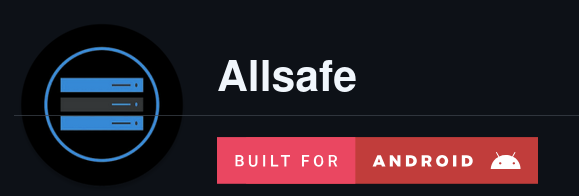
# **Reverse Engineering On Vulnerable APK**

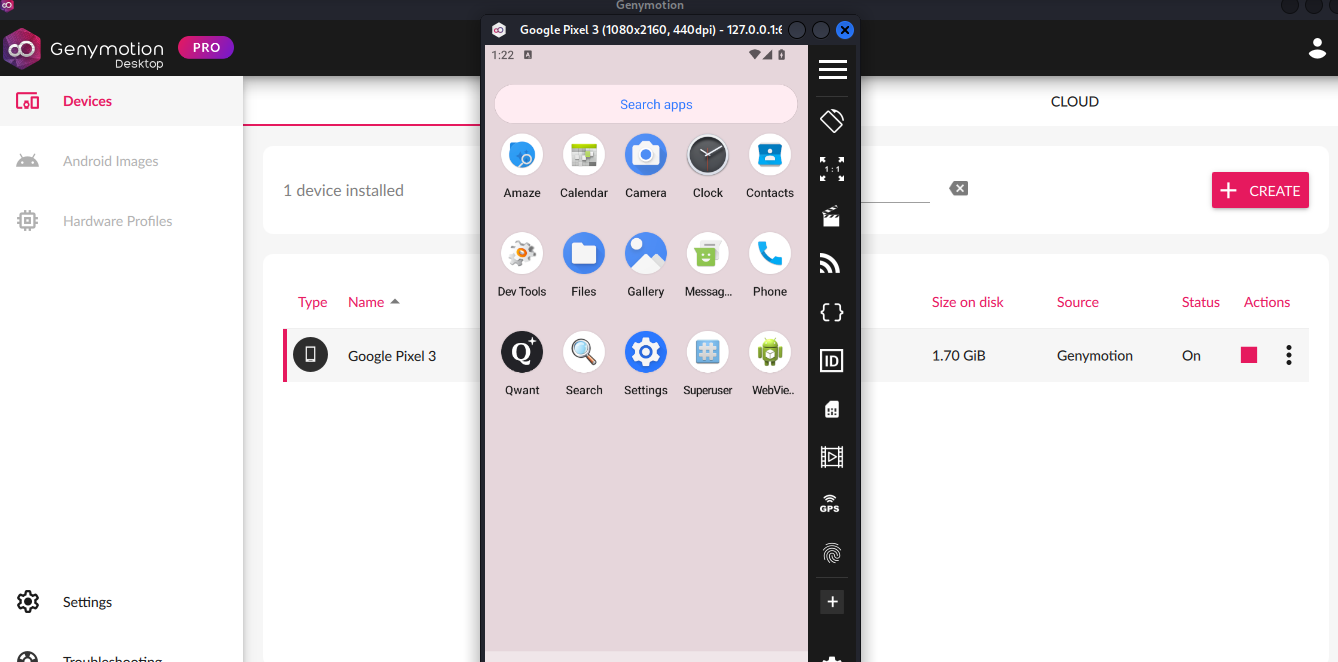
**Author :** Muhammad Ahsan Ijaz



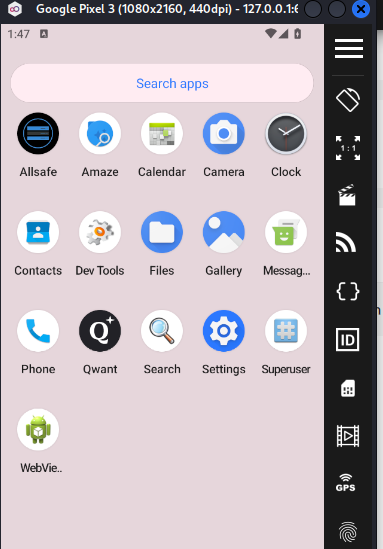
Allsafe is an intentionally vulnerable application that contains various vulnerabilities. Unlike other vulnerable Android apps, this one is less like a CTF and more like a real-life application that uses modern libraries and technologies.

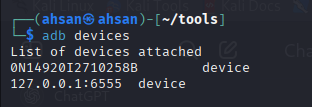
**Tools Used :**

Jadx-gui, adb, frida  
  
**Genymotion** (a popular Android emulator used for app development and security testing)

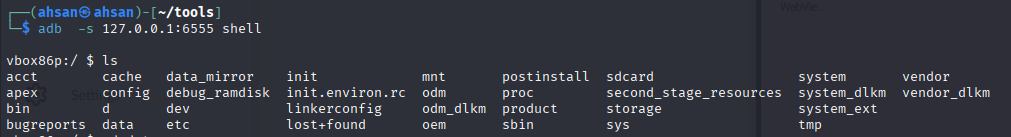


Drag and drop the **apk file** that you want to test . In this scenario I am using Allsafe apk

  
**adb devices** command, which lists connected Android devices. This verifies that the emulator is connected to the Android Debug Bridge (**adb),** a command-line tool that allows communication with an emulator or device.

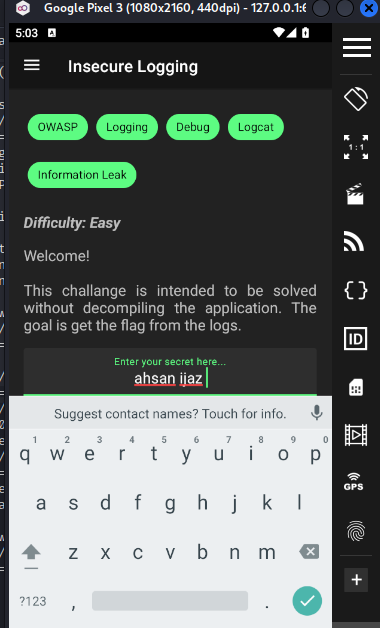


Root Access



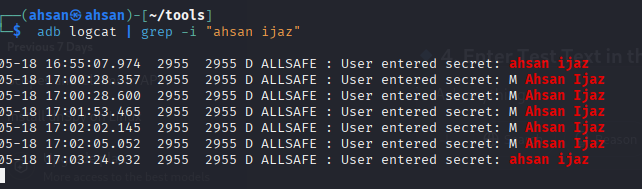
**Challenge 1**

This application logs the user-entered string in the application logs insecurely.

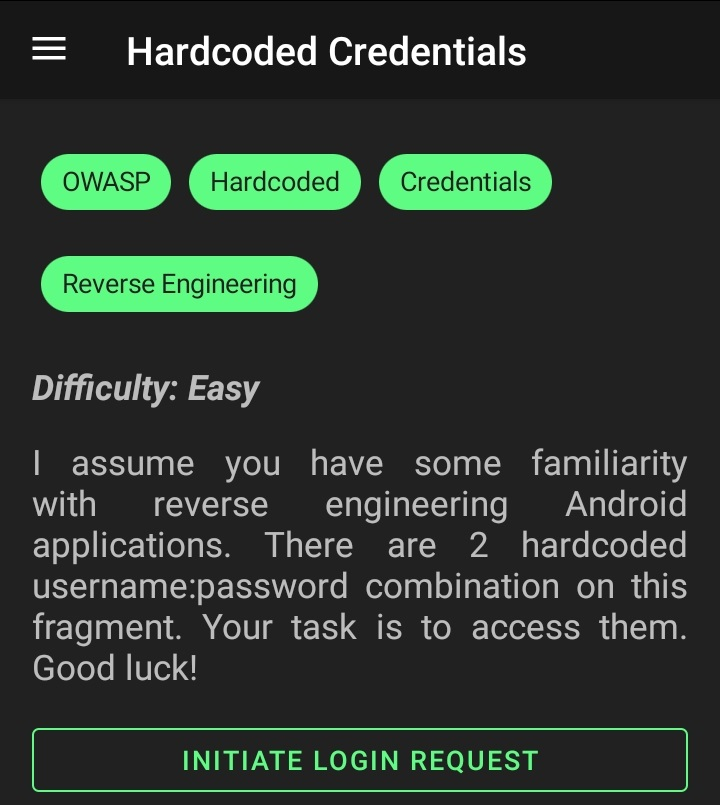


This challenge is straightforward and we don’t need to dig deep into the code. After you open the challenge and before typing anything in the textview you first need to use logcat to monitor the logs.

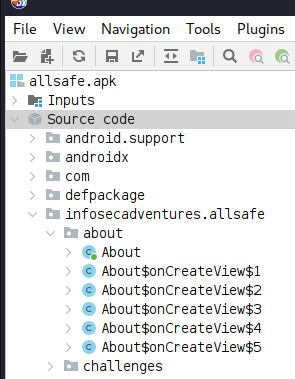
With grep we can also locate the entering text



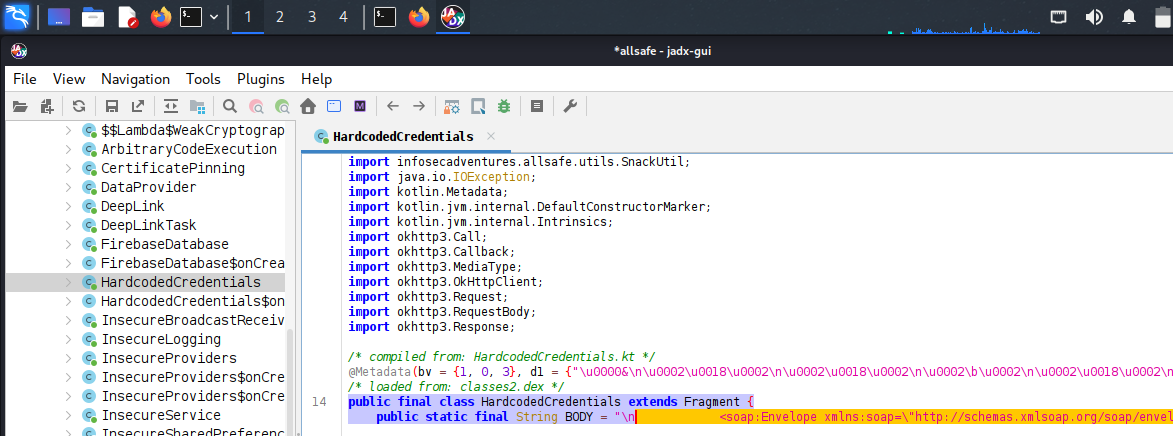
**Challenge 2**Hardcoded credentials mean that authentication details are stored in the application’s source code as plain text. To retrieve this information, we need to reverse engineer the application. For this purpose, I used the jadx-gui tool



**Navigate to Source code > challenges > hardcoded credentials**

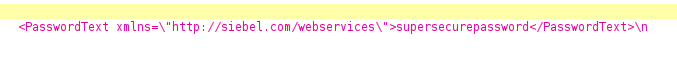


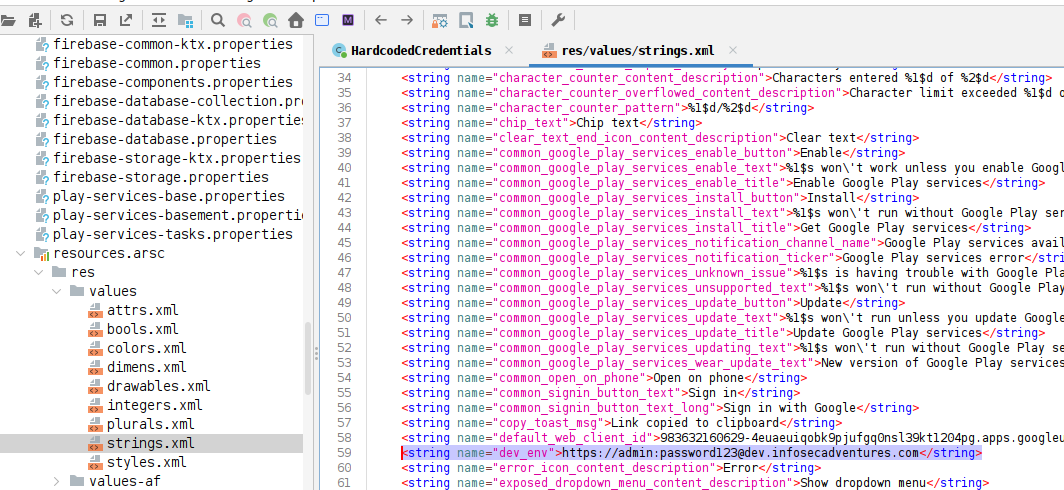
**Hard coded credentials**



When examining the class file, I observe a variable named BODY that stores SOAP request data. Within this data, we can identify a username and password combination embedded in the content.

**Username**   
**password**

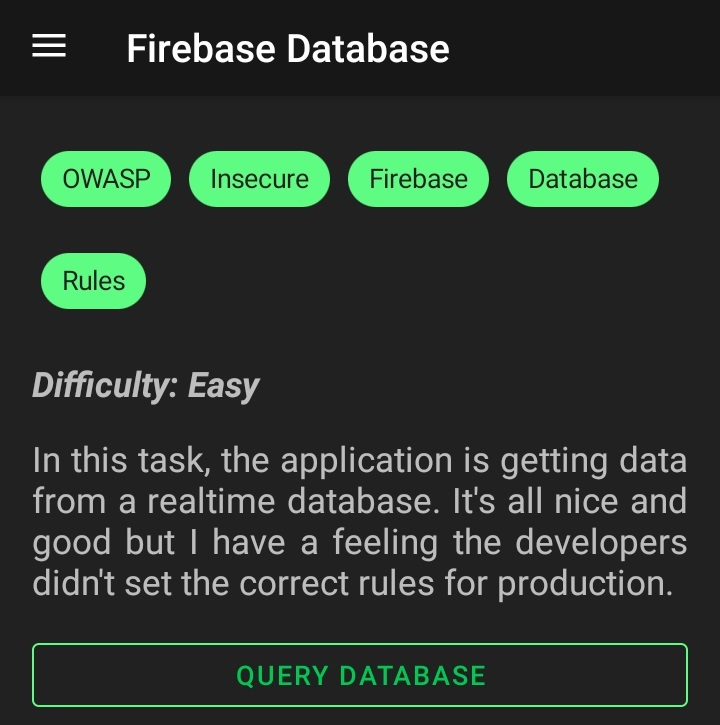
****

**Another hardcoded credentials  
**

**Username :admin**

**Password : password123**

**Challenge 3**

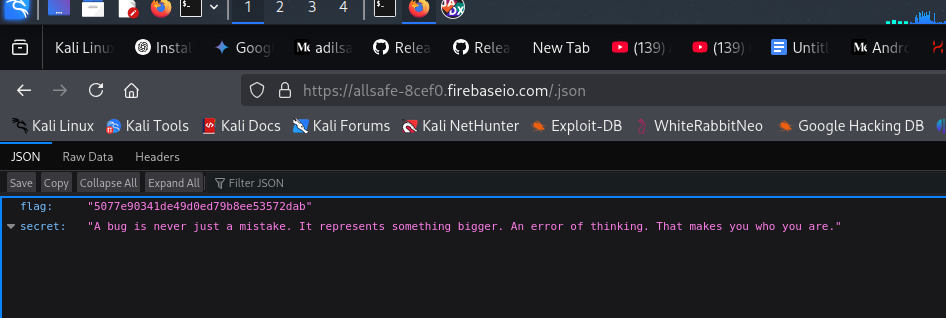
****

**Firebase** is a Realtime cloud-based NoSQL database that can be integrated with both Android and iOS apps. To start our testing we first need to find the firebase URL.



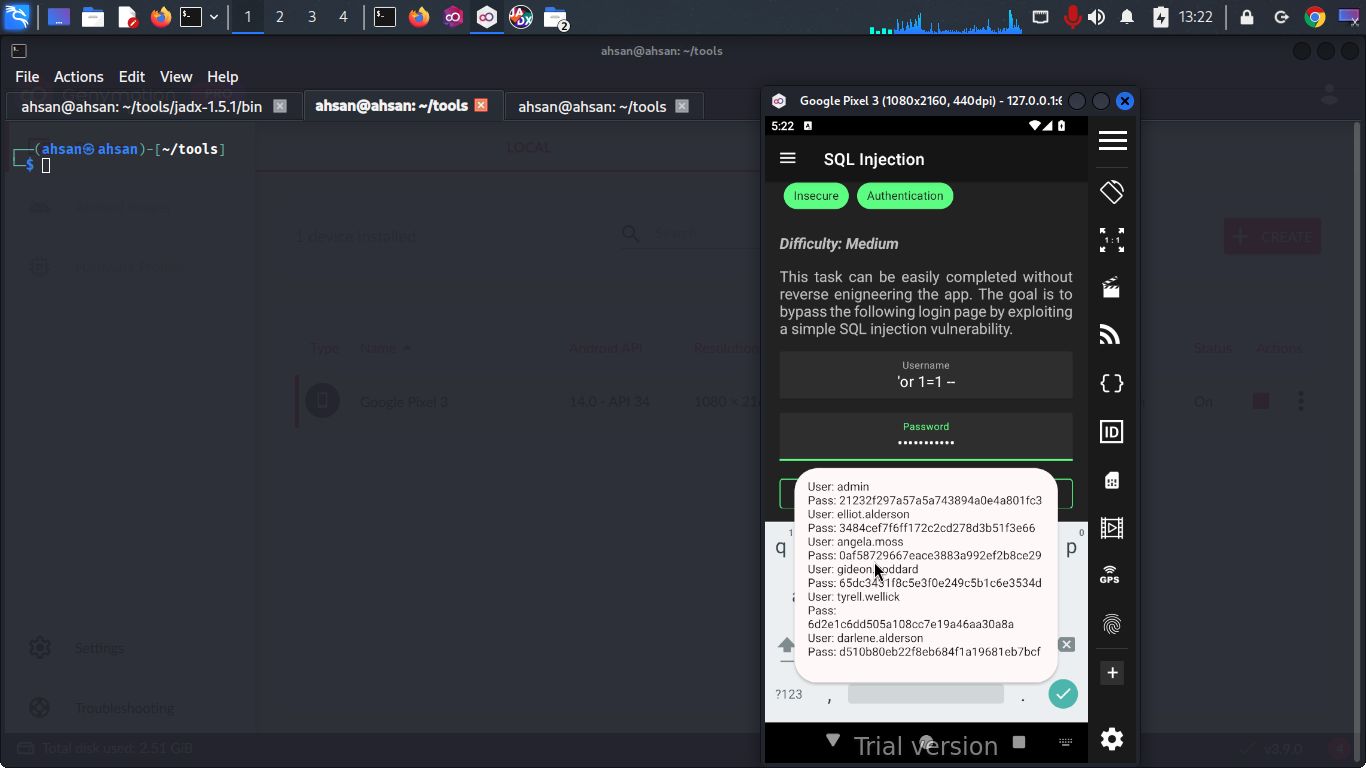


from our browser , access the /.json endpoint. If we get a Permission Denied response that means that the database is properly configured but if we get a json data or even null as a response that means you at least have read permission.

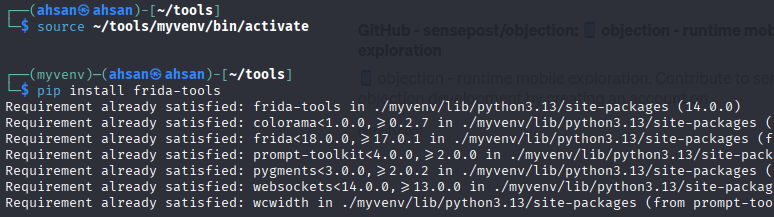
  
**We found a flag**

**Challenge 4**

**SQL injection**

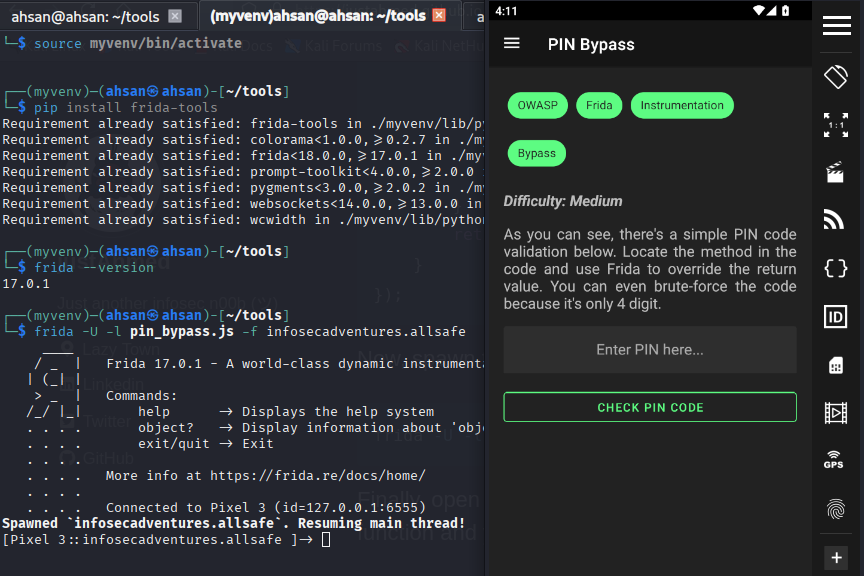


**Frida** is a dynamic instrumentation toolkit used in Android penetration testing to interact with and manipulate running applications in real-time. It allows security researchers and pentesters to inspect, hook, and modify app behavior without source code access.

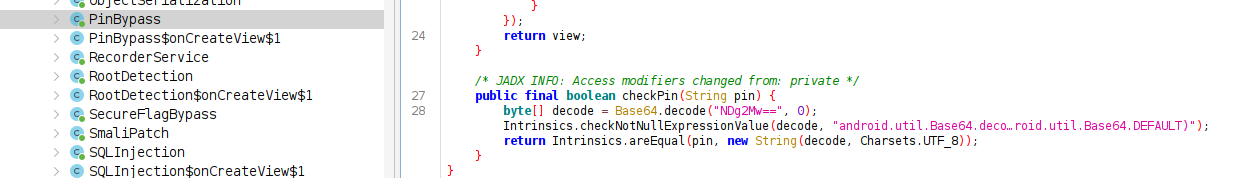


**Challenge**

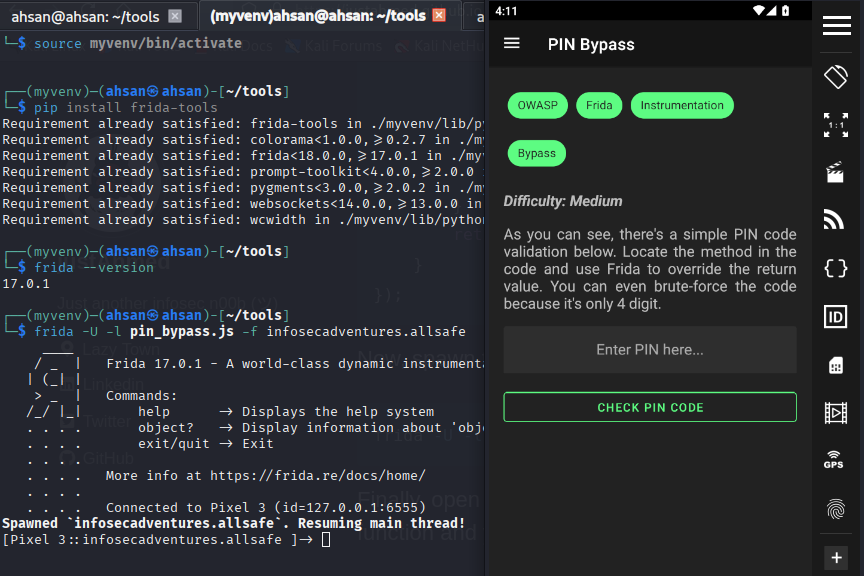
**Pin Bypass**

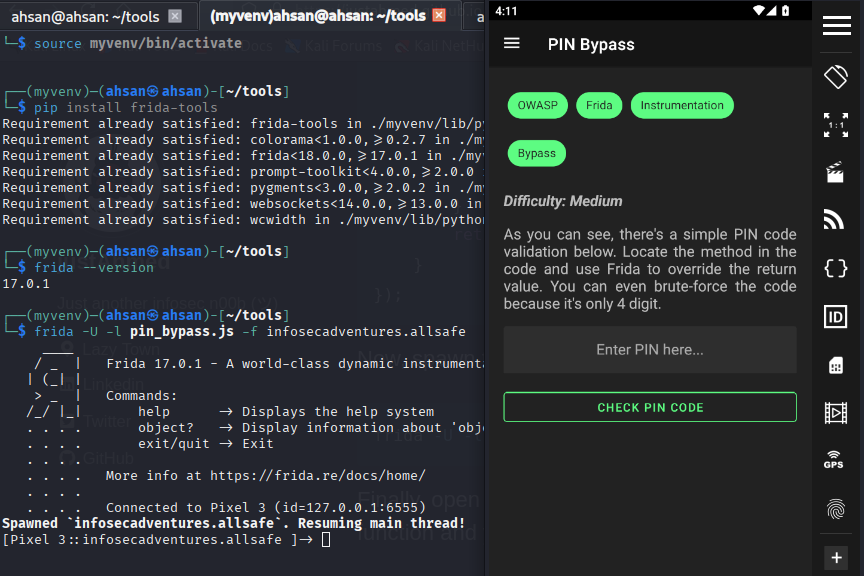


If we look at the source code, you’ll see that the function responsible for validating the PIN is called checkPin and it returns true if the PIN we entered is the same as the hardcoded one

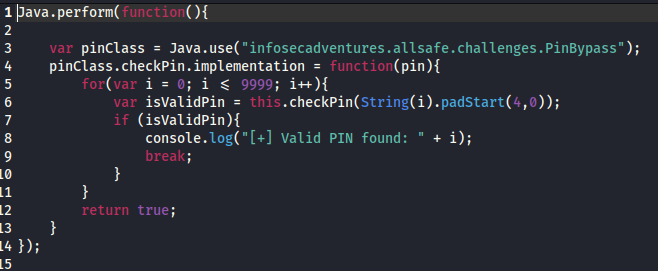
****

To brute-force the correct 4-digit PIN by calling the vulnerable method checkPin(String pin) directly we will be using Frida, instead of manually entering 0000 to 9999.





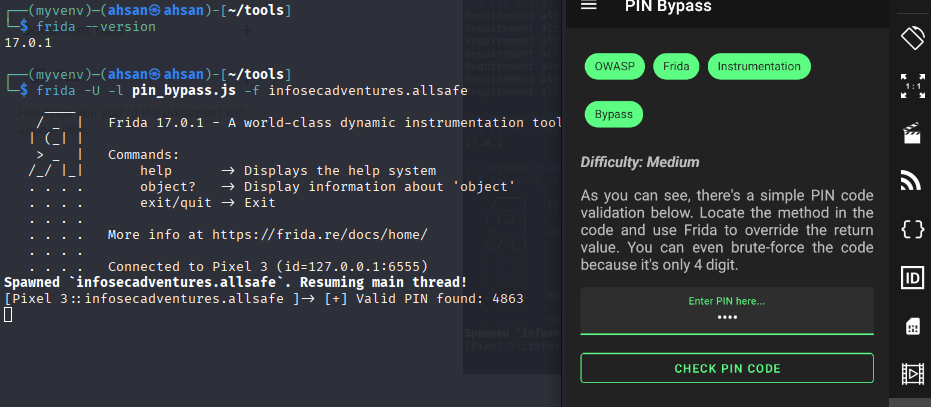
enter a dummy pin to trigger the checkPin function and you’ll find that the correct pin is logged in the frida REPL shell

**code for pin\_bypass  
  
**When the app calls checkPin("1234") from the UI, our script intercepts that call and:

1. Runs a brute-force loop from 0000 to 9999
2. Calls the original checkPin() with each value
3. When checkPin("4863") returns true, it logs:

Now, spawn the app using frida and give it the script to be injected as follows:

**frida -U -l pinbypass.js -f infosecadventures.allsafe**



**Access granted**

