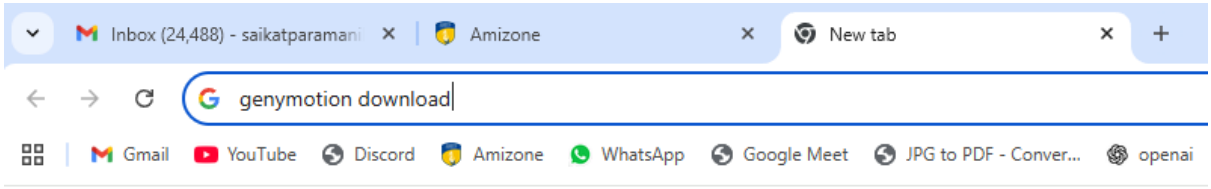
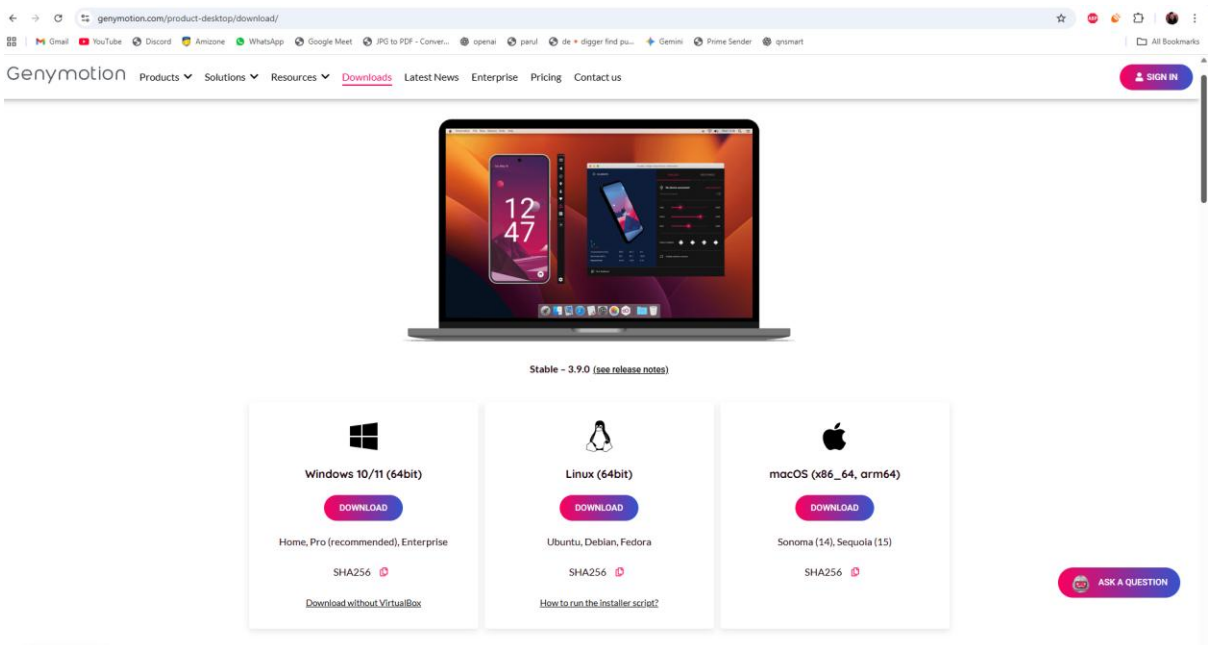
DIVA (Damn Insecure and Vulnerable App)



Download Genymotion

- Visit the official Genymotion download page: https://www.genymotion.com/product-desktop/download/
- Choose the correct installer based on your operating system (Windows, Linux, macOS).
- For Windows, download the **Genymotion 64-bit version** unl

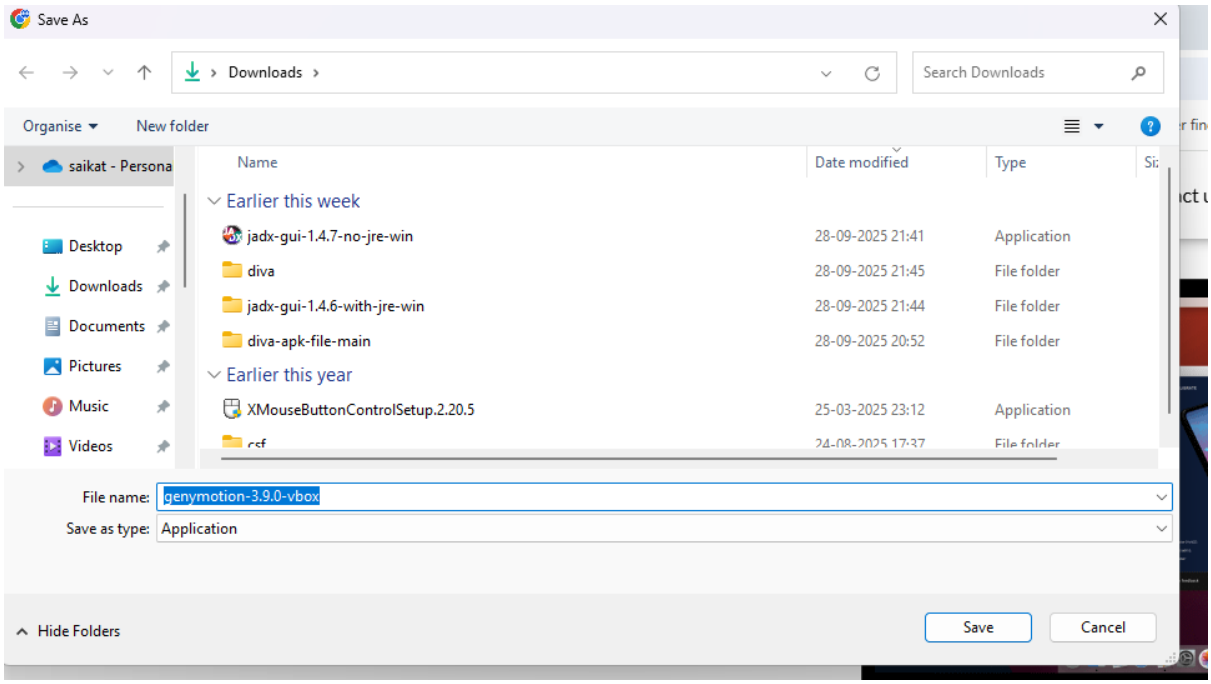


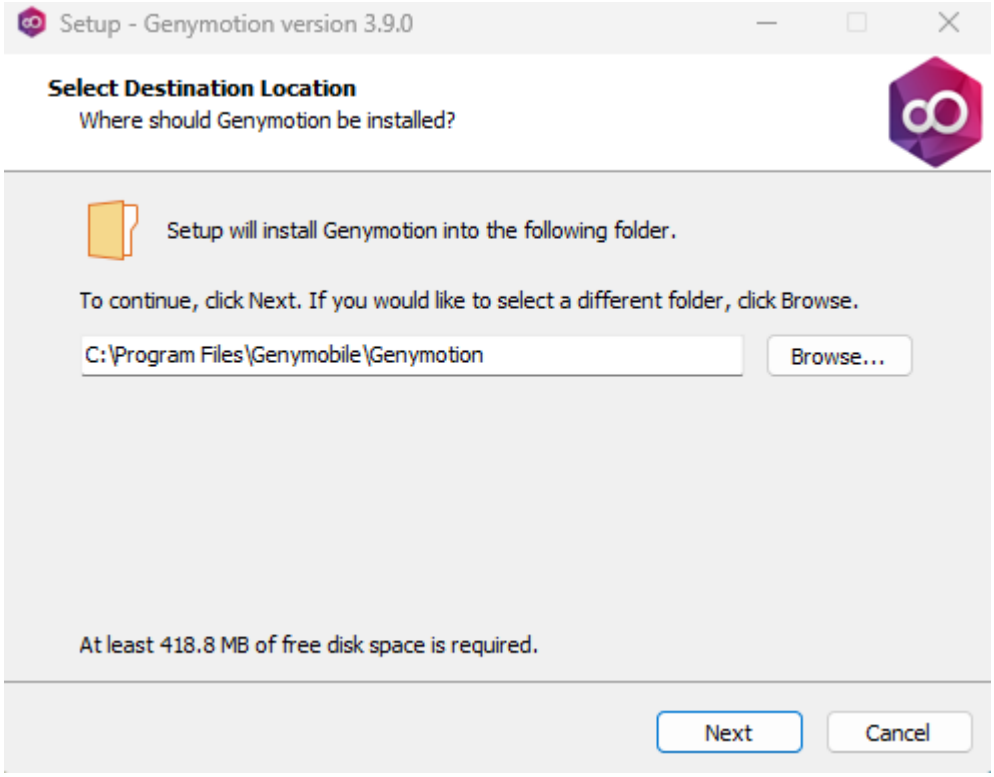
Install Genymotion

Run the downloaded installer (genymotion-3.9.0-vbox.exe).

Select the installation destination (default: C:\Program Files\Genymobile\Genymotion).

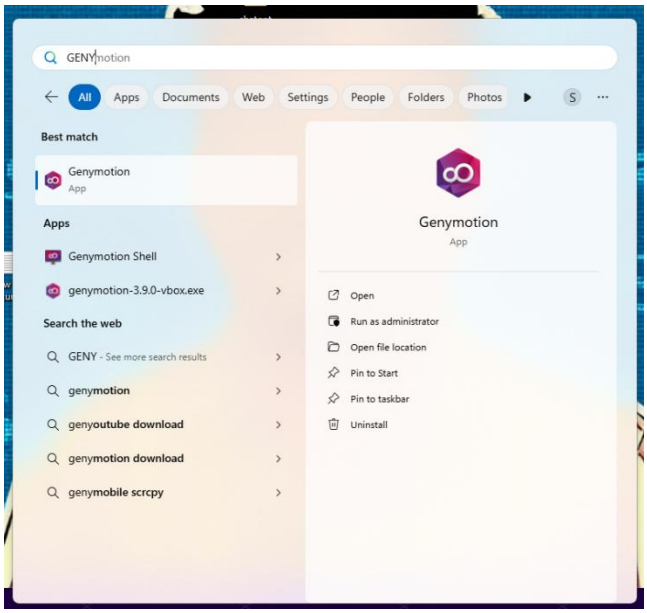
Click Next and follow the installation wizard until setup completes.





Launch Genymotion

- After installation, search for Genymotion in the Windows start menu.
- Click to launch the emulator.

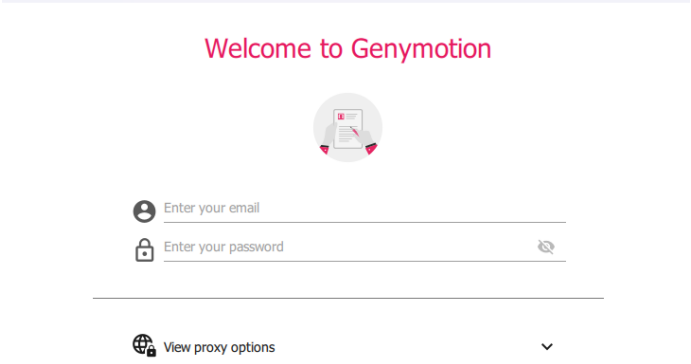


Genymotion First Launch & License Setup

When Genymotion is launched for the first time, it requires user login and license selection.

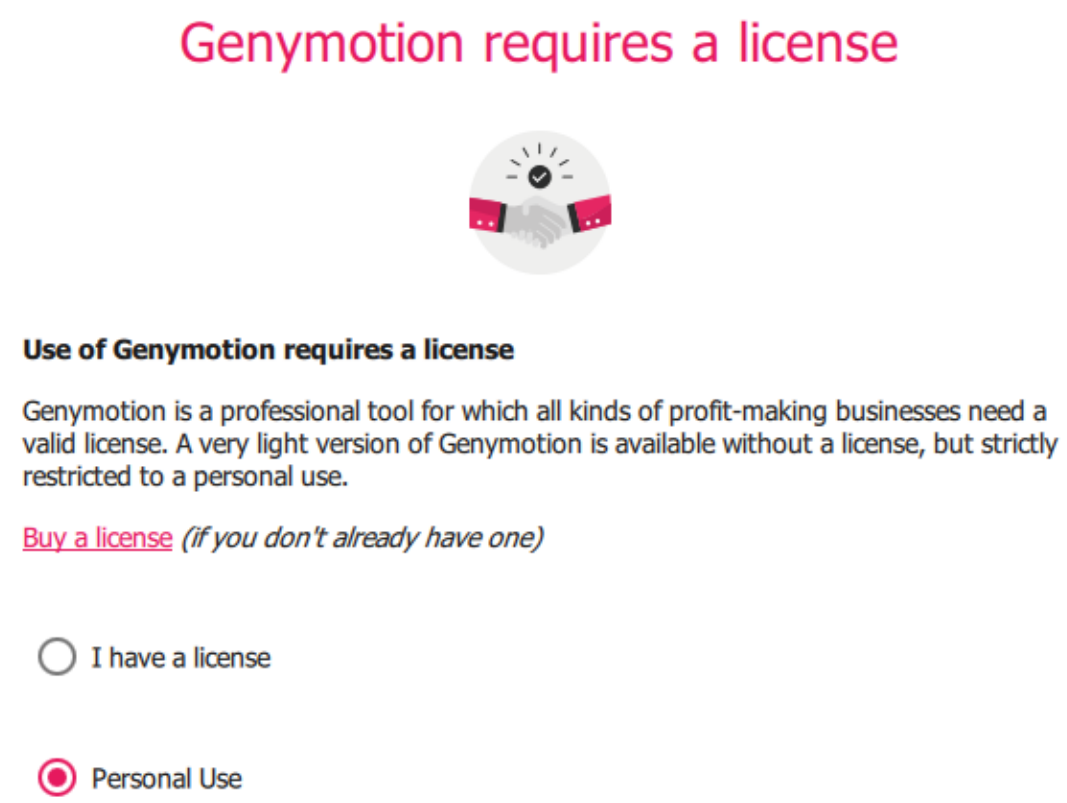
Step 4: Sign in to Genymotion

- You will be greeted with the Welcome to Genymotion screen.
- Enter your registered email and password to sign in. (If you don't have an account, create one on the Genymotion website.)



Select License Type

- Genymotion requires a license to use.
- For educational and practice purposes, choose Personal Use.
- Click Next to continue.

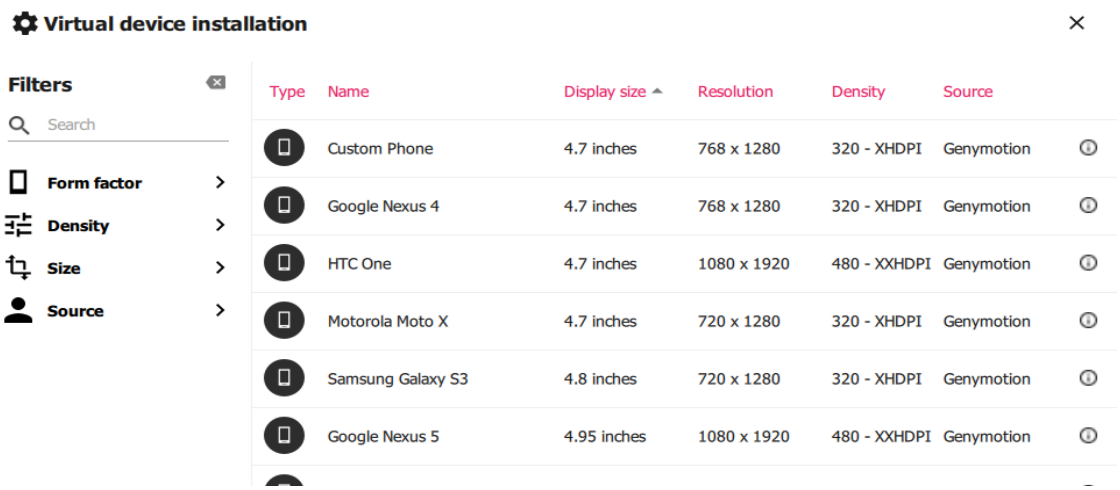


Creating and Configuring a Virtual Device

After logging in and selecting license type, you must create an Android virtual device inside Genymotion.

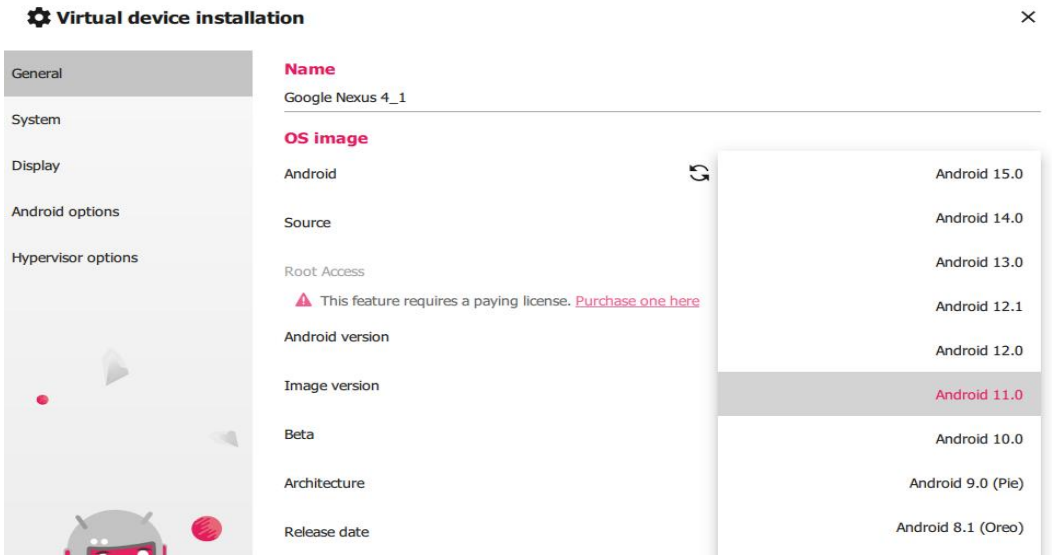
Step 6: Create a Virtual Device

- **From the main Genymotion window, click the Create button.**
- **A list of device templates appears (e.g., Google Nexus, HTC One, Motorola Moto X, etc.).**
- **Select a device template according to your preference (commonly Google Nexus 4/5).**



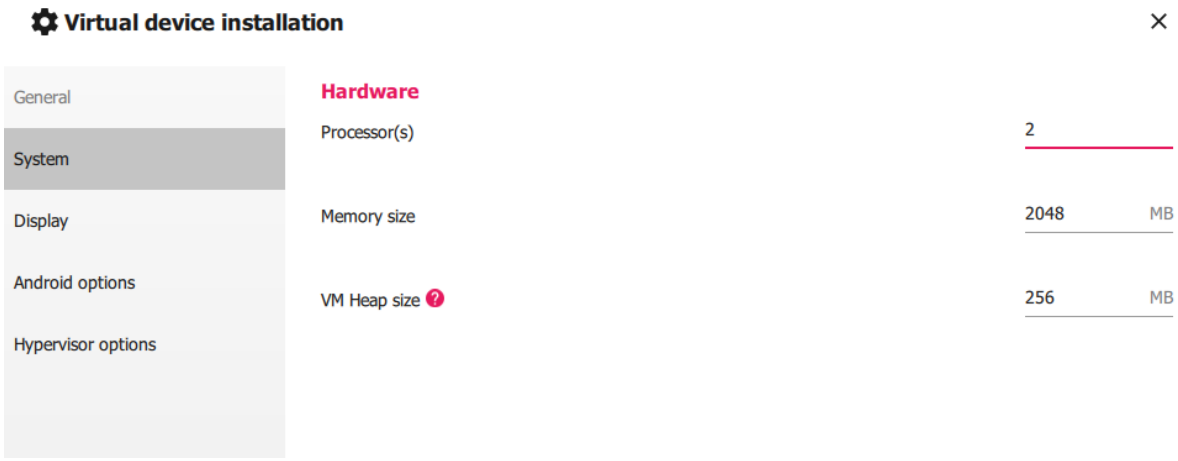
Select Android Version

- **After choosing a device, select the Android OS version.**
- **Example: Selecting Android 11.0 for a Google Nexus 4 virtual device.**



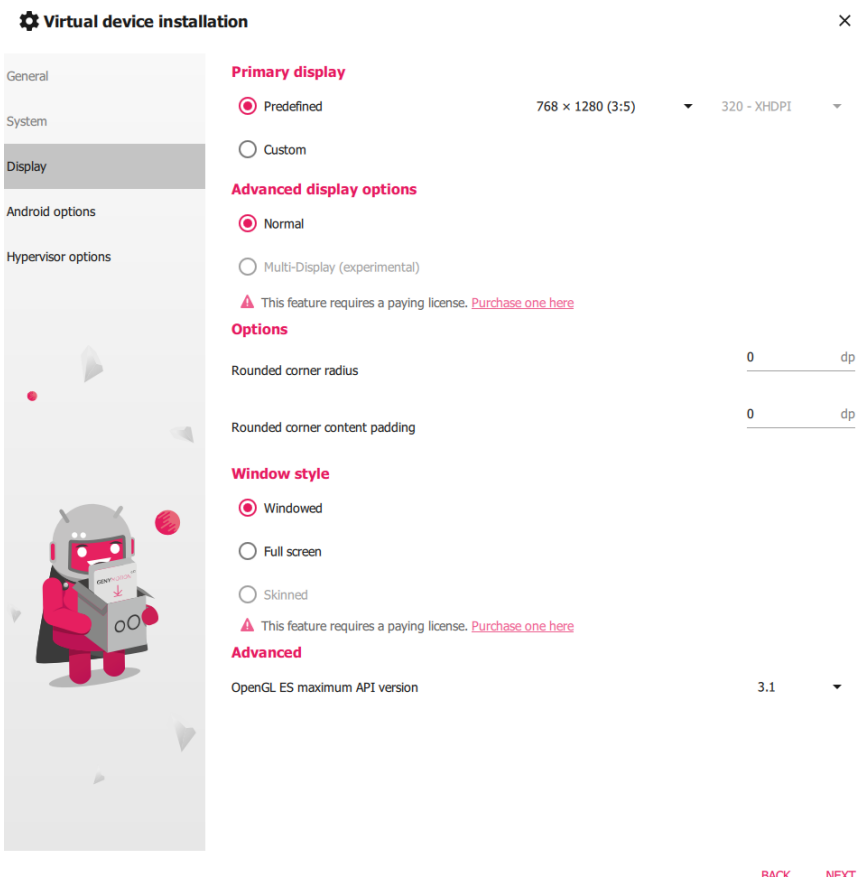
Configure System Resources

- **Assign system hardware resources to the virtual device:**
  - **Processor(s): 2**
  - **Memory size: 2048 MB**
  - **VM Heap size: 256 MB**
- **These can be adjusted depending on your system capabilities.**



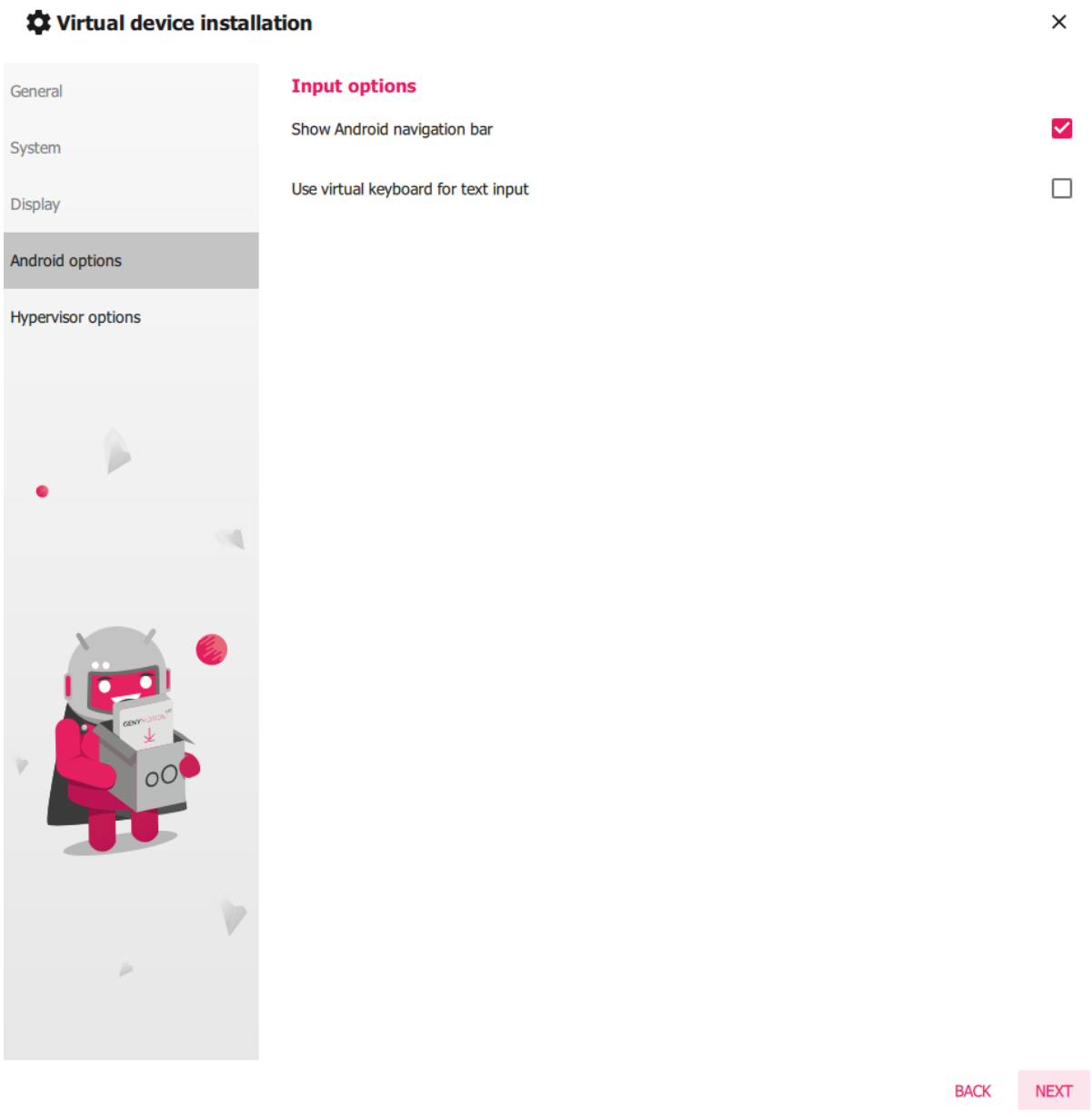
Configure Display Options

- Keep everything Default
- Click on next



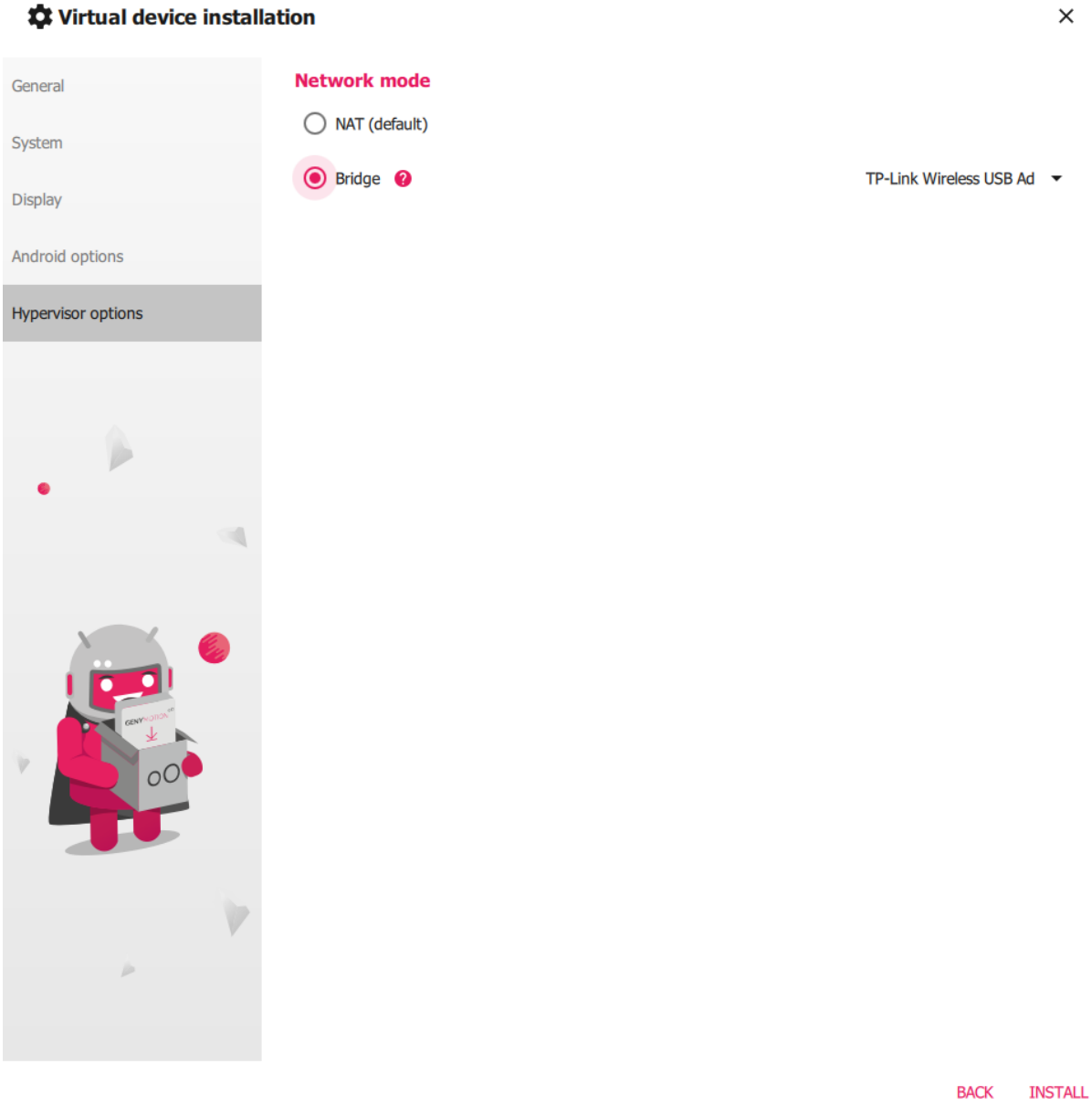
Configure Android Options

- **Select Android options such as:**
    - **Show Android navigation bar ✅**
    - **Use virtual keyboard (optional)**



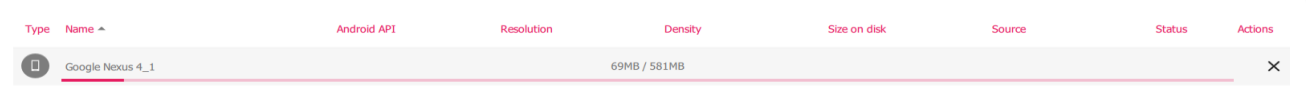
Configure Network (Hypervisor Options)

- **Select the network mode:**
    - **Click on Bridge**
    - **Click on next**



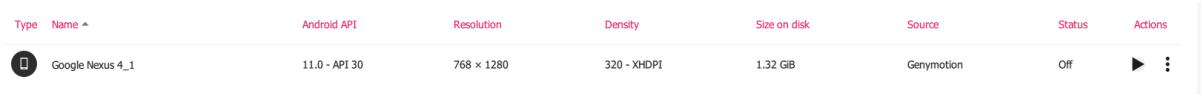
Install the Virtual Device

- **After completing configuration, click Install.**
- **The virtual device will download and install. Progress can be seen at the bottom.**
- **Once complete, the new virtual device will be listed in Genymotion.**

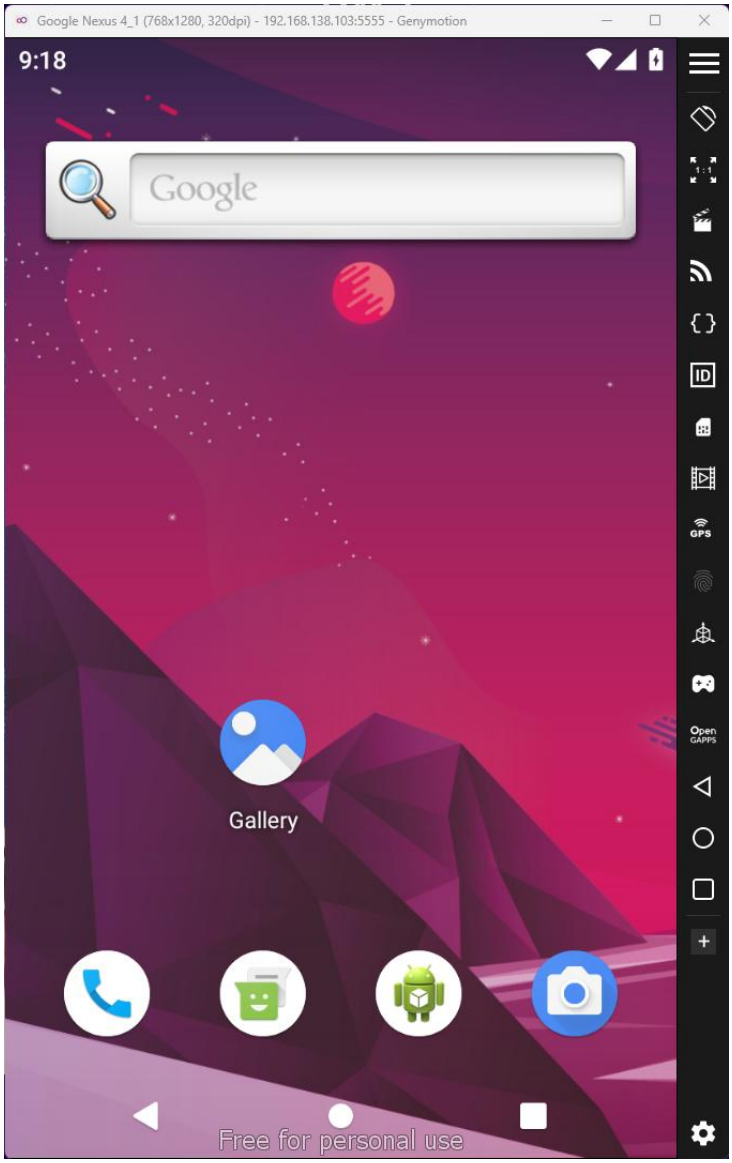


Run the Virtual Android

- **Click on play button under action**

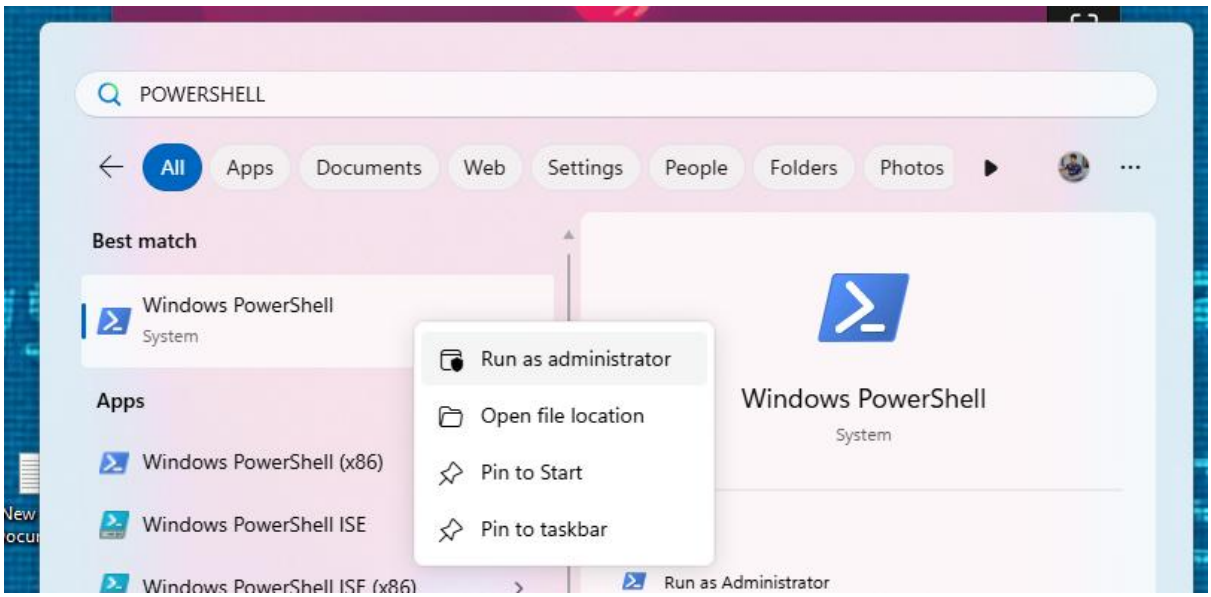


Android will open



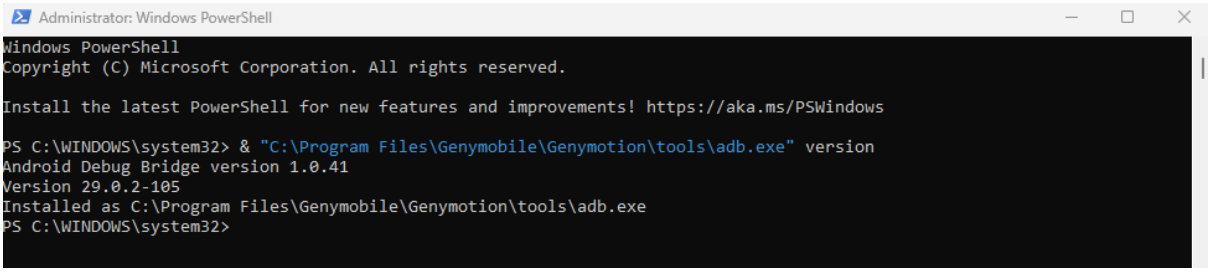
Open powershell

- **Run as administrator**

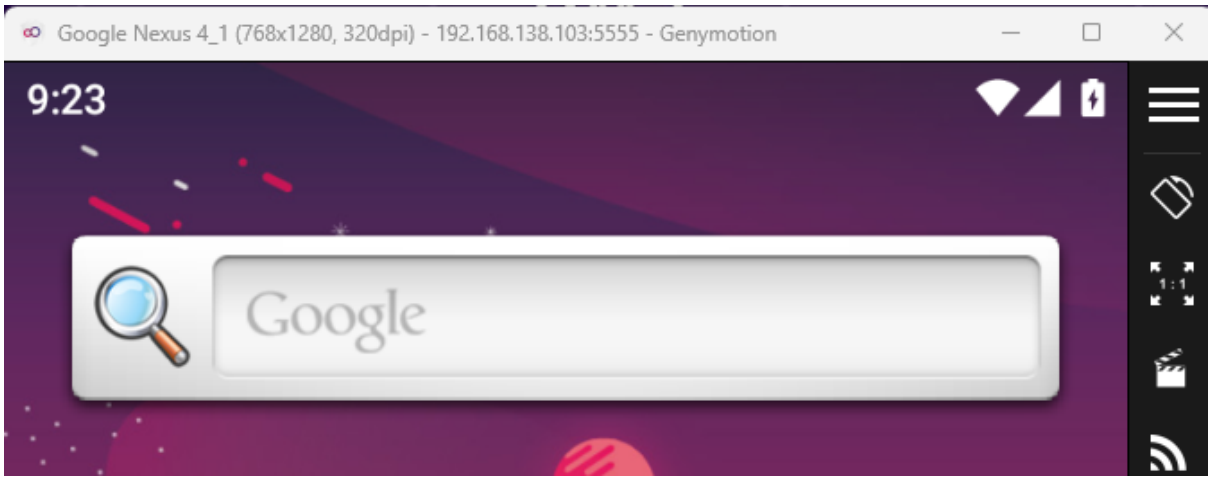


Verify ADB Installation

- **Run the following command to check ADB version:**
- & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" version



Note the ip of the android from the emulator title bar (eg : 192.168.138.103:5555)



Connect ADB to the Device

- Copy the device IP and port shown in the emulator title bar (e.g., 192.168.138.103:5555).

Run: & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" connect 192.168.138.103:5555

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" connect 192.168.138.103:5555
already connected to 192.168.138.103:5555
PS C:\WINDOWS\system32>
```

Verify Device Connection

- **Run:** & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" devices

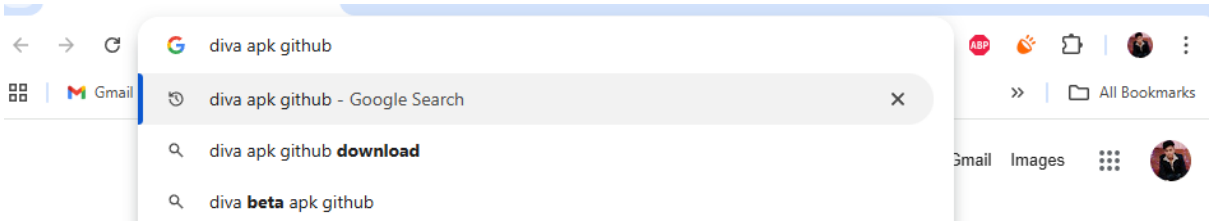
```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" devices
List of devices attached
192.168.138.103:5555    device
```

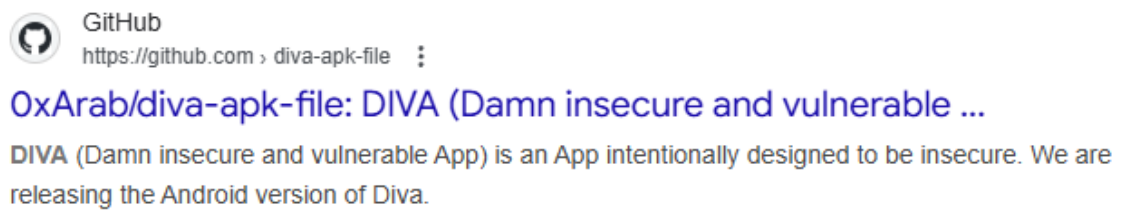
Downloading and Extracting the DIVA APK

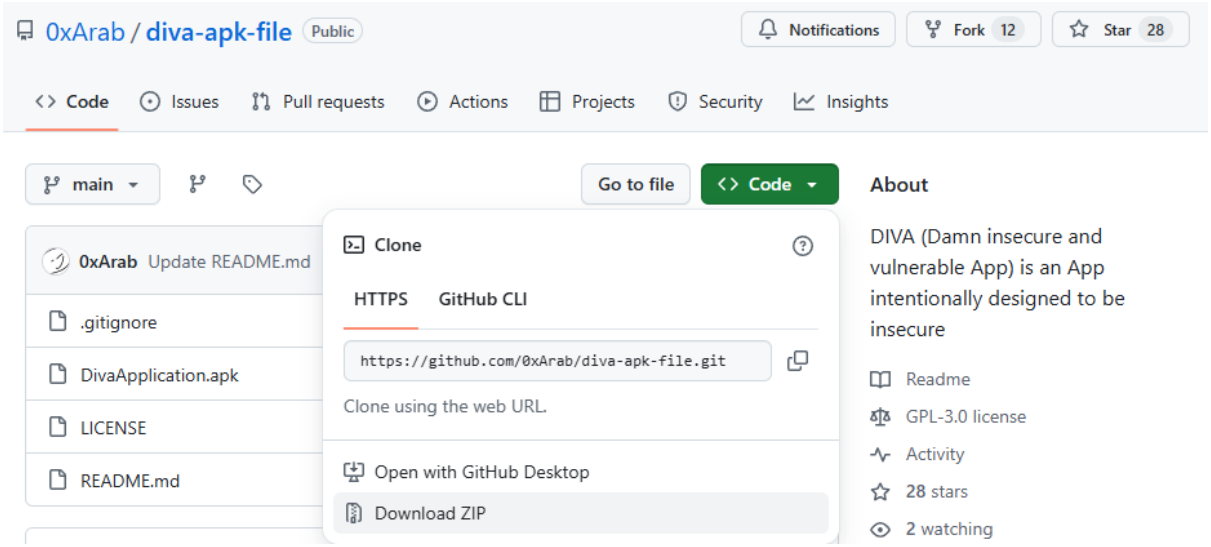
Next, we need to download the vulnerable APK file for DIVA.

Search and Download DIVA APK

- Open a web browser and search for diva apk github.
- Click on the GitHub repository: *0xArab/diva-apk-file*.
- On the repository page, click Download ZIP.

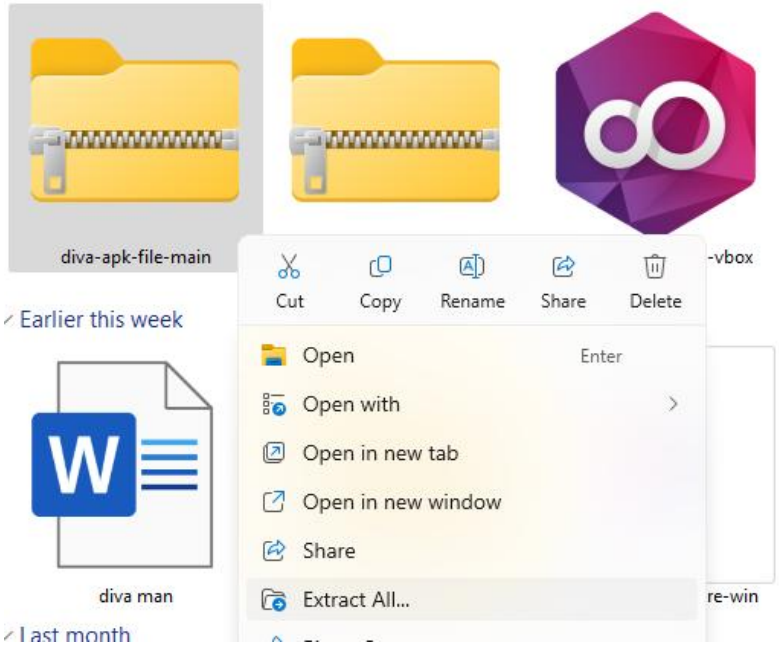






Extract the ZIP File

- Go to the Downloads folder and locate the file diva-apk-file-main.zip.
- Right-click on it and select Extract All.
- This will create a folder containing the DIVA APK file that we can later install on the emulator.

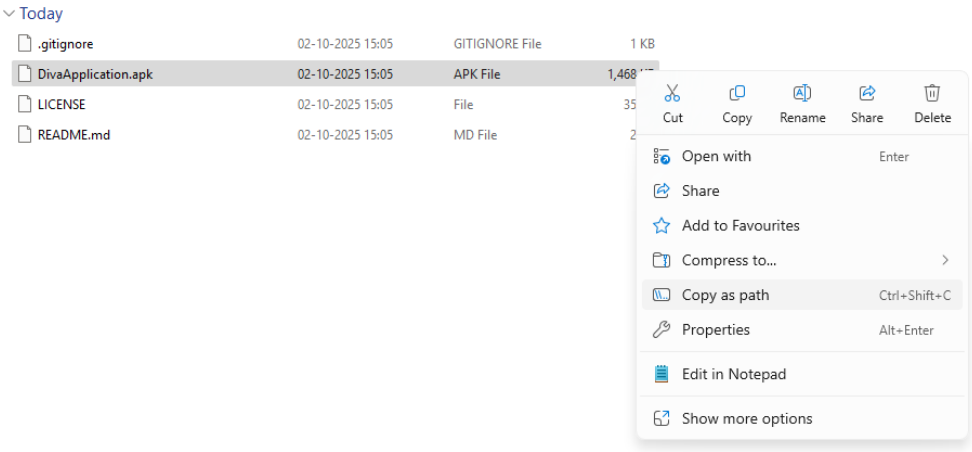


Installing the DIVA APK on Emulator

Once the APK file is extracted, we install it into the Genymotion virtual device using ADB.

Step 20: Locate the APK File

- Inside the extracted folder, locate DivaApplication.apk.
- Right-click on the file and select Copy as path to copy its full file location.



Install APK using ADB

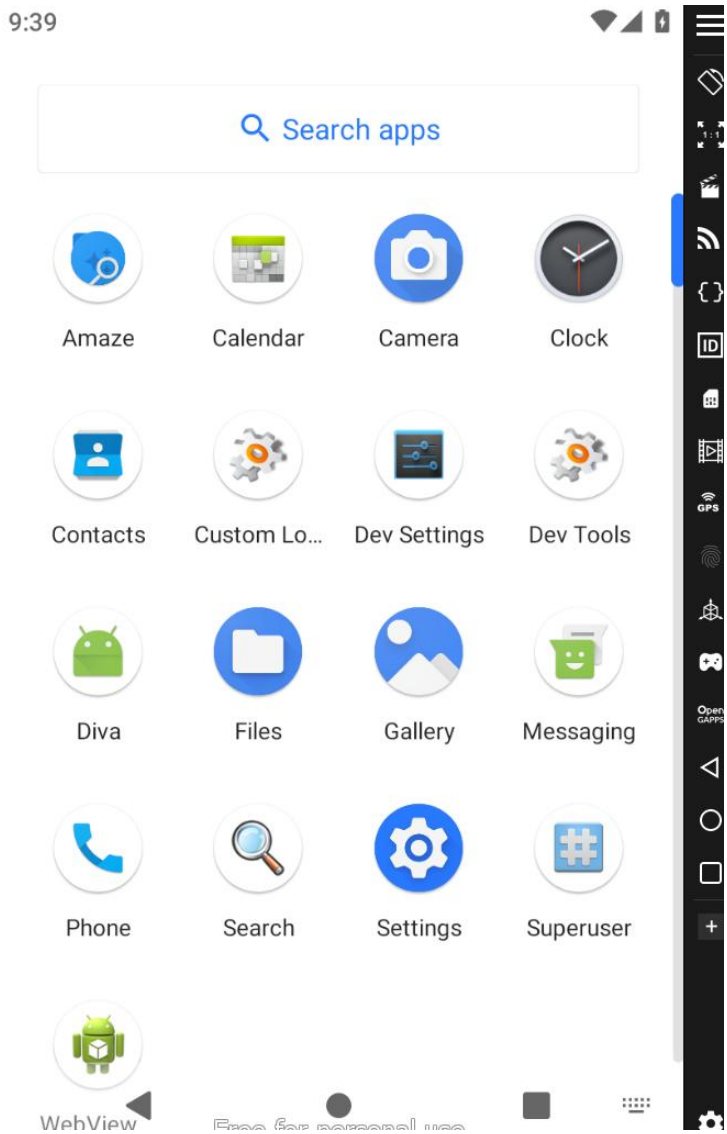
- Open PowerShell as Administrator.
- Use the following command format to install:

  Run: & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" install "C:\Users\saika\Downloads\diva-apk-file-main\diva-apk-file-main\DivaApplication.apk"

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" install
"C:\Users\saika\Downloads\diva-apk-file-main\diva-apk-file-main\DivaApplication.apk"
Performing Streamed Install
Success
PS C:\WINDOWS\system32>
```

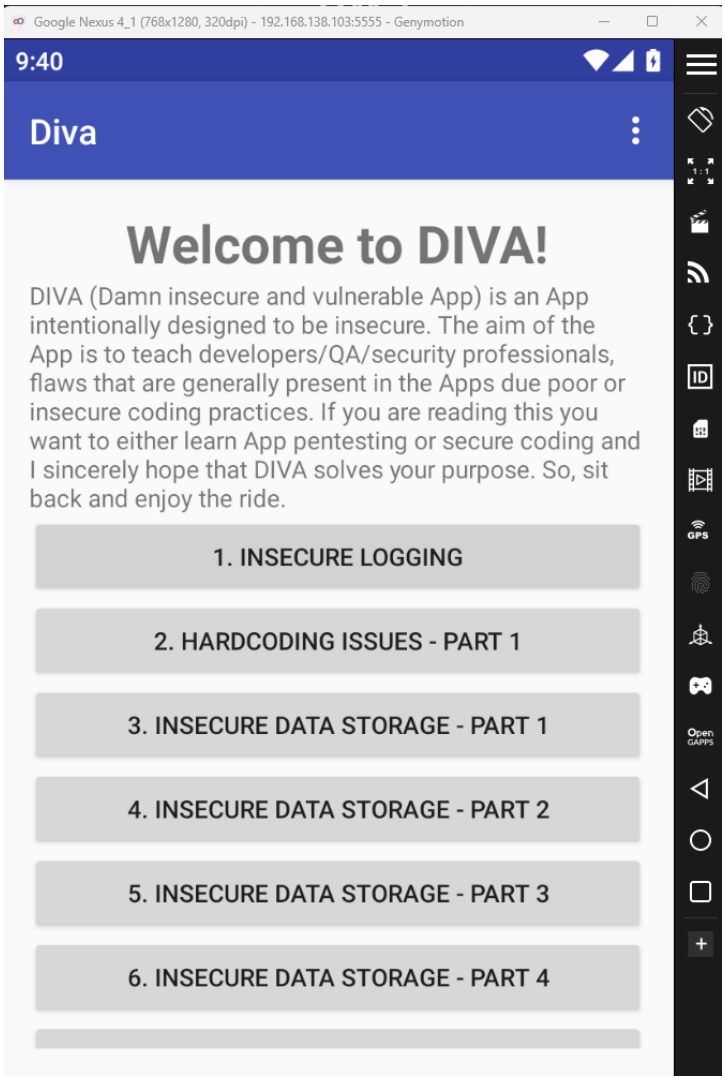
Verify Installation

- **Open the app drawer inside the emulator.**
- **You should see the Diva app icon.**



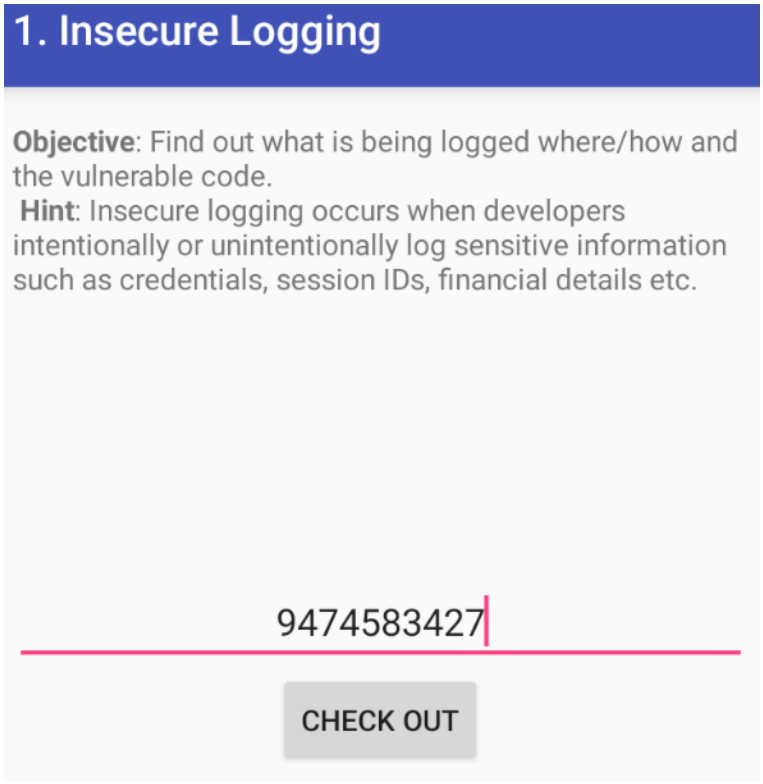
Open DIVA App

- Tap on the Diva icon.
- The welcome screen of DIVA will appear, showing a list of vulnerable challenges.



Open Insecure Logging Challenge

- From the DIVA app home screen, tap 1. Insecure Logging.
- You will see an input field where you can enter a value (e.g., a credit card number or ID) and press CHECK OUT.



Clear Previous Logs

- Before testing, clear the existing logcat logs to avoid confusion.

  Run: & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" logcat -c

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" logcat -c
PS C:\WINDOWS\system32>
```

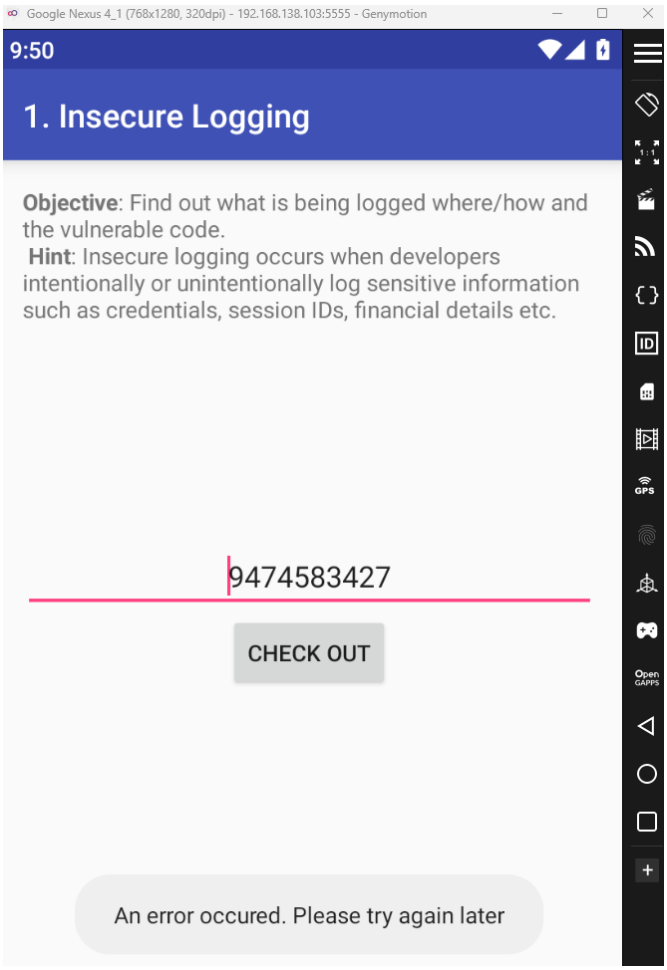
Monitor Logs

- Start monitoring logs in real-time:
- Run: & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" logcat

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" logcat
--------- beginning of main
10-02 09:48:57.905   634   737 D InputDispatcher: Waiting to send key to Window{9cb40ea u0 jakhar.aseem.diva/jakhar.aseem.diva.LogActivity} because there are unprocessed even
ts that may cause focus to change
--------- beginning of system
10-02 09:48:58.384   634   634 W WindowManager: removeWindowToken: Attempted to remove non-existing token: android.os.Binder@66725f1
10-02 09:48:59.400   634   737 D InputDispatcher: Waiting to send key to Window{9cb40ea u0 jakhar.aseem.diva/jakhar.aseem.diva.LogActivity} because there are unprocessed even
ts that may cause focus to change
10-02 09:49:00.005   861   861 D KeyguardClockSwitch: Updating clock: 9␂49
--------- beginning of kernel
10-02 09:49:00.748     0     0 D logd    : Skipping 18446744073709551471 entries from slow reader, pid 674, from LogBuffer::kickMe()
10-02 09:49:05.191     0     0 D logd    : logdr: UID=0 GID=0 PID=2913 b tail=0 logMask=99 pid=0 start=0ns timeout=0ns
```

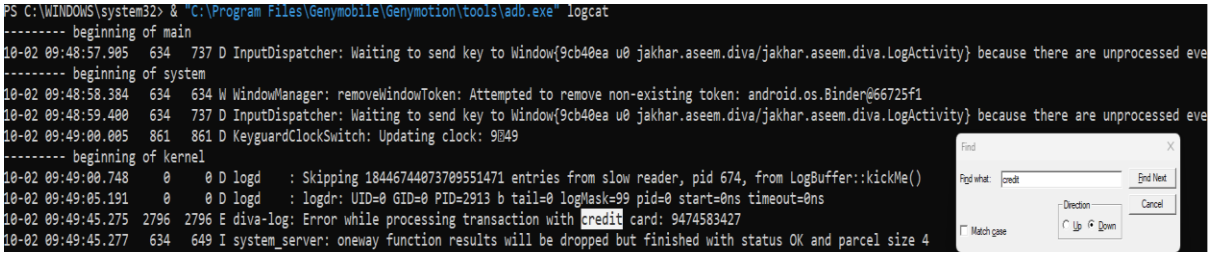
Trigger Logging Vulnerability

- Go back to the DIVA Insecure Logging screen.
- Enter a value (e.g., 9474583427) and press CHECK OUT.
- The app may show an error message on screen (e.g., *An error occurred. Please try again later*).



Analyze Logs

- Check the PowerShell logcat output , Press ctrl+f and type credit to find credit card details.
- You will find sensitive input (like the number you entered) logged in plain text, often with debug/error messages.
- Example log snippet:

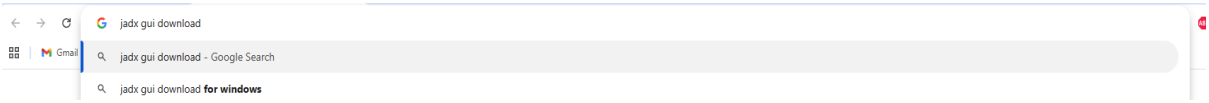


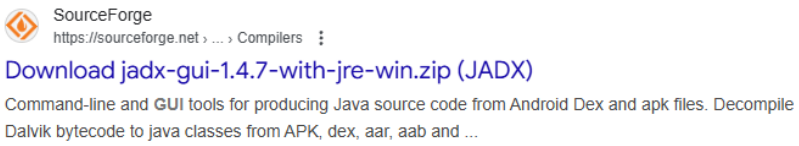
Setting up JADX for Static Analysis

JADX is a tool used to decompile APK files and view the source code. We will use it to analyze DIVA's vulnerable code.

Download JADX

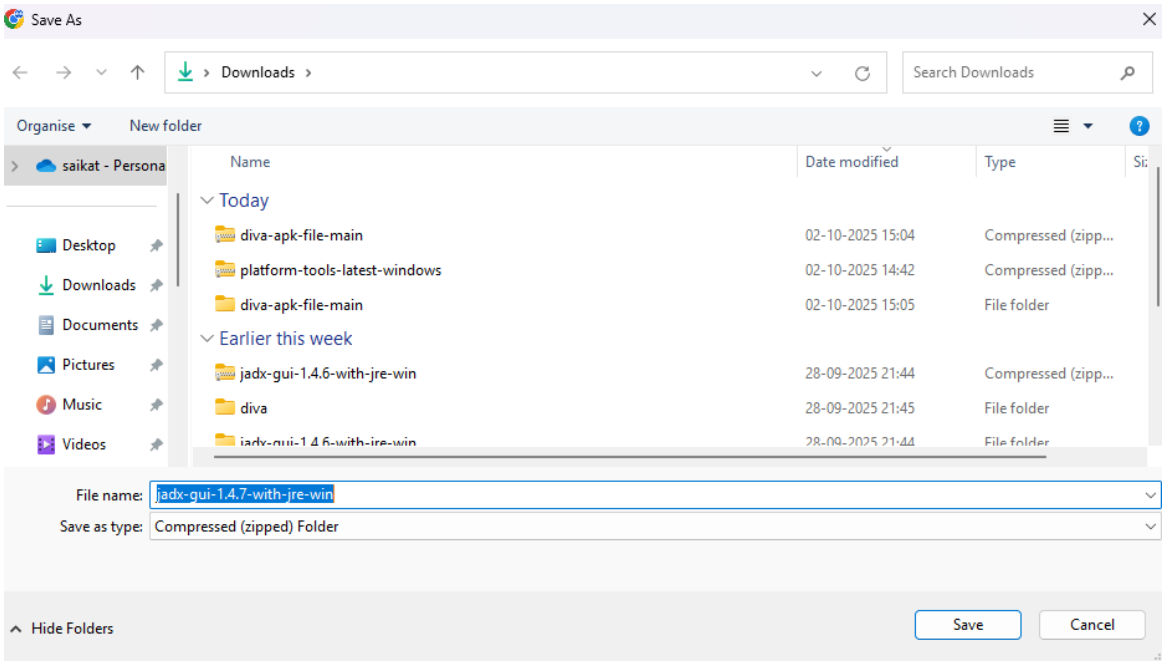
- Open a web browser and search for jadx gui download.
- Select the SourceForge link to download the latest version (e.g., jadx-gui-1.4.7-with-jre-win.zip).





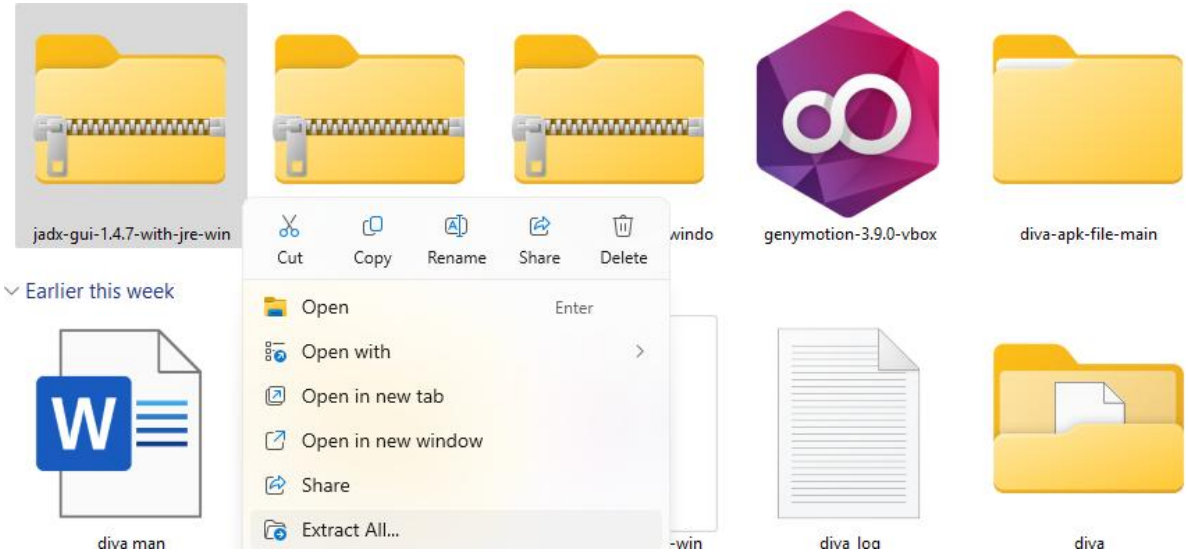
Save the JADX File

- Choose the Downloads folder as the save location.
- Save the compressed ZIP file.



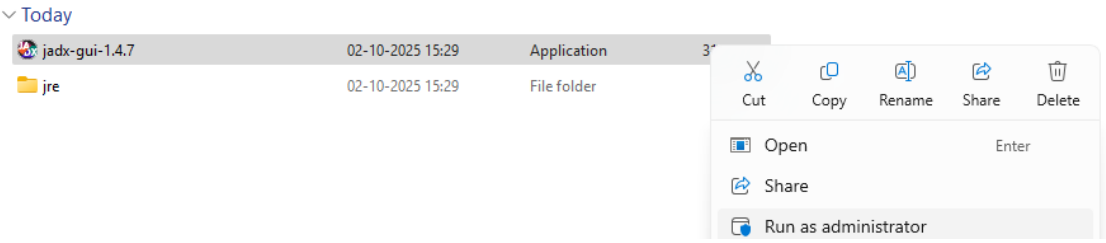
Extract the JADX ZIP

- Right-click on the downloaded file and choose Extract All.
- This creates a folder containing the JADX executable.



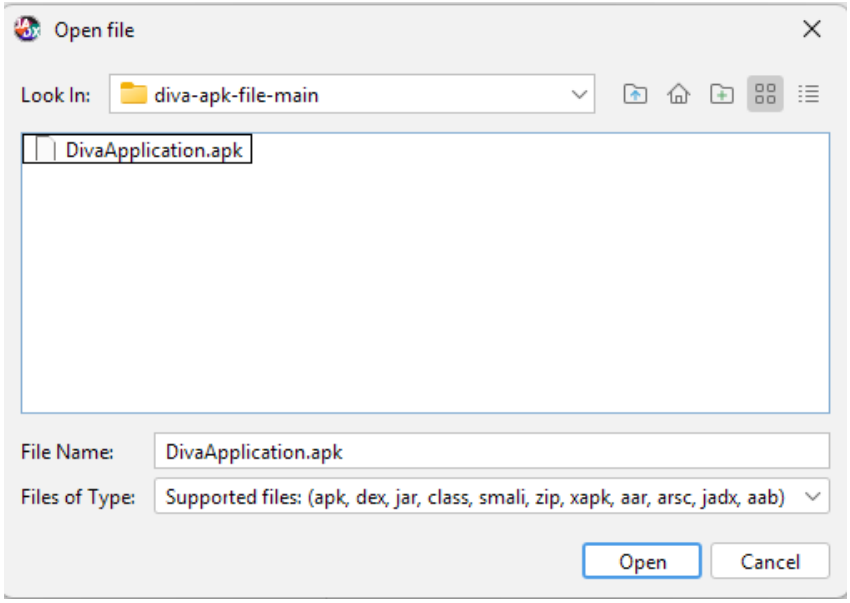
Run JADX

- Open the extracted folder and locate jadx-gui-<version> (e.g., jadx-gui-1.4.7).
- Right-click and select Run as Administrator.

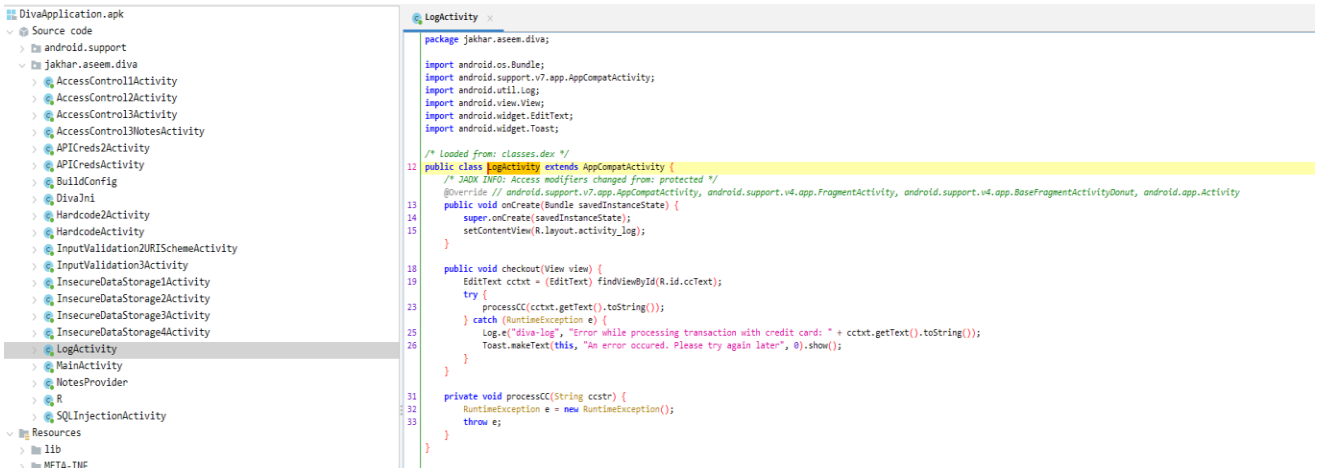


Load DIVA APK in JADX

- In JADX, go to File > Open File.
- Browse to the folder where the DIVA APK (DivaApplication.apk) is stored.
- Select the APK and click Open.



Static analysis of code why Insecure Login in DIVA is vulnerable

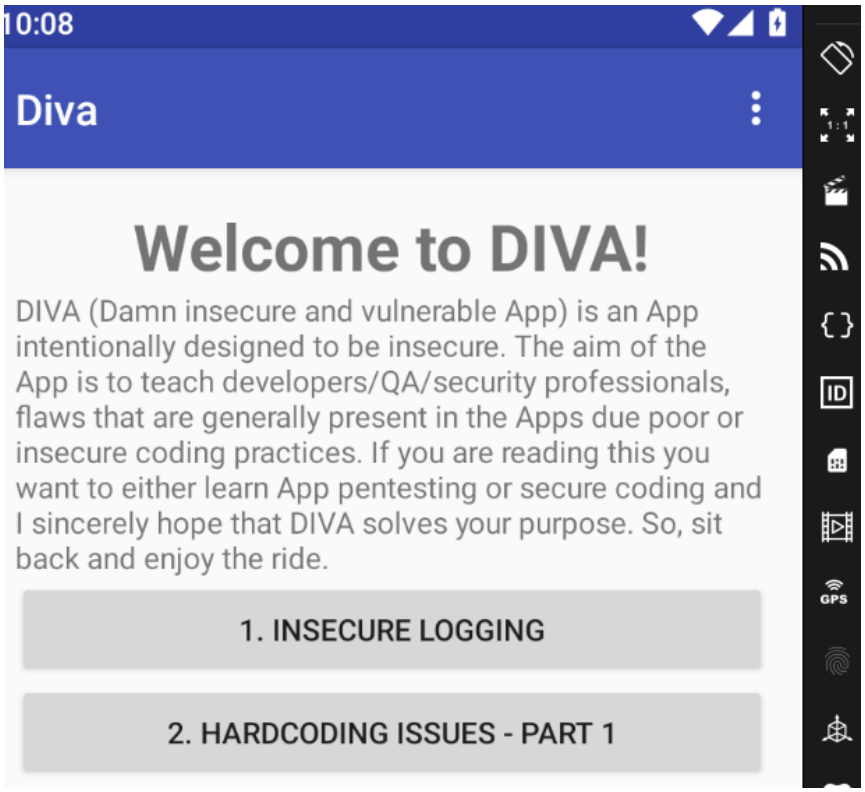


The code

Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toString()); is insecure because it logs sensitive data (credit card number) directly into system logs. Logs can be accessed by attackers (via adb logcat, rooted devices, or other apps), which leads to data leakage and privacy violations.

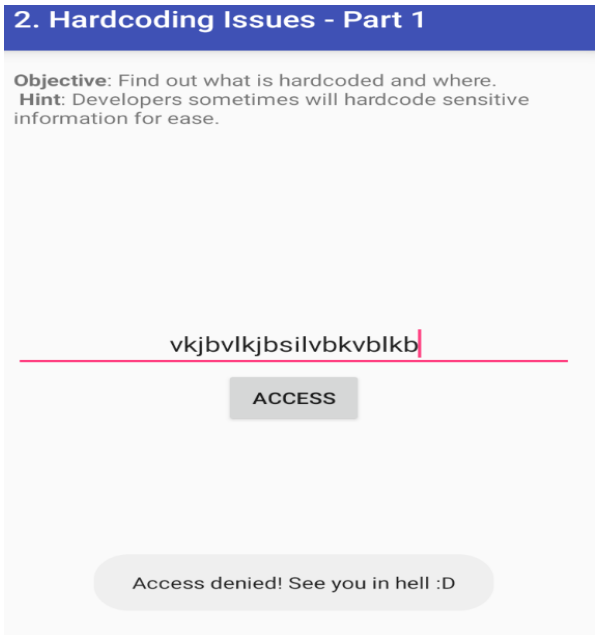
Hardcoding Issues – Part 1

This option demonstrates how developers sometimes hardcode sensitive information inside the source code. The goal is to identify and extract it.



Entering random input – Access Denied

When we try random values in the app, it shows *Access Denied*, confirming that the app only accepts a specific secret key.



Code view in JADX – HardcodeActivity.java

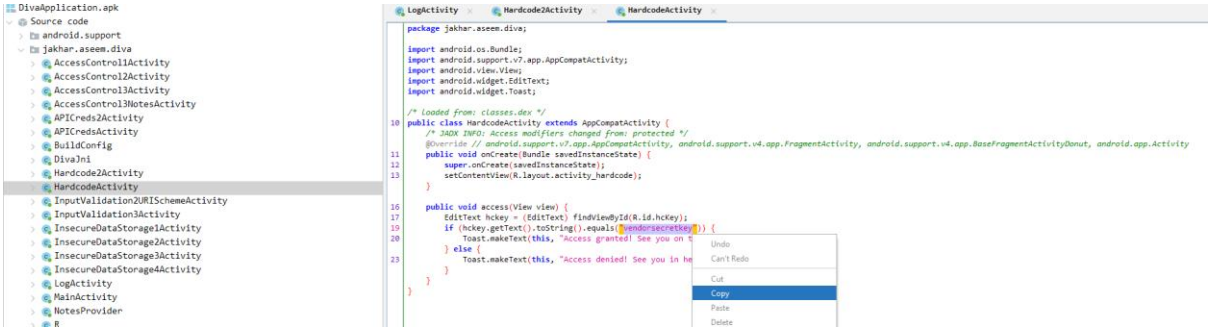
By decompiling the app using JADX, we find in HardcodeActivity.java that the secret key is hardcoded as:

if (key.getText().toString().equals("vendorsecretkey")) {

Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();

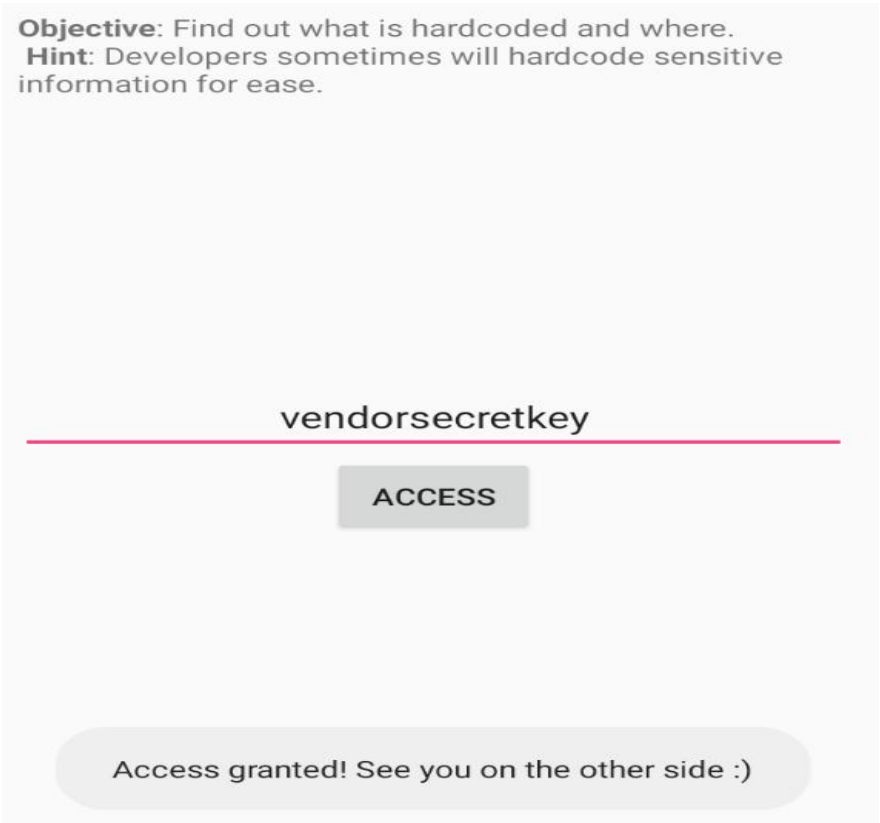
}

This means the secret value vendorsecretkey is directly written in the code.



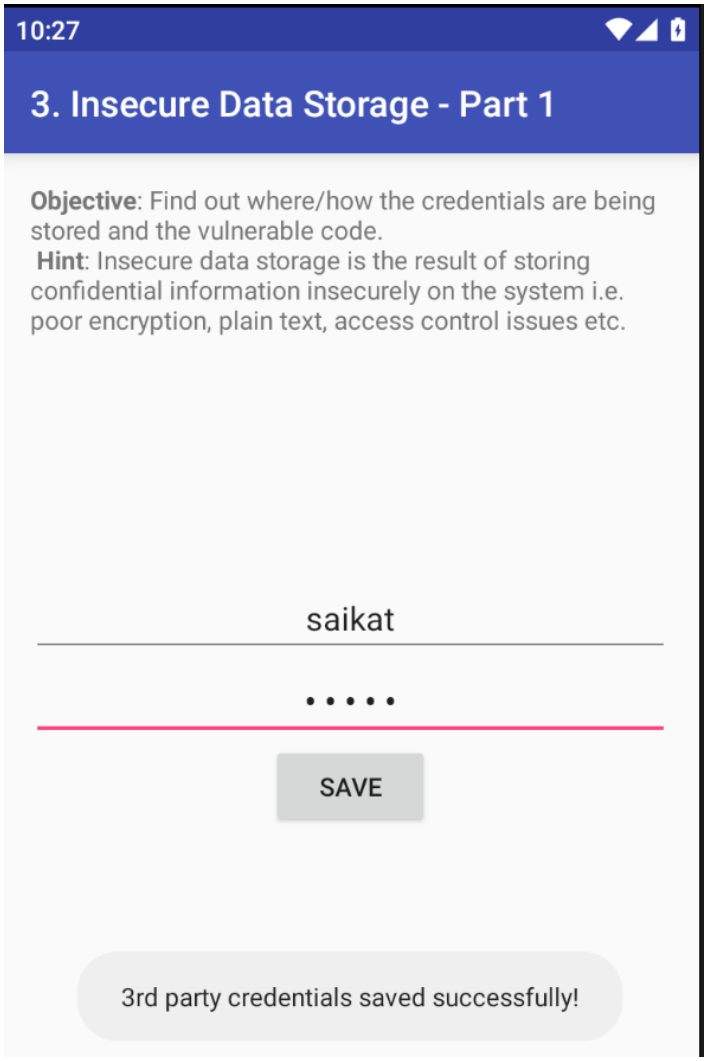
Entering vendorsecretkey – Access Granted

After discovering the hardcoded key in the code, entering vendorsecretkey in the app grants access successfully. This confirms the vulnerability.



Insecure Data Storage – Part 1

In this challenge, we enter a username (saikat) and password (12345) in the DIVA app. The app shows a message: *"3rd party credentials saved successfully!"* indicating the credentials were stored on the device.



To locate where the app stores data, we first confirm the package name using:

Run: adb shell pm list packages | findstr diva

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell pm list packages | findstr diva
package:jakhar.aseem.diva
```

Navigating to shared_prefs with ADB

Run one by one:

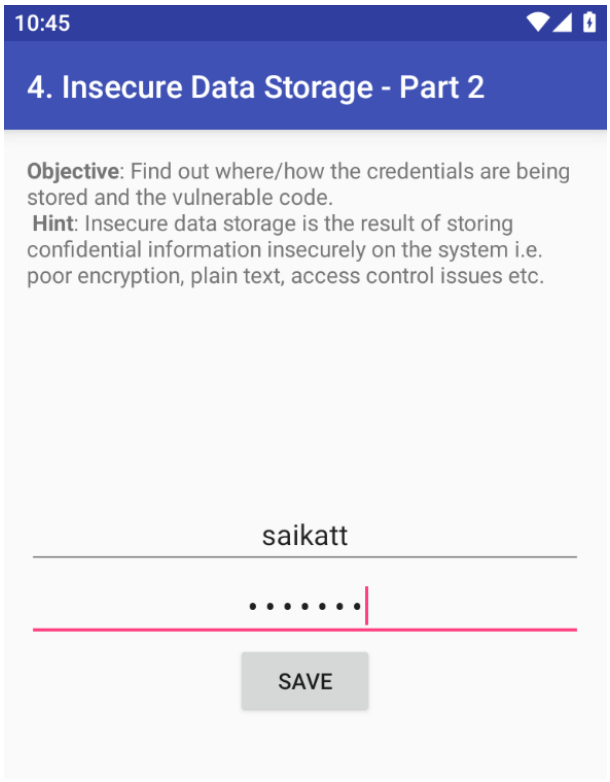
1. & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell
2. run-as jakhar.aseem.diva
3. cd /data/data/jakhar.aseem.diva/shared_prefs
4. ls -l
5. cat *.xml

```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell
vbox86p:/ # run-as jakhar.aseem.diva
vbox86p:/data/user/0/jakhar.aseem.diva $ cd /data/data/jakhar.aseem.diva/shared_prefs
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs $ ls -l
total 8
-rw-rw---- 1 u0_a129 u0_a129 153 2025-10-02 10:27 jakhar.aseem.diva_preferences.xml
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs $ cat *.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">12345</string>
    <string name="user">saikat</string>
</map>
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs $
```

This proves the credentials are saved in plain text. The app is vulnerable because it stores sensitive credentials in clear text. An attacker with device access could easily steal this data. The secure fix is to use encrypted storage or the Android Keystore system.

Insecure Data Storage – Part 2

In this challenge, we input username (saikatt) and password. The app confirms the credentials are saved. Our task is to check *where* and *how* they are stored on the device.



ADB shell – locating database storage (Run one by one)

1. & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell
2. run-as jakhar.aseem.diva
3. ls -la
4. cd databases
5. ls -la
6. sqlite3 ids2
7. .tables
8. Select * FROM myusers;

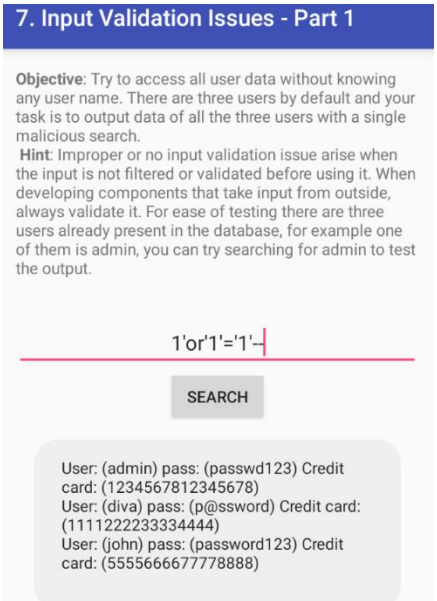
```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell
vbox86p:/ # run-as jakhar.aseem.diva
vbox86p:/data/user/0/jakhar.aseem.diva $ ls -la
total 56
drwxr-x--x   6 u0_a129 u0_a129        4096 2025-10-02 10:27 .
drwxrwx--x 167 system  system        12288 2025-10-02 09:38 ..
drwxrws--x   2 u0_a129 u0_a129_cache  4096 2025-10-02 09:38 cache
drwxrws--x   2 u0_a129 u0_a129_cache  4096 2025-10-02 09:38 code_cache
drwxrwx--x   2 u0_a129 u0_a129        4096 2025-10-02 10:42 databases
drwxrwx--x   2 u0_a129 u0_a129        4096 2025-10-02 10:27 shared_prefs
vbox86p:/data/user/0/jakhar.aseem.diva $ cd databases
vbox86p:/data/user/0/jakhar.aseem.diva/databases $ ls -la
total 68
drwxrwx--x 2 u0_a129 u0_a129  4096 2025-10-02 10:42 .
drwxr-x--x 6 u0_a129 u0_a129  4096 2025-10-02 10:27 ..
-rw-rw---- 1 u0_a129 u0_a129 20480 2025-10-02 09:40 divanotes.db
-rw-rw---- 1 u0_a129 u0_a129     0 2025-10-02 09:40 divanotes.db-journal
-rw-rw---- 1 u0_a129 u0_a129 16384 2025-10-02 10:42 ids2
-rw-rw---- 1 u0_a129 u0_a129     0 2025-10-02 10:42 ids2-journal
vbox86p:/data/user/0/jakhar.aseem.diva/databases $ sqlite3 ids2
SQLite version 3.28.0 2020-05-06 18:46:38
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  myuser
sqlite> select * FROM myuser;
saikatt|1234555
```

By opening the database with SQLite (using commands like sqlite3), we can see that the username and password are stored in **plain text** inside the database.

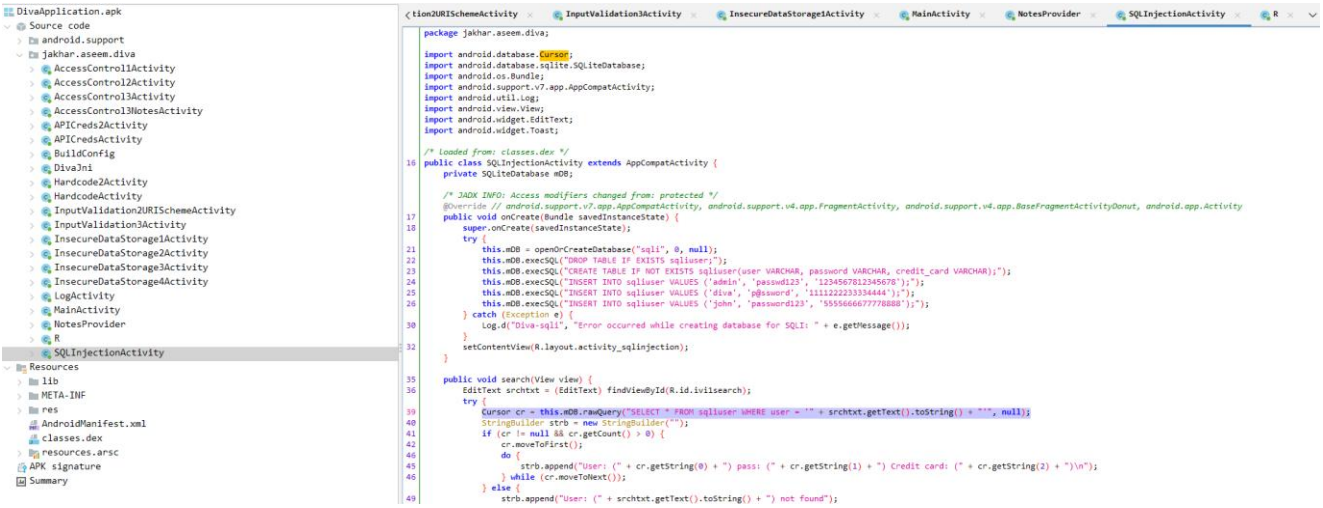
Input Validation Issue Part- 1

The app takes the text you type and directly concatenates it into an SQL query. Because there is no sanitization or parameterization, a malicious input can change the meaning of the SQL statement and return all records instead of just the intended one. This is classic SQL injection.

Run: '1' or '1'='1'--

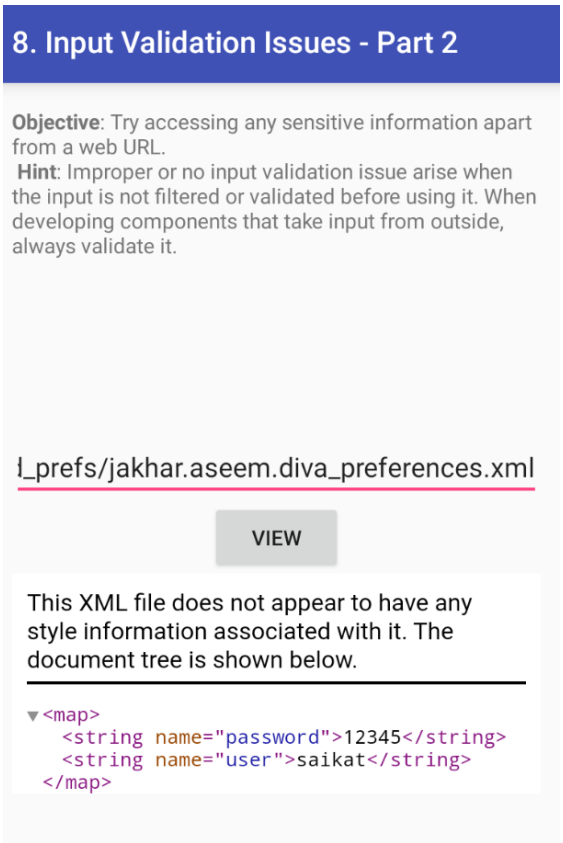


Viewing vulnerable code in JADX



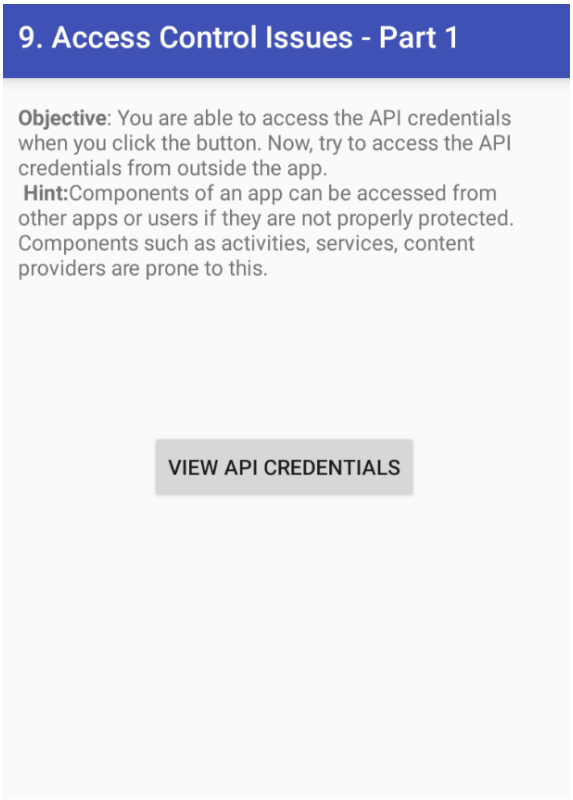
In the source code (InputValidation1Activity.java), the following line shows the vulnerability: **Cursor cr = mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);**

The input from the user is directly concatenated into the SQL query without sanitization or prepared statements, making the app vulnerable to SQL Injection.



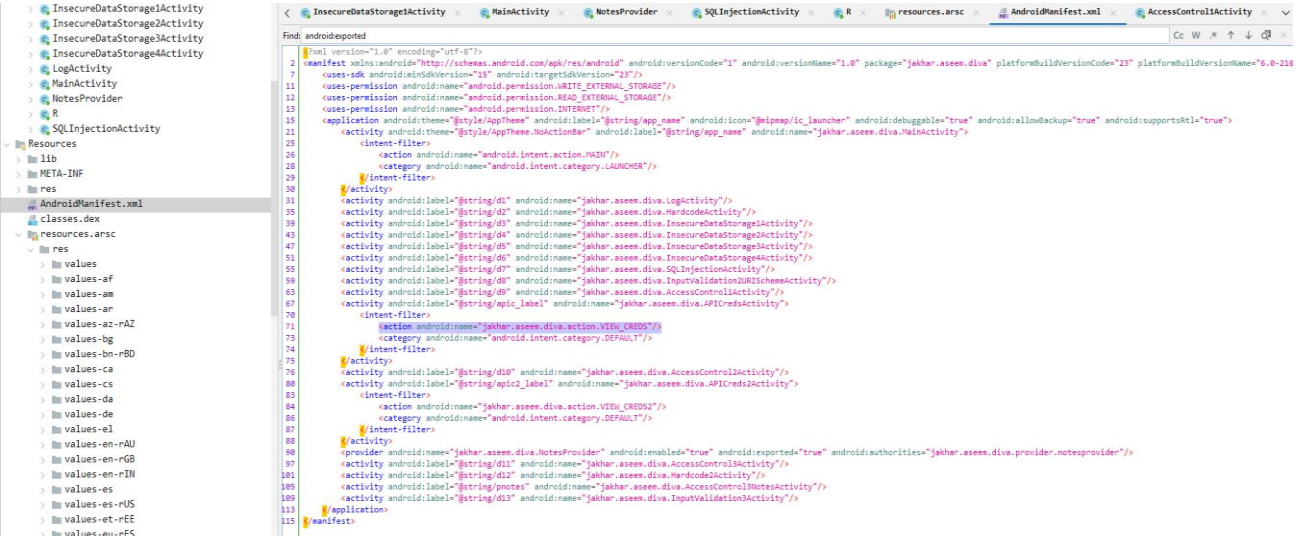
Access Control Issues – Part 1

The challenge states that API credentials can be viewed by clicking a button inside the app. The task is to try and access those credentials from outside the app, simulating how attackers exploit weak access controls.



AndroidManifest.xml showing exported activities

Looking at the app's manifest file, we see that the AccessControl1Activity is marked as exported = true.
This means other apps or even ADB commands can start this activity without restriction, exposing sensitive functionality.



Decompiled code of AccessControl1Activity

In the source code, the activity listens for an intent action:

jakhar.aseem.diva.action.VIEW_CREDS

When this intent is triggered, the app directly displays API credentials on the screen. Since there are no permission checks, any external app or user can call this intent.

Using ADB to trigger the activity externally

Using the following ADB command:

Run : adb shell am start -a jakhar.aseem.diva.action.VIEW_CREDS

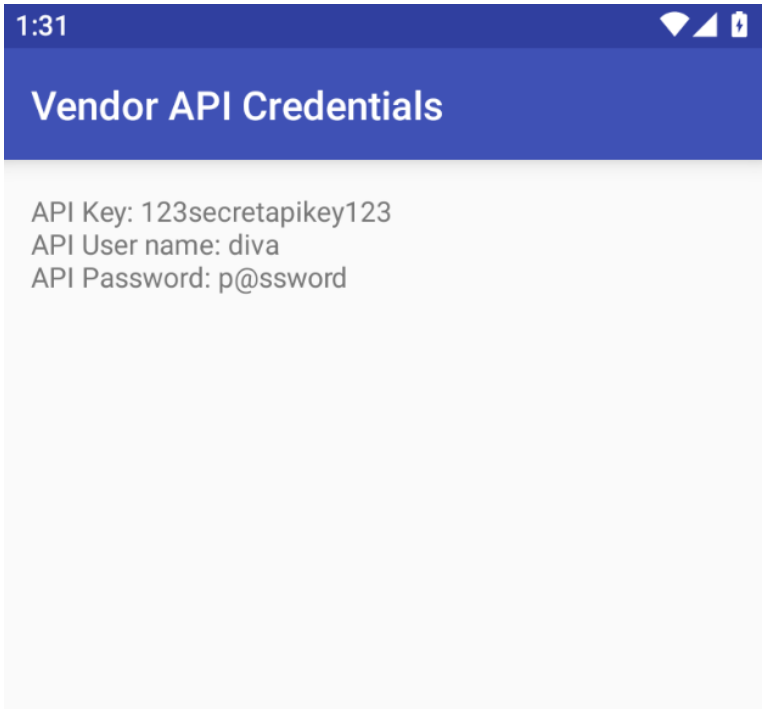
```
PS C:\WINDOWS\system32> & "C:\Program Files\Genymobile\Genymotion\tools\adb.exe" shell am start -a jakhar.aseem.diva.action.VIEW_CREDS
Starting: Intent { act=jakhar.aseem.diva.action.VIEW_CREDS }
PS C:\WINDOWS\system32>
```

API credentials displayed

As a result, the application displays sensitive vendor API credentials:

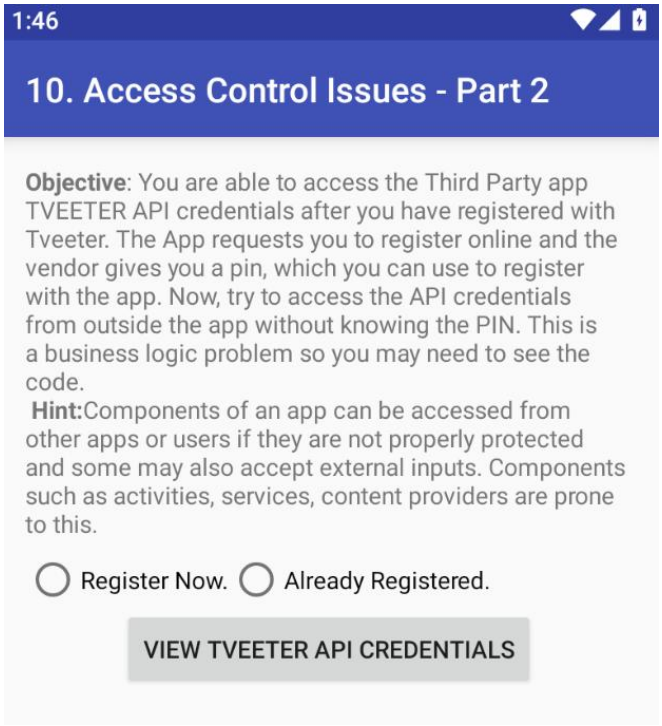
- API Key: 123secretapikey123
- Username: diva
- Password: p@ssword

This confirms that the activity is insecurely exposed.



Access Control Issues – Part 2

This challenge is about accessing TVEETER API credentials. Normally, users must register and provide a PIN to view them. The goal is to bypass the PIN check and access the credentials from outside the app.



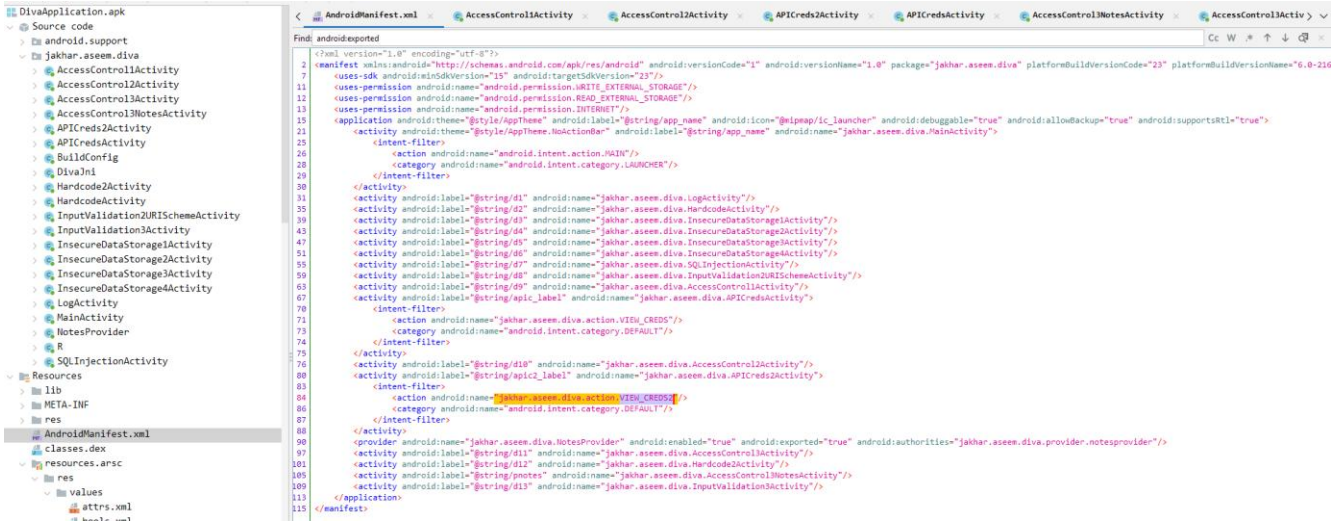
Viewing AccessControl2Activity code

The code for AccessControl2Activity shows that it expects an intent with a parameter called check_pin. If check_pin is false, the activity still proceeds to display the API credentials.
This is a business logic flaw — the app trusts the external intent parameter without verification.

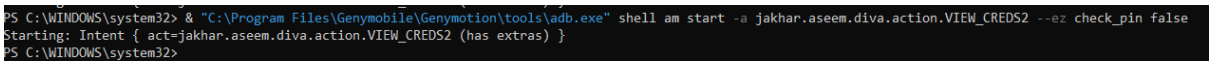
AndroidManifest.xml showing exported activity

In the manifest file, AccessControl2Activity is marked with an intent-filter that listens for the action**: jakhar.aseem.diva.action.VIEW_CREDS2**

Since it is exported and doesn't require permissions, any external app or ADB command can trigger it.



Using ADB to exploit the activity



We send the following ADB command to directly call the activity and pass a malicious parameter:

Run: adb shell am start -a jakhar.aseem.diva.action.VIEW_CREDS2 --ez check_pin false

The --ez check_pin false argument tricks the app into believing registration was already done, so it displays the credentials.

TVEETER API credentials displayed

The app exposes sensitive third-party credentials without requiring the actual PIN:

- TVEETER API Key: secrettveeterapikey
- Username: diva2
- Password: p@ssword2

This proves the access control weakness.