DIVA MOBILE APPLICATION PENETRATION TESTTING

report

AMJAD AMEEN PP

23-11-24

Contents

1.Assessment Overview

From Nov 18th, 2024 to Nov 24th, 2024, DIVA engaged my company to evaluate the security posture of its infrastructure compared to current industry best practices that included an external penetration test. All testing performed is based on the *OWASP Mobile Application Security Testing Guide (MASTG),*

Phases of penetration testing activities include the following:

1. Reconnaissance and Information Gathering
    - Objective: Collect information about the app, its platform, and backend to plan the test.
2. Static Analysis
    - Objective: Analyze app code (source or decompiled) to find vulnerabilities without running it.
3. Dynamic Analysis
    - Objective: Test the app's behavior during execution to detect vulnerabilities.
4. Backend/API Testing
    - Objective: Assess APIs and backend systems for security weaknesses.
5. Device-Level Testing
    - Objective: Check for insecure data storage or improper interactions with the device.
6. Reverse Engineering and Tampering
    - Objective: Evaluate app resilience to reverse engineering and unauthorized modifications.
7. Reporting
    - Objective: Document findings with remediation steps mapped to the OWASP Mobile Top 10.

2.Components

2.1 Penetration Test

Emulate an attacker attempting to exploit vulnerabilities in a mobile application to identify risks to sensitive data and systems. This includes gathering information about the app, analyzing its security posture, and identifying exploitable vulnerabilities.

2.2 Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

CVSS Score	Severity Level
0.0	None
0.1–3.9	Low
4.0–6.9	Mediam
7.0–8.9	High
9.0–10.0	Critical

Summary of Key References:

- **OWASP Mobile Top 10**: Provides a list of the most common mobile app risks. Used to categorize vulnerabilities.
- **CVSS v3.1**: Standardized scoring system for vulnerabilities. Helps map security risk to severity.
- **Security Impact Analysis**: Used to estimate the potential impact of vulnerabilities on the security of the mobile application.

3. Scope

Target Application

- Apk : diva-beta.apk
- Domain: mobile application

4. Executive Summary

The penetration testing of the **Diva Mobile Application** was conducted using **Genymotion**, **ADB Shell**, and **JADX CLI**, focusing on the **OWASP Top 10 Mobile Risks**. The assessment identified several security issues, including insecure data storage, weak encryption, and flaws in authentication mechanisms. Key vulnerabilities were found in areas such as code tampering and reverse engineering, which could potentially expose sensitive data or allow unauthorized access. The report provides detailed findings and recommendations to enhance the security of the application, including strengthening encryption, improving authentication practices, and addressing code vulnerabilities.

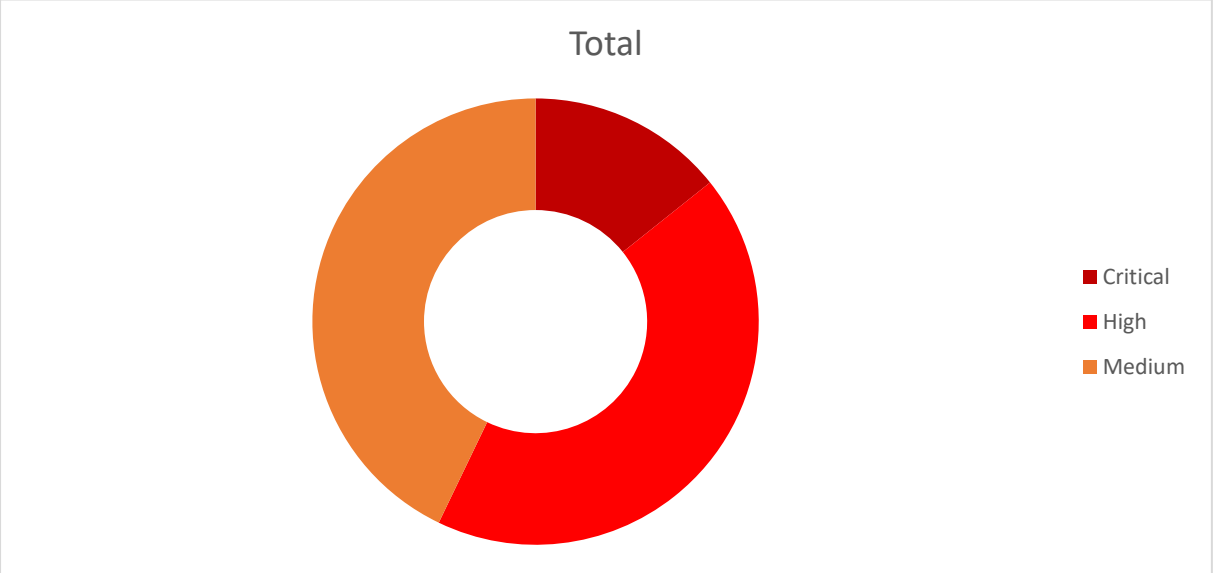
Tools Used:

- **Genymotion Simulator**: To emulate various mobile environments for testing vulnerabilities.
- **ADB Shell**: For interacting with the application on the Android device and performing commands to analyze behavior.
- **JADX CLI**: Used to decompile and analyze the mobile application's APK, searching for code weaknesses, insecure configurations, and reverse engineering potential.

5. Attack summary

SI NO	Vulnerability	Description	**Severity**
1	Insecure Logging	The application fails to properly log sensitive data, potentially exposing information in logs that can be accessed by attackers.	Medium
2	Hardcoding Issue	Sensitive information such as API keys, passwords, or tokens are hardcoded in the application code, making them easily exploitable.	High
3	Sensitive Data Stored in Shared Preferences	Sensitive information is stored insecurely in Shared Preferences, which can be easily accessed by attackers with physical access to the device.	High
4	Sensitive Data stored in local storage	Sensitive information is stored insecurely in local storage, which can be easily accessed by attackers with physical access to the device.	Medium
5	Sensitive Data Stored in Temporary Files	Sensitive data is stored in temporary files, making it vulnerable to unauthorized access, especially if the device is compromised.	Medium
6	Input validation issue (sql injection)	his vulnerability occurs when user input is directly concatenated into SQL queries without proper sanitization or parameterization, enabling attackers to manipulate database queries.	Critical

7	Input validation issue (web URL)	This vulnerability occurs when a web application fails to properly validate or sanitize user input, specifically in URL parameters. Malicious users can manipulate the URL to inject unwanted characters, commands, or harmful data, leading to potential security risks such as unauthorized access, data leakage, or execution of malicious code.	High



5.1 Insecure Logging

Description	Insecure logging happens when sensitive information (like passwords, tokens, or personal data) .in here we found credit card number
Impacts	Technical impacts: • Data Exposure: Sensitive information (e.g., passwords, API keys, personal data, session tokens) can be logged in plaintext and accessed by unauthorized users or attackers, leading to data breaches.

	• Privilege Escalation: Attackers gaining access to logs can obtain valuable information such as authentication tokens or system configurations, potentially escalating privileges or bypassing security controls. User impacts • Data Breach: Sensitive information such as passwords, session tokens, or personal data may be exposed in logs, increasing the risk of a data breach if the logs are accessed by unauthorized users. • Unauthorized Access: Attackers gaining access to logs could extract authentication credentials or other sensitive data, enabling them to impersonate users or gain unauthorized access to accounts or resources. • Privilege Escalation: Attackers can use information from logs (e.g., error messages, stack traces) to escalate their privileges, bypass security mechanisms, or exploit vulnerabilities in the application.
Mitigations	• Avoid logging sensitive information (e.g., passwords, tokens, personal data). • Mask or encrypt sensitive data before logging. • Restrict log file access with proper access controls. • Use secure logging mechanisms with encryption.

Proof of concept :

Our first task in the Damn Insecure and Vulnerable Application (DIVA) is capturing sensitive data in logs.

First of all I have performed the adb devices to check whether the device is connected or not

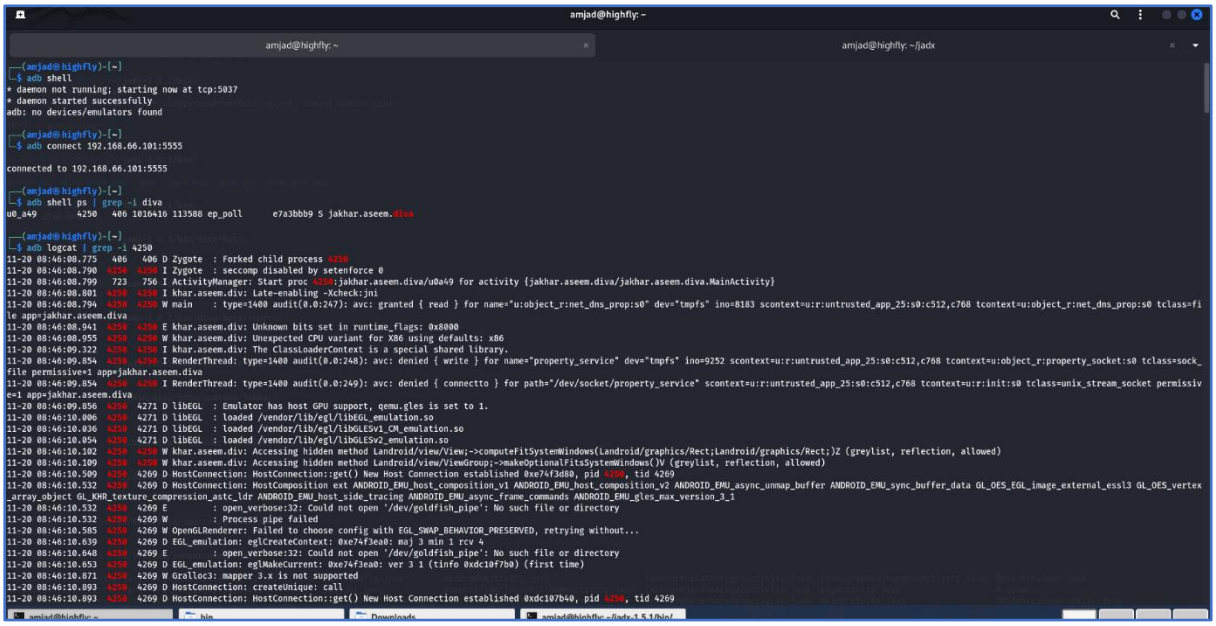
commands are :

adb shell connection checking : adb shell

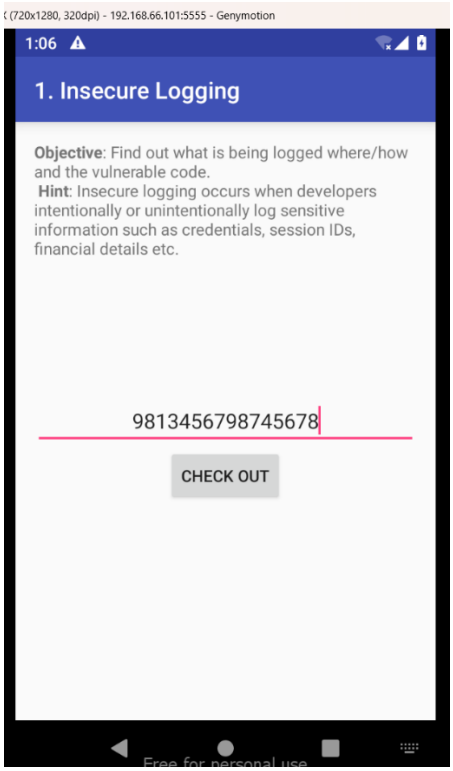
connecting device : adb connect $ip

Identifying Running Processes(pid) : adb shell ps | grep -i diva

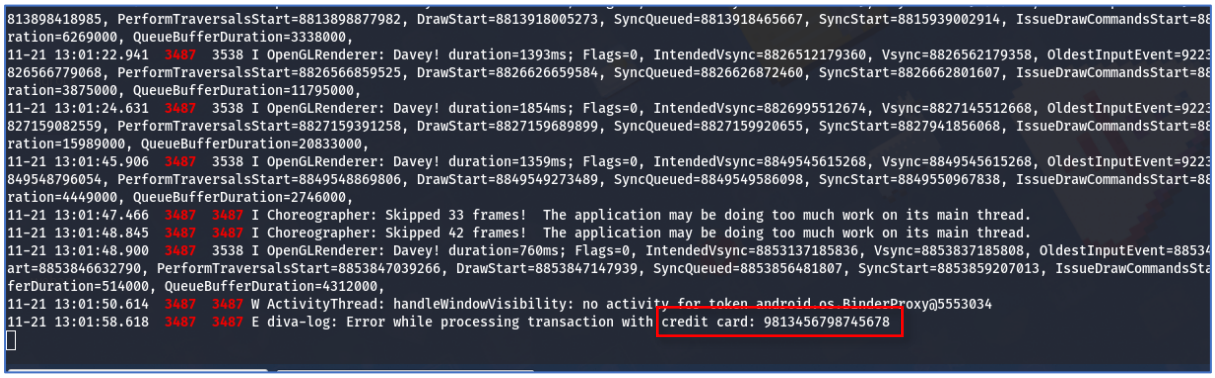
Identifying Sensitive Information in logcat : adb logcat |grep -i (pid)



Then we add a value in insecure logging page on diva



The credit card number displayed in logcat



5.2 Hardcoding issue

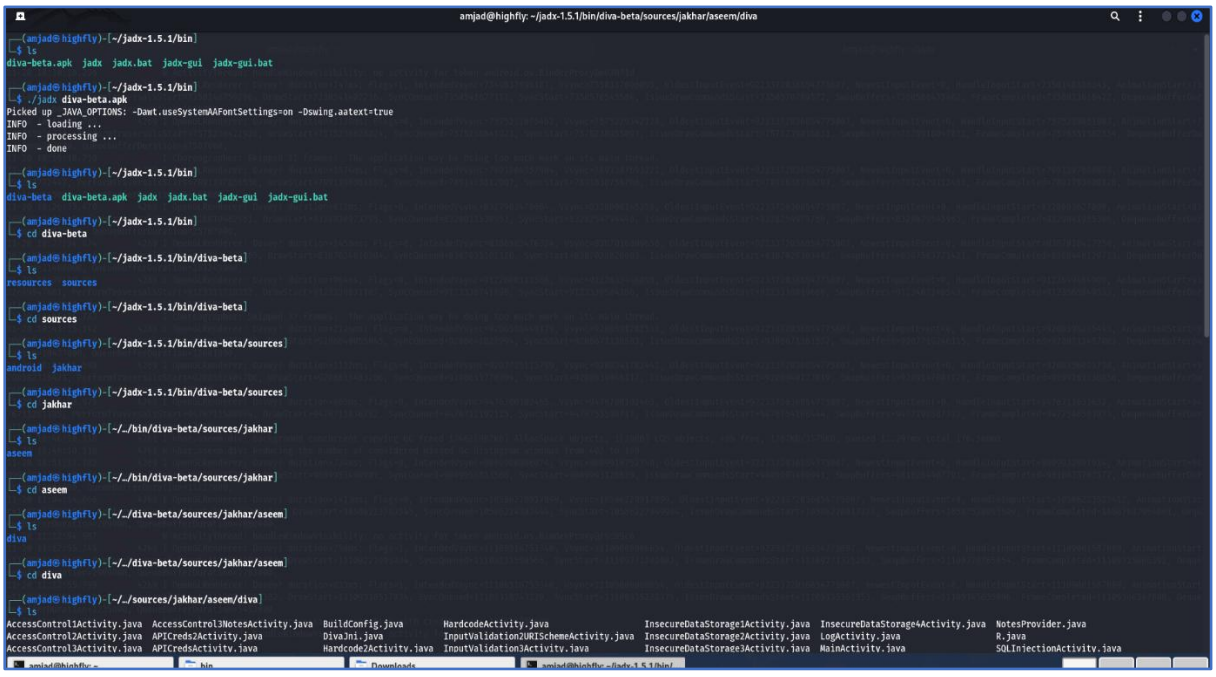
Description	Hardcoding occurs when sensitive information, such as API keys, passwords, encryption keys, or other confidential data, is directly embedded in the mobile application's source code. This makes the information accessible to anyone who can decompile or reverse-engineer the app. In here using jadx-cli tool for tool used to decompile Android APK files and view the source . in here we take the encryption key from source code
Impacts	Technical impacts : • **Data Leakage**: Sensitive data can be extracted from decompiled source code. • **Unauthorized Access**: Attackers can misuse hardcoded credentials to gain access to systems. • **Security Risks**: Increases vulnerability to attacks like credential stuffing or brute force. • **Reputation Damage**: Exposing sensitive data can damage the app's reputation and user trust. User impacts : • **Privacy Breach**: Exposes users' personal data.

	• **Account Compromise**: Attackers can access user accounts. • **Data Loss**: Sensitive user data may be deleted or leaked. • **Loss of Trust**: Users may abandon the app due to security concerns.
Mitigations	• **Avoid Hardcoding**: Do not embed sensitive information like credentials or API keys directly in the code. • **Use Secure Storage**: Store sensitive data in secure environments, such as encrypted storage or secure vaults. • **Fetch Dynamically**: Retrieve sensitive data dynamically from a secure server or API during runtime. • **Apply Encryption**: Encrypt sensitive data at rest and in transit to prevent unauthorized access.

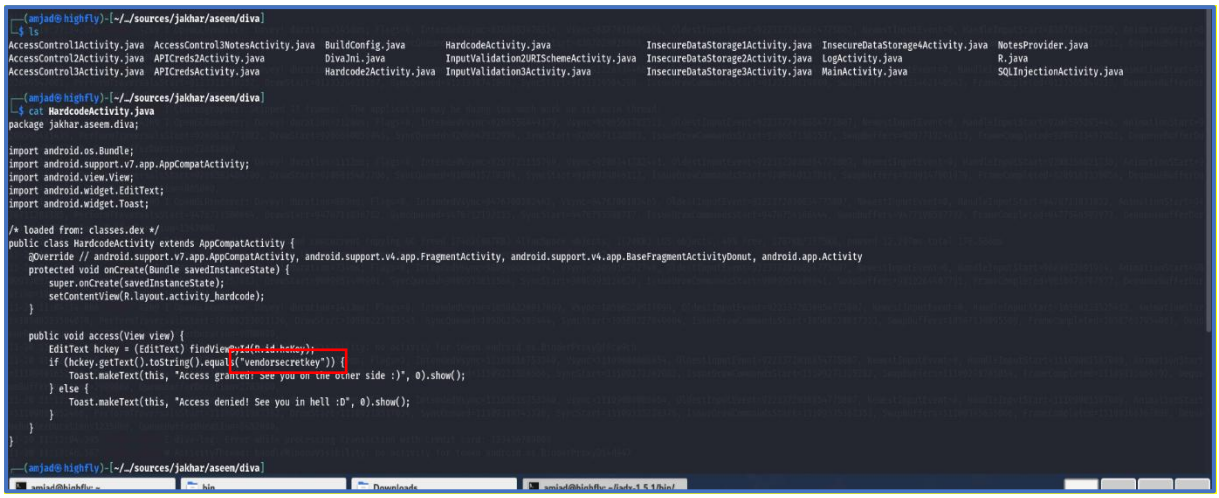
Proof of concept :

Firstly we have jadx-CLI : https://github.com/skylot/jadx/releases/tag/v1.5.1

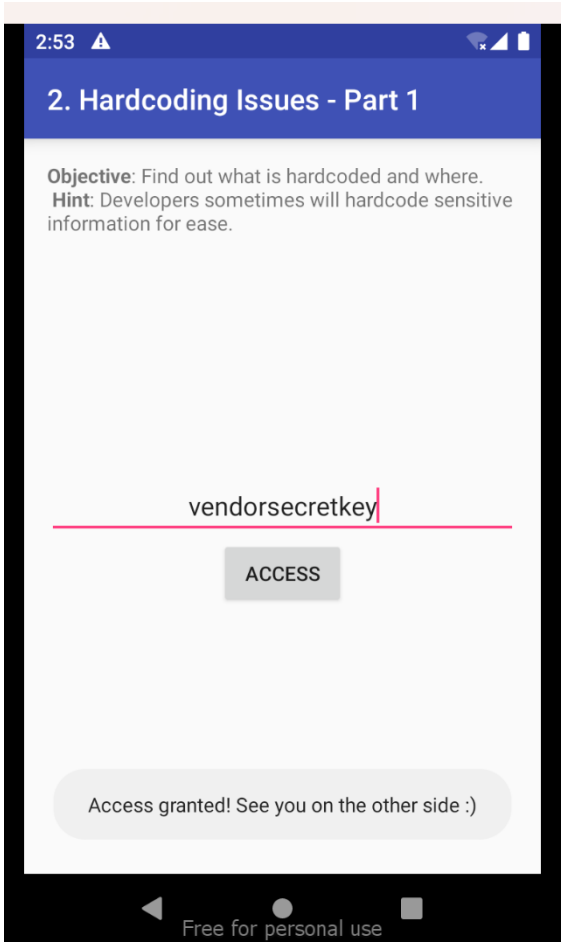
Then ,



Take all source code and check it ,



```
┌──(amjad㉿highFly)-[~/../sources/jakhar/aseem/diva]
└─$ ls
AccessControl1Activity.java  AccessControl3NotesActivity.java  BuildConfig.java       HardcodeActivity.java              InsecureDataStorage1Activity.java  InsecureDataStorage4Activity.java  NotesProvider.java
AccessControl2Activity.java  APICreds2Activity.java            DivaJni.java           InputValidation2URISchemeActivity.java  InsecureDataStorage2Activity.java  LogActivity.java                   R.java
AccessControl3Activity.java  APICredsActivity.java             Hardcode2Activity.java  InputValidation3Activity.java      InsecureDataStorage3Activity.java  MainActivity.java                  SQLInjectionActivity.java

┌──(amjad㉿highFly)-[~/../sources/jakhar/aseem/diva]
└─$ cat HardcodeActivity.java
package jakhar.aseem.diva;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* loaded from: classes.dex */
public class HardcodeActivity extends AppCompatActivity {
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivityDonut, android.app.Activity
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        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();
        }
    }
}

┌──(amjad㉿highFly)-[~/../sources/jakhar/aseem/diva]
```



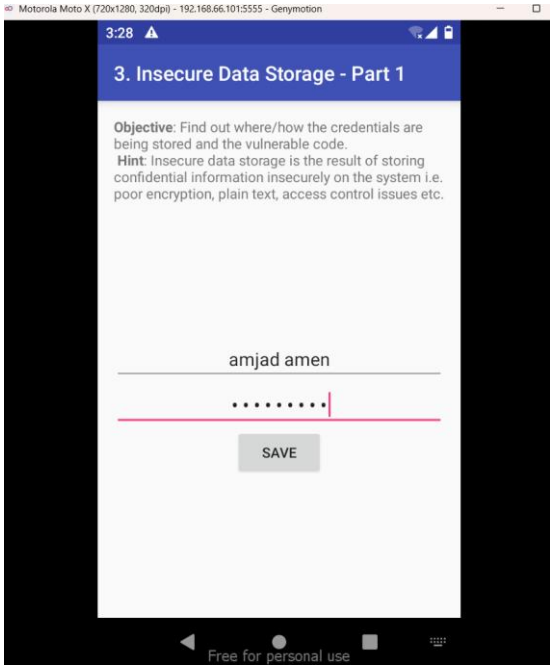
5.3 Sensitive Data stored in shared preferances

Description	This vulnerability arises when sensitive information, such as user credentials, tokens, or personal data, is stored insecurely on the device. Local storage mechanisms like Shared Preferences, SQLite databases, or temporary files .in here we take credentials in shared preferances
Impacts	Technical impact: • **Data Leakage**: Sensitive information (e.g., credentials, tokens, personal data) can be easily accessed by attackers through reverse engineering or device compromise. • **Unauthorized Access**: Exposed credentials or tokens can allow attackers to gain unauthorized access to backend systems or user accounts. • **Privilege Escalation**: Attackers may misuse stored data to escalate privileges within the application or system. User impacts: • **Privacy Breach**: Users' sensitive information, such as personal details or financial data, can be exposed. • **Account Compromise**: Attackers can gain unauthorized access to user accounts. • **Identity Theft**: Exposed data may be misused for fraudulent activities like identity theft.
Mitigations	• **Avoid Storing Sensitive Data Locally**: Only store non-sensitive information on the device if possible. • **Use Secure Storage APIs**: Utilize platform-provided secure storage mechanisms, such as Android's

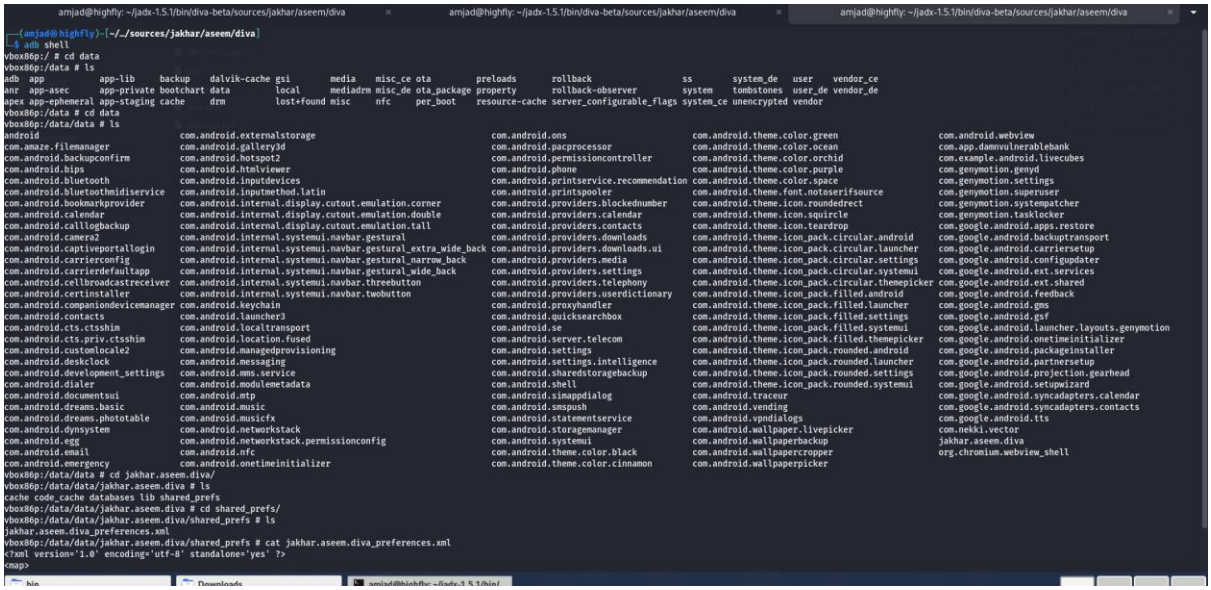
	Encrypted Shared Preferences or iOS Keychain. • **Encrypt Stored Data**: Implement strong encryption for all sensitive data stored locally, including files, databases, and preferences.

Proof of concept :

Fill the values ,



adb shell command : **adb shell**



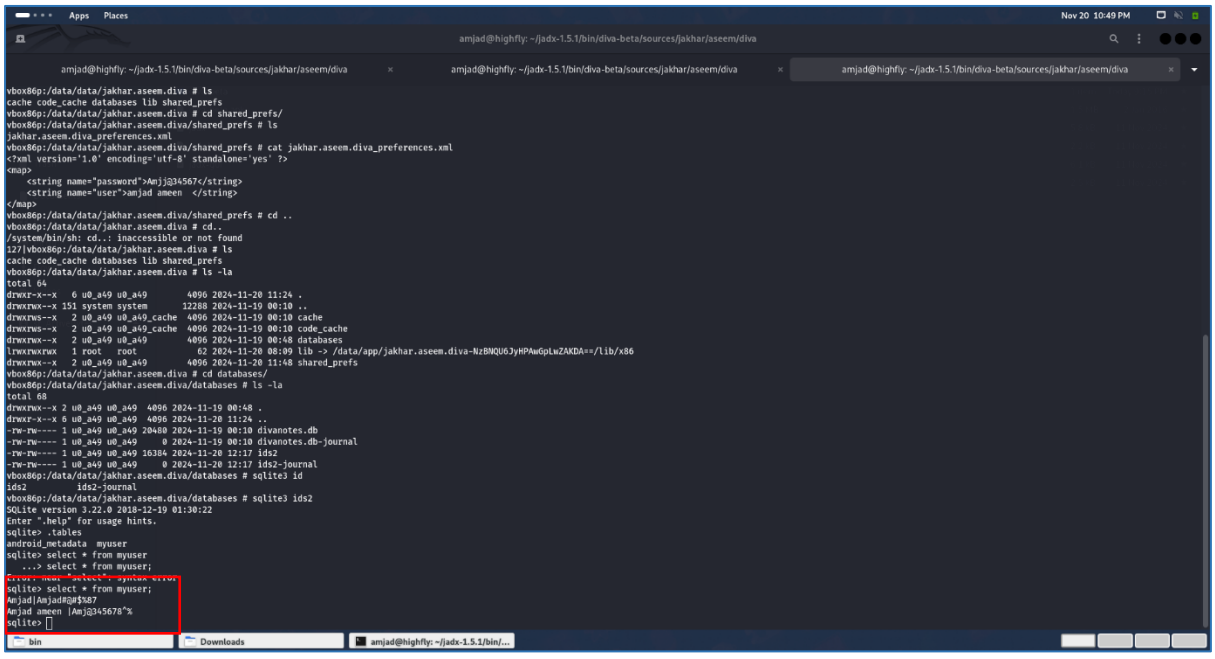
Take a xml file and read it ,use cat command

5.4 Sensitive Data Stored in local storage

Descriptions	This vulnerability occurs when sensitive information, such as user credentials, tokens, or personal data, is stored insecurely in local storage mechanisms like Shared Preferences, SQLite databases, or temporary files. Without proper encryption or security controls, this data becomes accessible to attackers through reverse engineering or device compromise.in here we found the credentials in a databases
Impacts	Technical impacts: • **Data Exposure**: Sensitive information, like credentials or tokens, can be easily extracted by attackers.

	• **Unauthorized Access**: Exposed data may allow attackers to gain access to user accounts or backend systems. • **Privilege Escalation**: Attackers can use the exposed data to escalate their access rights within the application or system.
Mitigations	• **Use Encrypted Shared Preferences**: Leverage Android's Encrypted Shared Preferences to securely store sensitive data. • **Avoid Storing Sensitive Data Locally**: Store sensitive information, such as tokens or credentials, on secure servers instead of local storage. • **Implement Strong Encryption**: Encrypt all sensitive data before storing it in Shared Preferences.

Proof of concept :

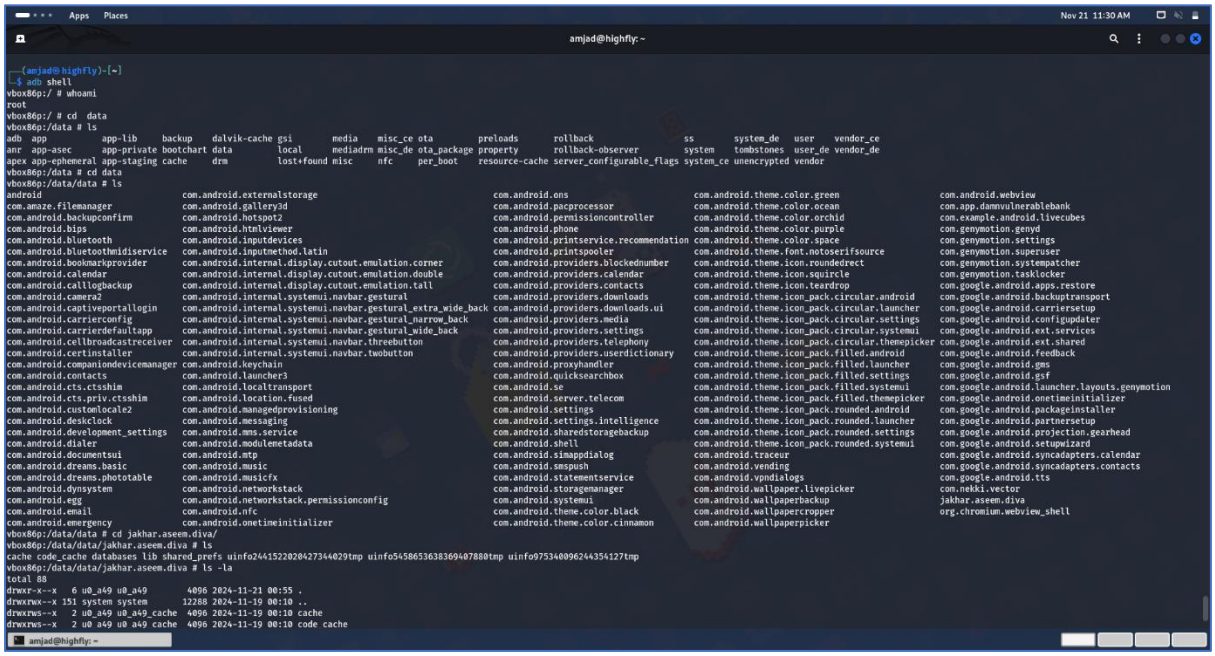


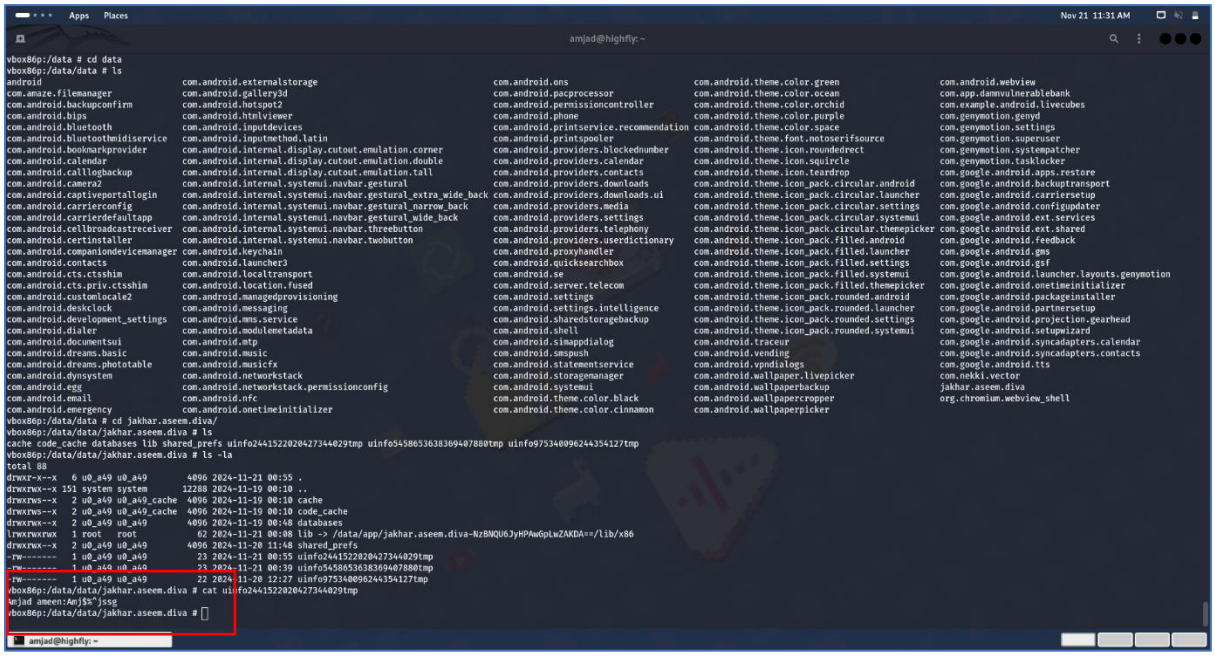
In this stage we are exploiting the sqLite and take the credentials ,

5.5 Sensitive Data Stored in Temporary Files

Description	During the penetration testing of the DIVA (Damn Insecure and Vulnerable App) mobile application, sensitive credentials were discovered in temporary files. This vulnerability arises when the application improperly stores sensitive data, such as login credentials or tokens, in temporary directories without adequate security measures.
impacts	• **Credential Compromise**: Attackers can access sensitive credentials (e.g., username/password) from temporary files. • **Privilege Escalation**: Leaked admin credentials could give attackers full control over the app's backend. • **Data Leakage**: Sensitive user or application data is exposed, increasing security risks. • **Exploitation by Malware**: Malware could target and extract sensitive data from temporary files.
mitigation	• Avoid Temporary File Storage: Keep sensitive data in memory instead of writing it to disk. • Use Secure Storage: Utilize secure APIs like Android Keystore or iOS Keychain for sensitive data. • Encrypt Data: Encrypt sensitive data before storing it in temporary files if storage is necessary. • Set File Permissions: Apply strict access controls to prevent unauthorized access to temporary files.

Proof of concept :





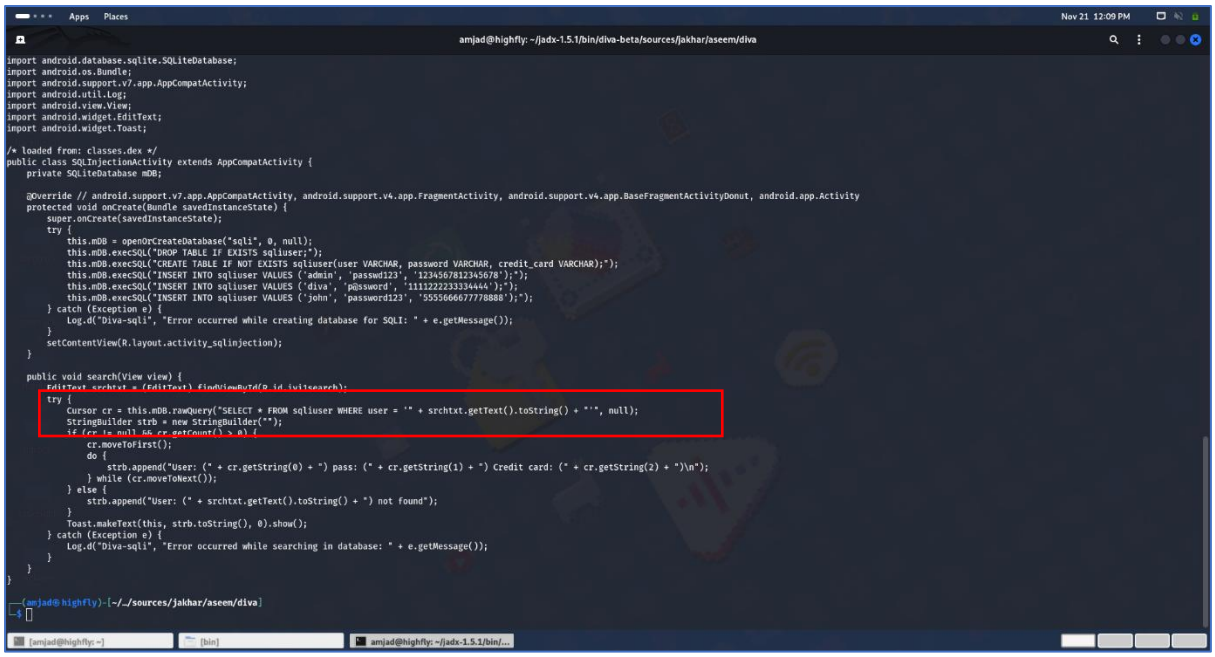
In here we take the credentials in a temp file and compromise ,

5.6 input validation issue (sql injection)

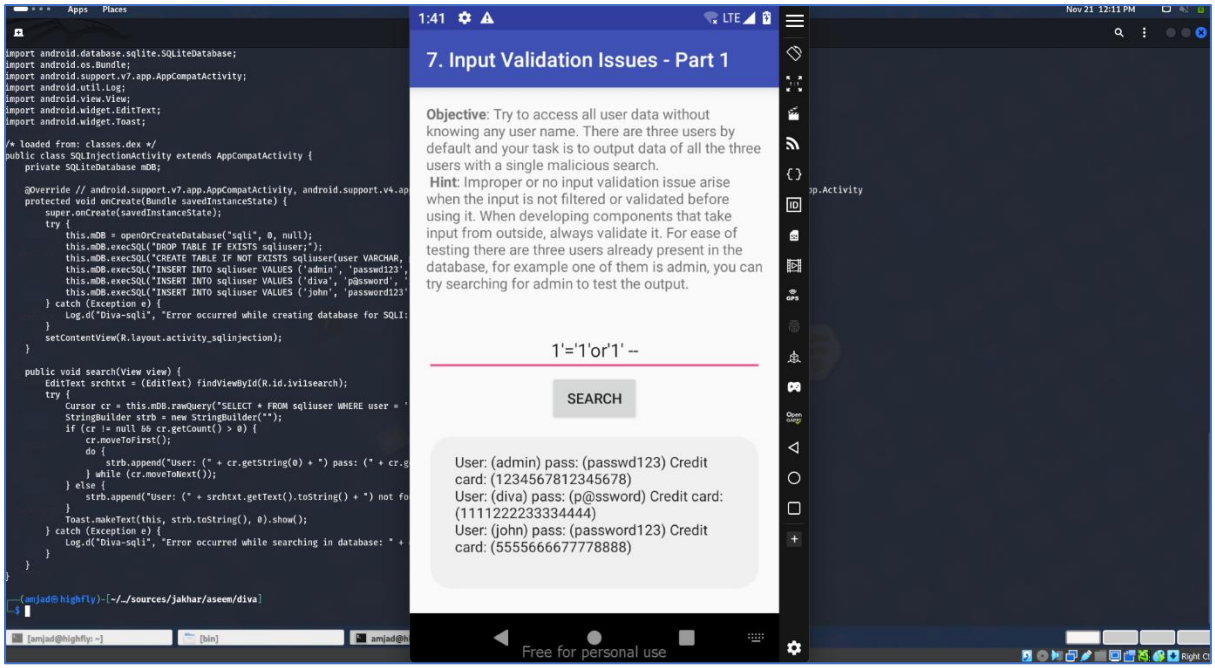
Description	During penetration testing of the DIVA (Damn Insecure and Vulnerable App) mobile application, an SQL Injection vulnerability was identified due to inadequate input validation. User input was found to be directly concatenated into SQL queries, allowing attackers to manipulate the database and extract sensitive information. Credentials were retrieved by exploiting this vulnerability. In here we using ' OR '1'='1 query for compromise it .
Impacts	Technical impacts : • Unauthorized Database Access: Attackers can gain unauthorized access to the database and sensitive information. • Data Manipulation: Attackers can modify, delete, or add records in the database. • Data Leakage: Sensitive user data (e.g., credentials) can be exposed to attackers. • Privilege Escalation: Exploitation may lead to unauthorized access to higher privilege levels. • System Compromise: Attackers could execute arbitrary commands or escalate attacks via database vulnerabilities. User Impacts : • Credential Theft: User credentials, such as usernames and passwords, may be stolen. • Account Takeover: Attackers could impersonate users and access their accounts. • Loss of Privacy: Sensitive personal information could be exposed or leaked.
Mitigations	• **Use Parameterized Queries**: Always use prepared statements with parameterized queries to prevent SQL Injection.

	• **Validate and Sanitize Inputs**: Ensure all user inputs are validated and sanitized to allow only expected data types and formats. • **Escape Special Characters**: Properly escape special characters (like ', ", ;) in user input to prevent query manipulation.

Proof of concept :



Then inject the query for search bar , then



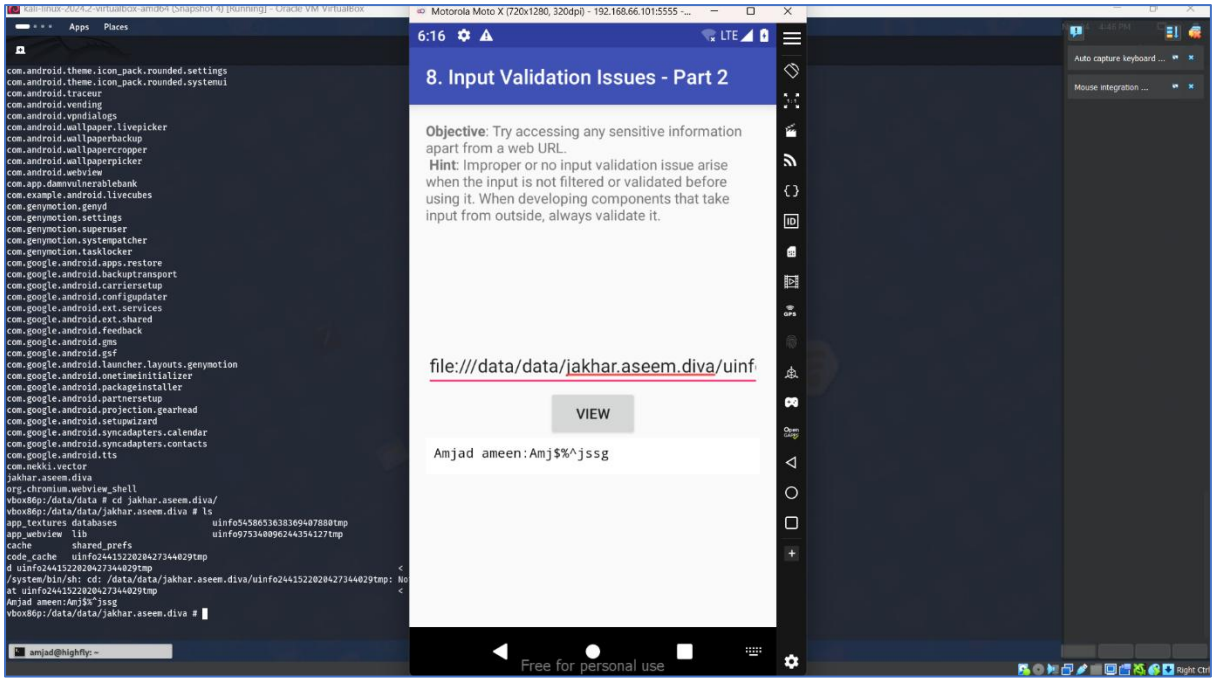
Take all usernames , passwords and sensitive informations

5.7 input validation issue (URL injection)

Description	An URL Injection vulnerability was identified. This issue occurs due to insufficient input validation, allowing attackers to manipulate URL parameters. By injecting malicious URL inputs, attackers could access unauthorized resources or perform actions like retrieving sensitive data, including user credentials.in here we take the credentials
Impacts	Technical impacts: • Unauthorized Data Access: Attackers can manipulate URLs to access restricted application resources. • Sensitive Data Exposure: Credentials and other sensitive information may

	be leaked or stolen from temporary files. • Application Integrity Compromise: URL manipulation can alter the application's expected behavior, potentially leading to data corruption or unauthorized actions. • System Vulnerability: An unpatched URL injection vulnerability can allow further attacks, like code execution or privilege escalation. User Impacts: • Credential Theft: User credentials, including usernames and passwords, can be exposed. • Account Compromise: Attackers can impersonate legitimate users and access sensitive data. • Privacy Violations: Users' personal information may be leaked or accessed without permission.
Mitigations	• **Validate and Sanitize Inputs**: Ensure all URL parameters are validated and sanitized. • **Use Whitelisting**: Only accept known safe values for URL parameters. • **Encode URL Inputs**: Properly encode user inputs before adding them to URLs. • **Avoid Storing Sensitive Data in Temp Files**: Do not store credentials or sensitive data in temporary files.

Proof of concept :



Use URL : file://data/data/jakhar.aseem.diva/uinfo244152202042734402 9tmp

6. Conclusion

The penetration testing of the DIVA mobile application uncovered seven vulnerabilities, including one critical (SQL Injection), three high (insecure logging, sensitive data stored in shared preferences and local storage), and three medium (sensitive data in temporary files, hard-coded credentials, and URL injection). These vulnerabilities pose significant risks to user data and application security. Immediate remediation is recommended, focusing on secure data storage, proper input validation, removal of hard-coded sensitive information, and reviewing logging practices to strengthen the application's security posture and protect against exploitation.