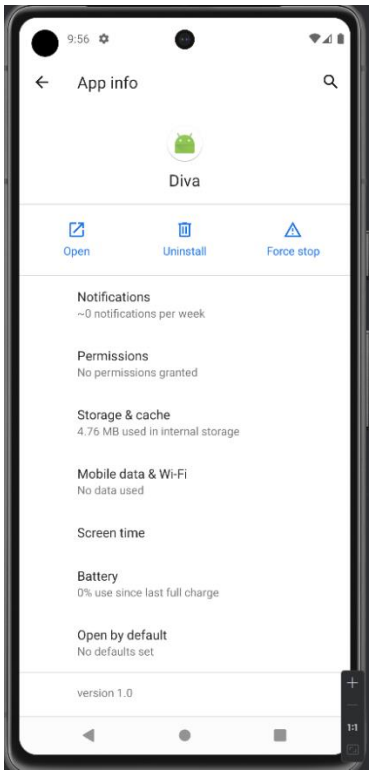


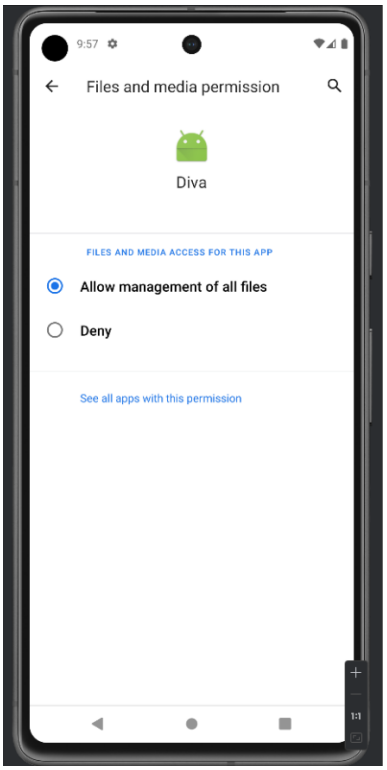
8. **Insecure Data Storage Part 4:** Same as part 1,2,3 here also we have to find out where/how the credentials are being stored and the vulnerable code.

8.1-8.5 Follow same steps as 5.1-5.5



8.6 Initially we will get the error as File error occurred because we don't have permission to sd card. So we have to give permission



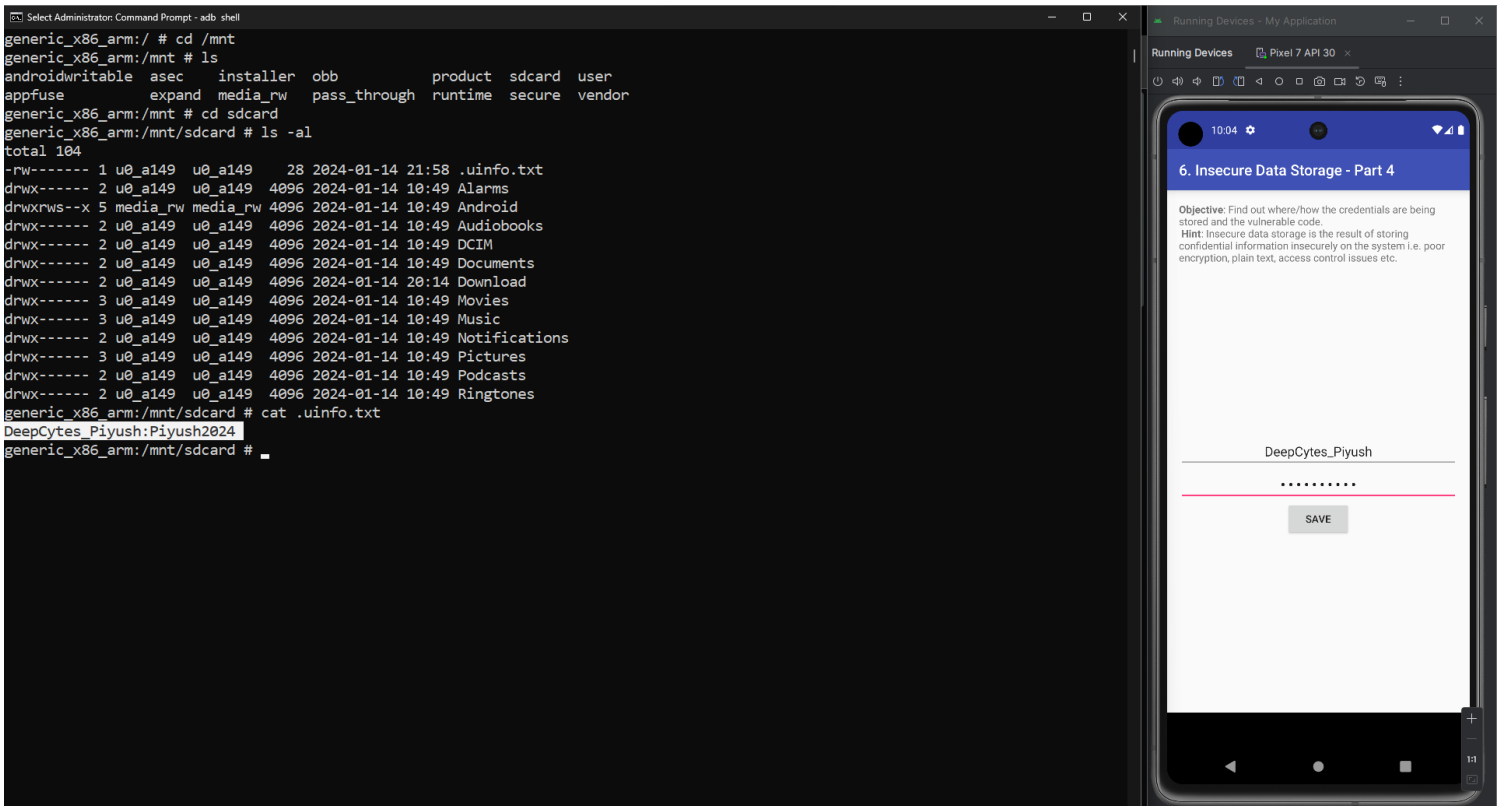


Now the file gets saved.

8.7 After Analysing the source code of *Jakhar.aseem.diva.InsecureDataStorage3Activity* we come to know that the data is getting stored in External Storage Directory and the file name will be “.uinfo.txt”.

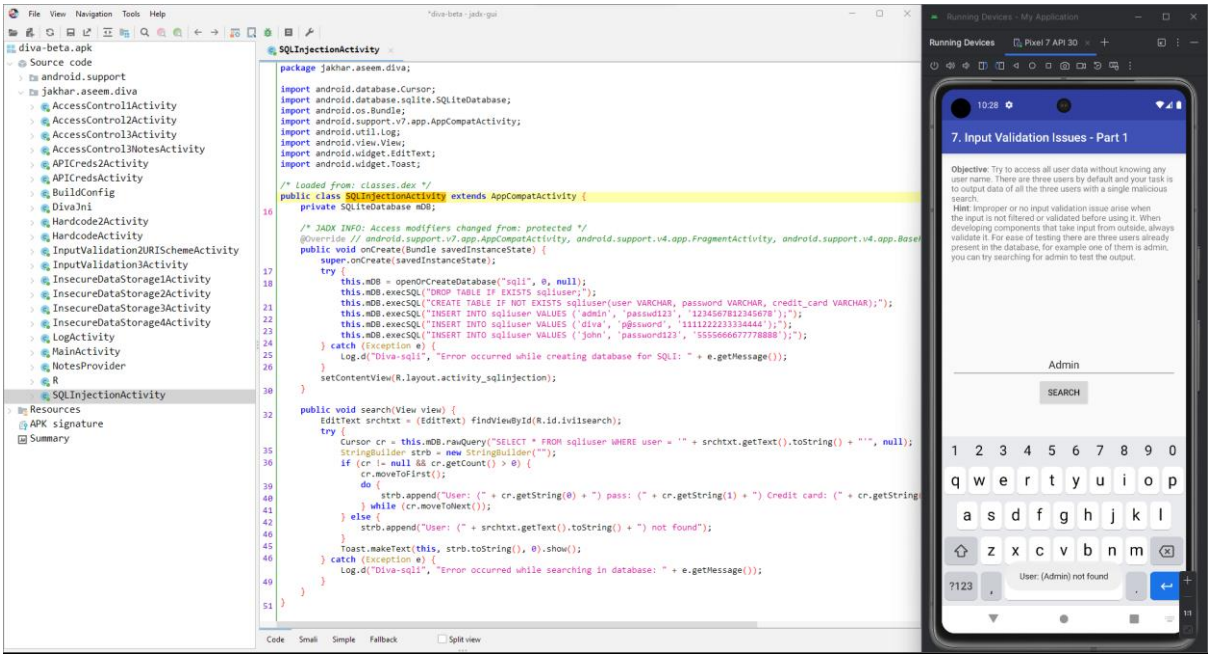
8.8 Here file is getting created as .unifo.txt this starting . means the file is hidden

8.9 Storage media always get mounted inside our MNT folder so we will use command cd /mnt

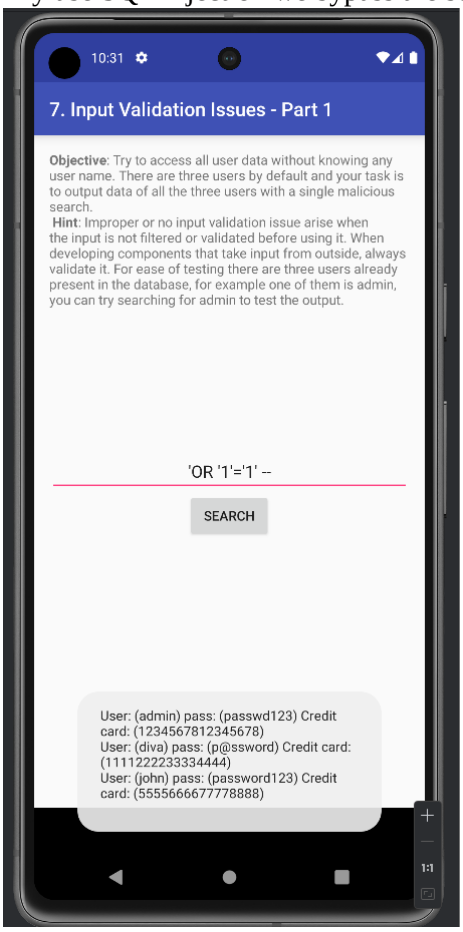


9. Input Validation Part 1:

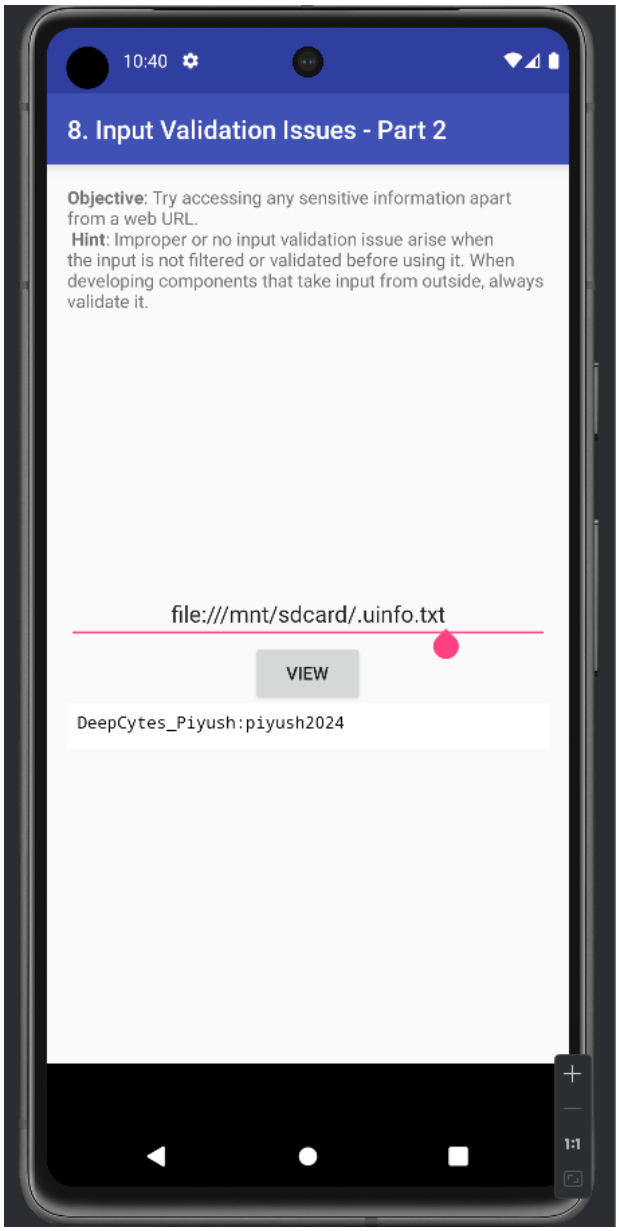
9.1 Using *Jadx-gui* to read the source code of the application, you will notice that this input validation is vulnerable to SQL injection. Also, you can see all three user credentials present in the database just by reading the source without digging deep, which is not a safe way to store credentials.



9.2 By use SQL injection we bypass the search validation and get all three credentials in the database.



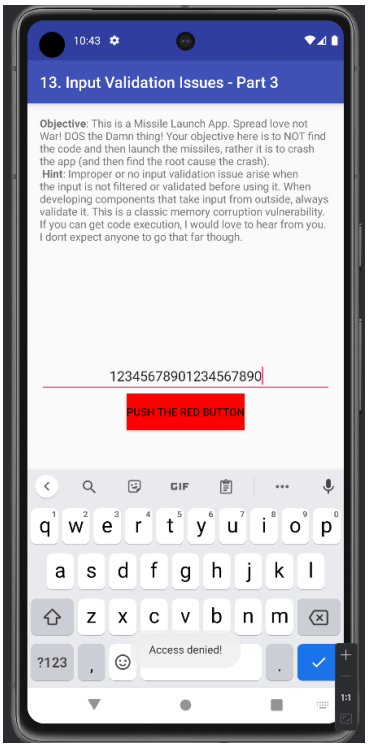
10. **Input Validation Issues Part 2:** This challenge is about accessing sensitive information using the file path of the sensitive information. Let's try the file path from the previous challenge ***(Insecure Data Storage Part 4)*.**



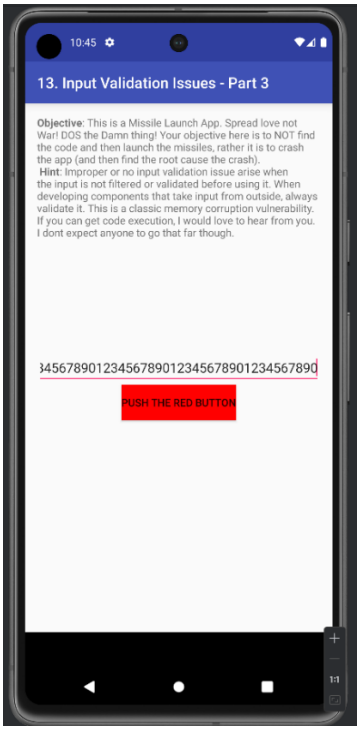
11. **Input Validation Issues Part 3:**

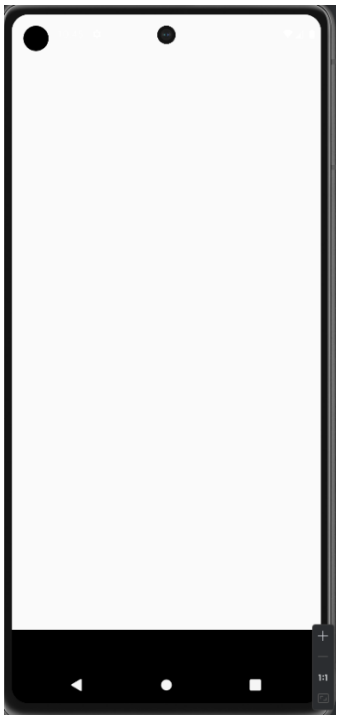
Here our objective is not to find the code rather it is to crash the application, to crash the application we generally follow the technique known as fuzzing in which we simply pass a lengthy or somewhat let's say random alphabetic character or some different language characters.

Application will only crash if there is no exception handling in the background, if the application was not able to handle the input properly only then our application will crash so to cause a denial of service attack on this application, we have to run repeated random numbers as many times as we like.



Access Denied, Now, let's increase the number of random numbers





Result: Application has Stopped working.