Mobile Application Security Assessment

CHANDU DISSANAYAKE
CA/S3/6378

CODE ALPHA INTERNSHIP | CYBER SECURITY DOMAIN | PROJECT 2

Table of Contents

Abstract 2
Lab Setup 3
Tools And Their Functionality 4
Implementation 5
Vulnerability Analysis 7
1. Insecure Logging 7
2. Hardcoding Issues 9
3. Insecure Data Storage 11
4. Input Validation Issues 13
Conclusion 15

ABSTRACT

This project explores the security vulnerabilities in the DIVA (Damn Insecure and Vulnerable Application), an Android-based intentionally insecure app designed for learning mobile application security. The primary goal is to identify and analyze common vulnerabilities in Android applications, such as insecure logging, hardcoding issues, insecure data storage, and improper input validation. Using a hands-on lab setup, tools such as ADB (Android Debug Bridge) and JADX were utilized to uncover these flaws and propose potential mitigations. This report provides insights into the technical aspects of mobile security and showcases how attackers can exploit vulnerabilities to compromise an application. Ultimately, the report emphasizes the way of identifying and exploiting such security flaws.

LAB SETUP

When considering Lab setup, it is possible to use Windows, Linux or Mac Operating Systems. However, I selected Windows platform because it is better to use Host Operating System other than virtual one.

Requirements

- **Laptop/PC:** Running Windows Operating System with ADB tools installed for interacting with the Android device was used.
- **Android Device:** Here in this case, Redmi Note 11 Pro was used, configured with Developer Options and USB debugging enabled.
- **DIVA Installation:** The DIVA APK was installed on the Android device as the target application.
- **Reverse Engineering Tool:** JADX GUI was used to decompile and explore the APK's source code for vulnerabilities.

Step By Step Process

1. Enabling Developer Options and USB Debugging:

- Navigate to Settings > About Phone on the Android device.
- Tap on "Build Number" seven times to enable Developer Options.
- Go to Settings > Developer Options and enable USB Debugging.

2. Connecting Android Device to Laptop:

- Use a USB cable to connect the phone to the laptop.
- Verify the connection using the command:
- cmd > "*adb devices*"
- This should list the connected device.

3. Installing DIVA APK:

- Download the DIVA APK from the official GitHub repository.
- Install the app

4. Setting up JADX for Reverse Engineering:

- Install the JADX GUI on the laptop.
- Pull the APK from the connected phone:
- Cmd > "*adb pull /path/jakhar.aseem.diva/base.apk*" .
- Open the APK in JADX to decompile and analyze its source code.

TOOLS AND THEIR FUNCTIONALITY

Android Debug Bridge (ADB)

ADB is a versatile command-line tool that allows communication between a computer and an Android device. It is widely used by developers and security professionals to execute commands, install or uninstall applications, and access device logs. ADB facilitates debugging by enabling control over a connected Android device via USB or a wireless connection. One of ADB's most powerful features is the *ADB shell*, which acts as a terminal for the Android device, allowing users to execute commands directly on the device's operating system. This capability provides deep access to the device's file system and running processes, making ADB an essential tool for exploring, analyzing, and testing Android applications.

Damn Insecure and Vulnerable Application (DIVA)

DIVA is a specially designed Android application created for educational purposes to teach mobile security vulnerabilities. It contains intentionally insecure code that mimics real-world application flaws, such as insecure logging, weak data storage practices, and poor input validation. DIVA provides a hands-on platform for security enthusiasts to understand and exploit common vulnerabilities. By testing DIVA, users can develop skills in penetration testing, learn about Android security best practices, and identify potential risks in real-world applications.

JADX-GUI

JADX-GUI is a powerful reverse engineering tool used to decompile Android APK files into human-readable Java source code. Its graphical interface simplifies the analysis process, allowing users to navigate through an application's code structure easily. By using JADX-GUI, security analysts can identify issues such as hardcoded credentials, improper coding practices, or unprotected API keys. This tool is essential for understanding how applications are developed, uncovering vulnerabilities, and providing insights for improving application security.

IMPLEMENTATION

As demonstrated below, the first thing I done was enabling developer options and turning on USB Debugging in Android phone. Then, I installed DIVA (Damn Insecure and Vulnerable Application) into the mobile.

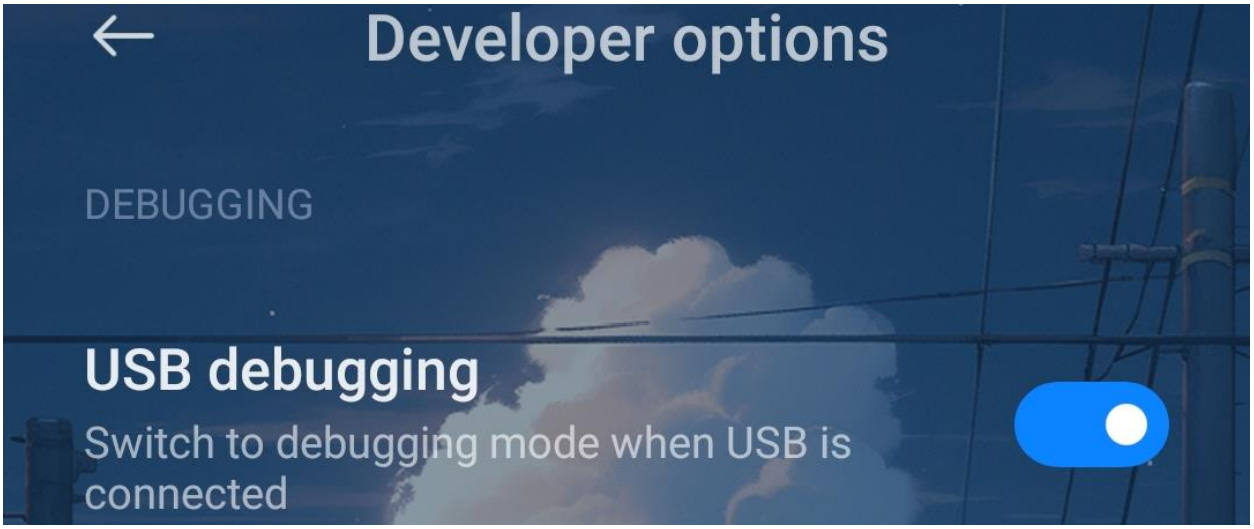

Figure 1: Turning on USB debugging

Thereafter, I connected the phone to the lap via USB cable. As I mentioned previously, here, I used the Windows Operating System to complete the project. So, I installed the ADB package manually and added that path as a system variable. Therefore, I was able to access ADB from the command line interface. I verified the connection using "adb devices" command. It will demonstrate the serial numbers of connected devices. Next, with the use of adb console, I explored the content inside the mobile and pulled the data related to DIVA.

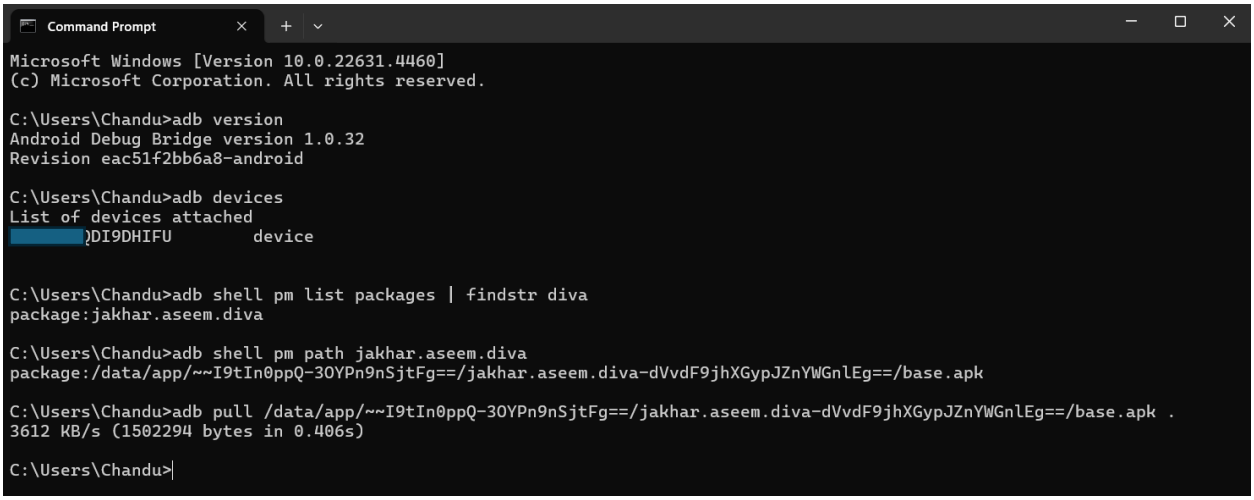

Figure 2: Pulling the data related to DIVA

As the Reverse Engineering Tool, I installed JADX – GUI on to the lap and opened the base.apk which was the pulled data from DIVA.

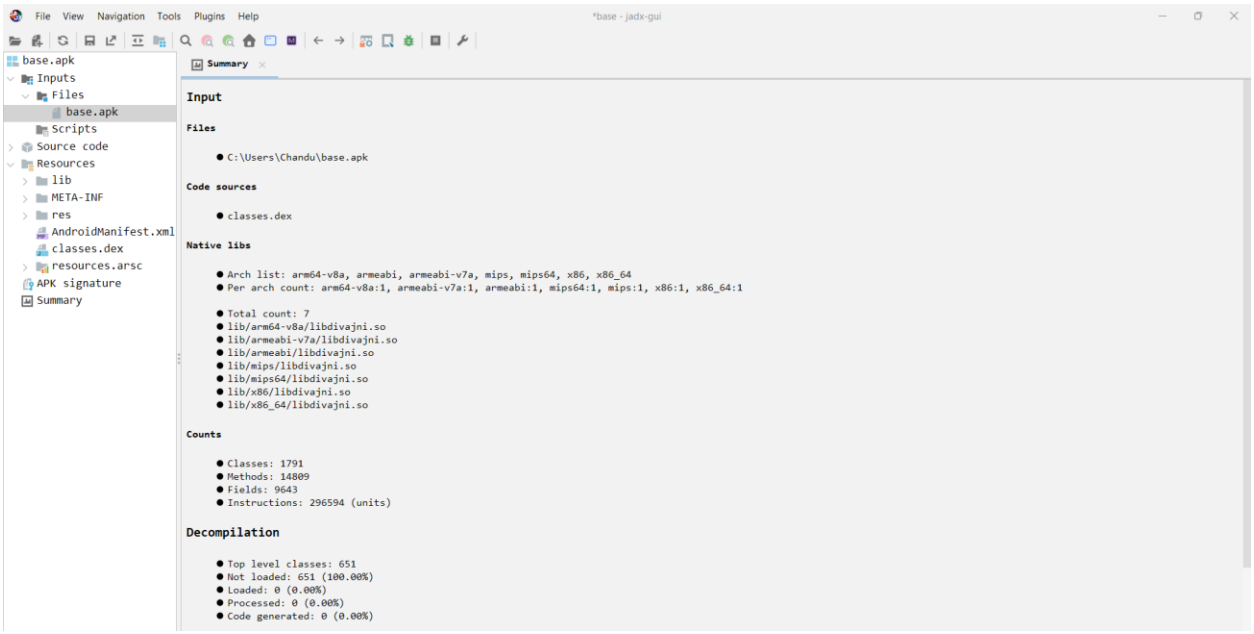

Figure 3: Opening base.apk with JADX-GUI

VULNERABILITY ANALYSIS

1. Insecure Logging

Here, the first security flaw that I analyzed was about Insecure Logging. Android applications often log critical information for debugging purposes. Insecure logging occurs when sensitive data, such as user credentials or tokens, is written to system logs.

Analysis:

Below is the code snippet related to the insecure logging part within DIVA. According to the code snippet, one thing is clear that the user input is going to be noted within a log file.

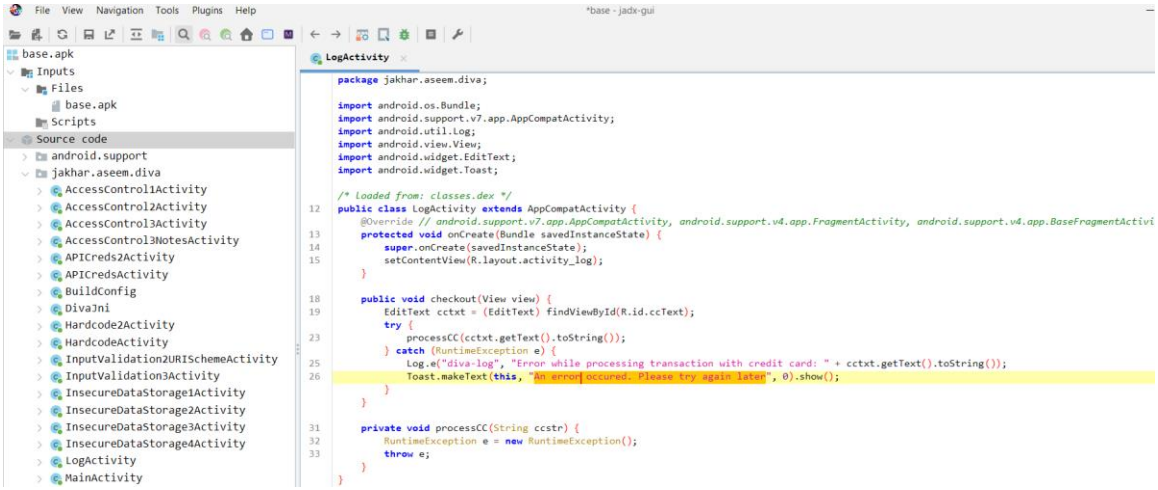

Figure 4: Code snippet related to the insecure logging

So, I used adb shell (terminal of the mobile) to explore the log file. First of all, I ensured the name of the package and then I searched for pid value of diva app. I got the pid value as 9770. Then I searched for logs related to the pid value of 9770 with the use of logcat.

Figure 5: Explore the logs related to the pid value of 9770 (diva)

I accessed real time log. Behalf of that, I used my mobile and opened the diva app. Then entered a value within Insecure logging section as “123456” and clicked check out.

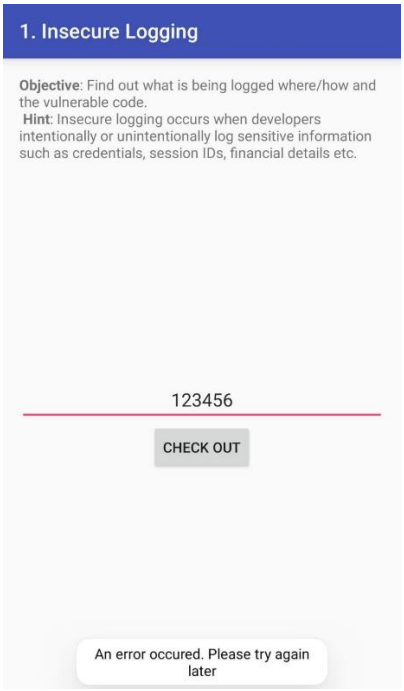

Figure 6: Opening diva app and enter a value into the insecure logging section

Here, that entered value was noted within the log. It proves that the diva application contains a vulnerability related to insecure logging.

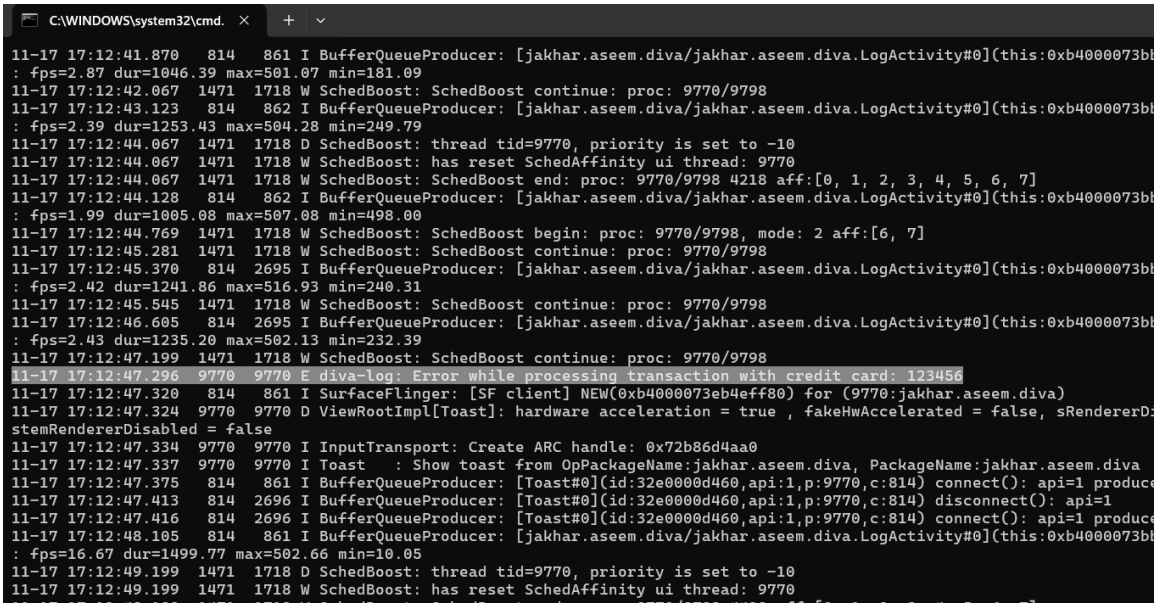

Figure 7: The real time log which noted the entered value

Impact:

- Attackers with access to the device logs could extract sensitive information, leading to credential theft or unauthorized access.

Mitigation:

- Developers should avoid logging sensitive data in production environments.
- Use secure logging libraries that mask or exclude sensitive information.

2. Hardcoding Issues

The second security flaw that I analyzed was a Hardcoding Issu. Hardcoding sensitive information, such as API keys, credentials, or encryption keys, into the source code is a common mistake.

Analysis:

Here, the first thing I done was open the diva application using the mobile and enter a value into the Hardcoding issue section as “abc123”. I got a popup message as “Access denied”.

Figure 8: Opening diva app and enter a value into hardcoding issue section

Here, below is the code snippet related to the hardcoding issue. Source code itself contains the key or the password to get the response as “Access granted”. I highlighted it within the screenshot. The key is “vendorsecretkey”. That means the application is vulnerable because of the hardcoding issue.

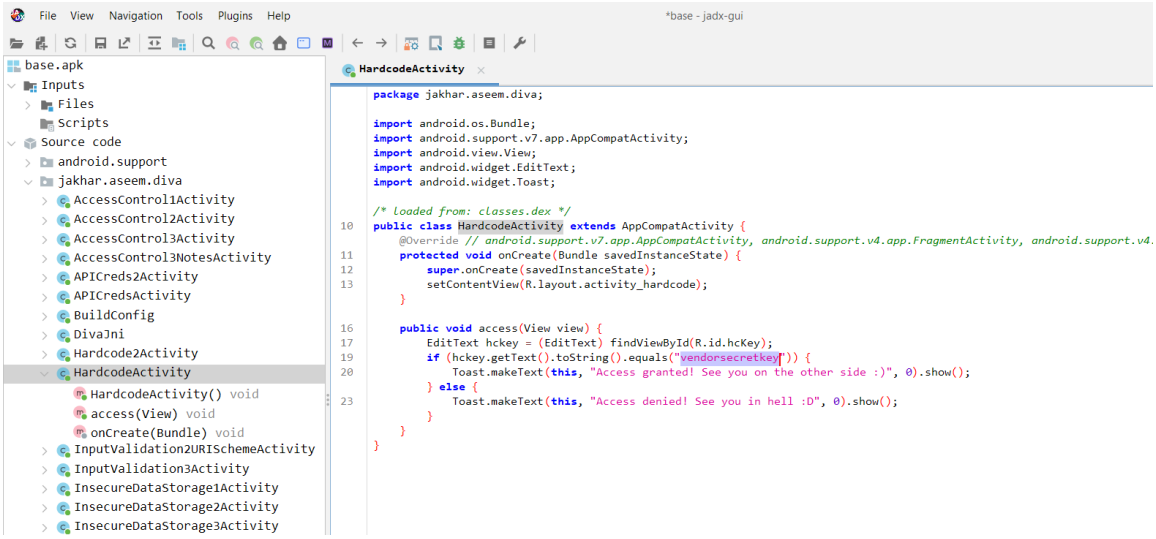

Figure 9: Code snippet related to the hardcoding issue

Again, I accessed the hardcoding issue section in diva application and entered the key into it. I got the response as “Access granted”.

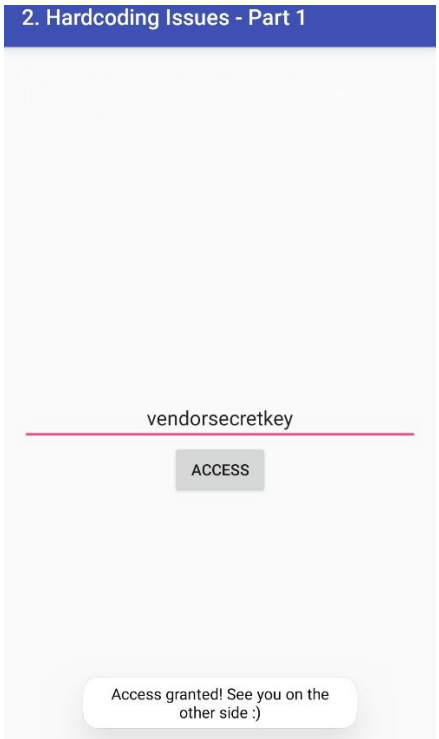

Figure 10: Enter the key that found within source code

Impact:

- An attacker can extract and misuse hardcoded values for unauthorized API access or further exploitation.

Mitigation:

- Use environment variables or secure storage mechanisms to handle sensitive information.
- Implement runtime retrieval of keys from secure servers.

3. Insecure Data Storage

The third security flaw that I analyzed was Insecure Data Storage. Improper handling of sensitive data in local storage can lead to its exposure, especially if weak encryption or no encryption is used.

Analysis:

Here, the first thing I done was open the diva application using the mobile and enter a username and a password into the insecure data storage section as “wmcd” and “helloimchandu” respectively. Then I clicked save and got a message as “credentials saved successfully”.

Figure 11: Enter a username and a password into insecure data storage section in diva app

Below is the code snippet related to the insecure data storage vulnerability. According to the code snippet, credentials should be saved within a directory that represents shared preferences.

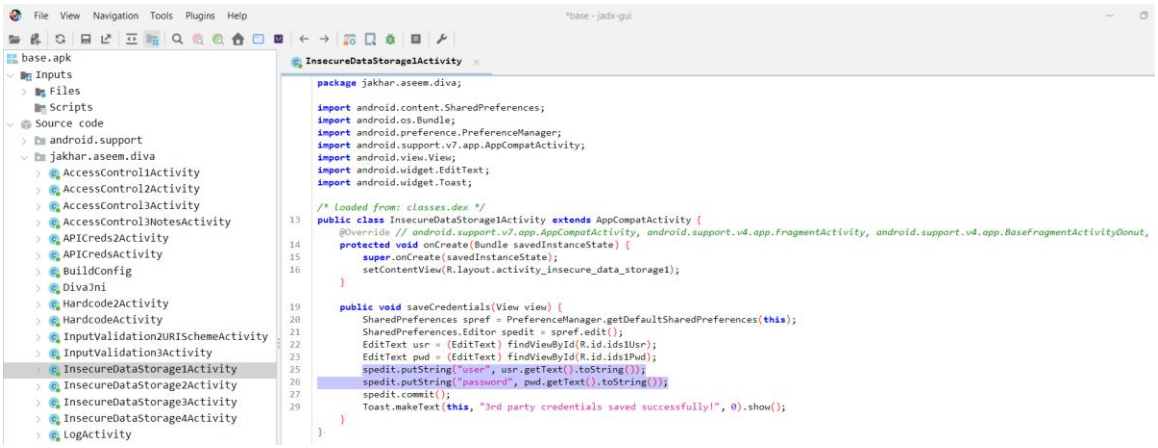

Figure 12: Code snippet related to the insecure data storage vulnerability

As I found there are four folders related to the diva application. One folder name is somewhat similar to the name that we got from source code. That is “shared_prefs”. I explored that folder and found a .xml file. After opening it, I was able to see the credentials as below. That means here there is an insecure data storage vulnerability.

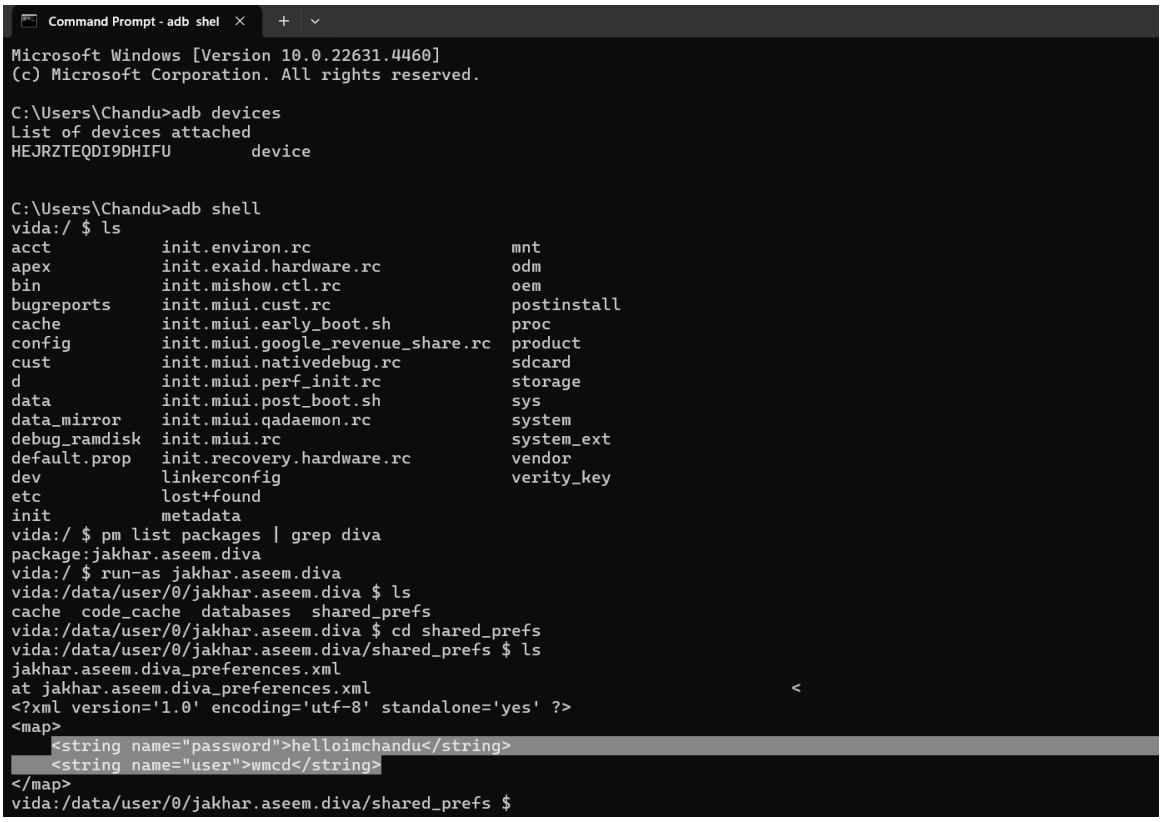

Figure 13: Exploration of finding the saved credentials

Impact:

- If an attacker gains access to the device, they can extract sensitive data and compromise the user.

Mitigation:

- Use Android's EncryptedSharedPreferences or secure libraries like SQLCipher for data storage.
- Implement access controls to protect sensitive files.

4. Input Validation Issues

The fourth security flaw that I analyzed was an input validation issue. Improper input validation can lead to vulnerabilities such as SQL injection or code injection.

Analysis:

Here in the input validation issue section in diva application, it is mentioned that "try to access all user data without knowing any username". First name I tried was "admin". Because as a tradition, admin keyword is used by many programmers and creators to test or to analyze the applications. Anyhow, I received the information about that user.

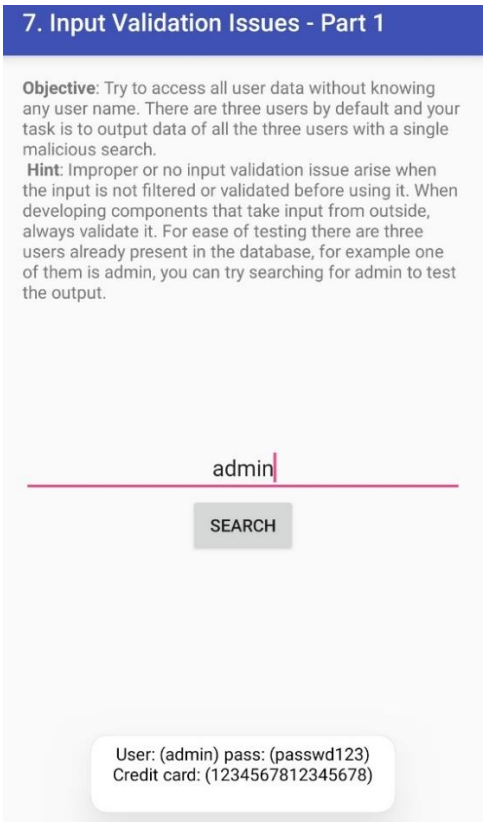

Figure 14: Enter admin as the username for input validation issue section in diva app

As below, here I done a SQL injection and retrieved information about all users. "SELECT * FROM users WHERE username = 'admin' OR 1=1--';". This should be the SQL query executed in back end. It simply means retrieving all the data related to users where username equals admin or 1 equal 1. Accordingly, 1=1 the condition is true, so it will retrieve the information of all users. That's how the SQL injection attack works. Input validation is required to prevent these attacks.

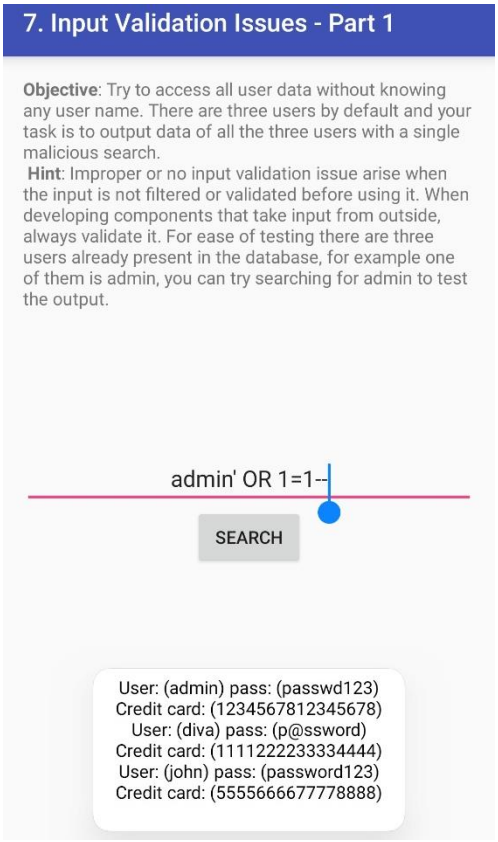

Figure 15: Launching an SQL Injection attack to retrieve info for all users

Impact:

- An attacker can execute arbitrary code, retrieve unauthorized data, or crash the application.

Mitigation:

- Validate all inputs on both the client and server side.
- Use parameterized queries to prevent SQL injections.

CONCLUSION

This project highlights the significance of secure coding practices, and the potential risks associated with common vulnerabilities in Android applications. By analyzing DIVA, we demonstrated how attackers can exploit insecure logging, hardcoded values, insecure data storage, and improper input validation to compromise an application. Each vulnerability was identified using practical tools and techniques, showcasing real-world scenarios.

Key Takeaways:

- Security should be integrated into the development of a lifecycle to prevent vulnerabilities.
- Developers must follow best practices, such as avoiding hardcoding sensitive data, implementing input validation, and encrypting sensitive data.
- Applications should undergo regular security audits and penetration testing to identify and fix vulnerabilities.

By addressing these vulnerabilities, we can build secure and robust applications that protect user data and ensure a safe user experience.