جامعة الملك سعود
King Saud University
College of Computer & Information Sciences
Information Technology Department

IT371 Application Security

-Project-

First Semester 2025-2026

External Application Penetration Testing Technical Report

For

Diva Application

Supervised By: Dr.Reem Albarrak

Document Info

Item	Description
Document Title	External Application Penetration Testing Technical Report for Diva Application
Requestor	L. Reem Albarrak for IT371 course project
Author/s	Aseel Almubaddel-Tala Alsheail-Norah Aldalal-Alanoud Aloraydi
Date Created	November 29, 2025

Table of Content

1 Executive Summary

1.1 *Introduction*

This report presents the results of an external mobile application penetration test conducted on the DIVA (Damn Insecure and Vulnerable Application) as part of the IT371 – Application Security course. The primary objective of this assessment was to identify, analyze, and document security vulnerabilities intentionally embedded within the application. Since DIVA is designed as an educational tool, the testing process focused on understanding common mobile application security flaws and applying industry-standard testing methodologies to detect and validate them.

Throughout the engagement, the assessment team simulated real-world attack techniques to evaluate the security posture of the application across multiple components, including input handling, data storage, logging practices, access control, and credential management. The findings documented in this report reflect practical vulnerabilities that can occur in production-level applications when secure coding principles are not properly implemented.

1.2 *Scope*

The specific scope of this project includes performing the Application Penetration test for the specified duration on the below mentioned applications.

Application Name	Platform	Version	Environment	Approach
DivaApplication	**Android**	**1**	**Mac OS + Windows**	**White Box Penetration Testing**

1.3 *Risk Rating*

The risk rating for the issues and their impact on the operation of the organization is explained in the table 1 below. The overall risk rating reported will be based on vulnerability identification with its potential to be exploited by adversaries.

In general, the following factors were considered to arrive at the risk rating for vulnerability:

- Technical Impact: The extent to which an attacker may gain access to a system and the severity of it on the application. This metric will take the security triad CIA (Confidentiality, Integrity and Availability) values into account.
- Likelihood: This metric will take the Popularity and Simplicity of an exploit into consideration.
    - Popularity describes the existing or potential frequency of exploitation of the vulnerability.
    - Simplicity is the amount of effort required to exploit the vulnerability.

Overall Risk Severity				
Technical Impact (Confidentiality, Integrity, Availability)	HIGH	MEDIUM	HIGH	CRITICAL
	MEDIUM	LOW	MEDIUM	HIGH
	LOW	INFO	LOW	MEDIUM
		LOW	MEDIUM	HIGH
		Likelihood (Popularity and Simplicity)		

Table 1 Risk Severity

1.4 *Threat Security Level*

Vulnerabilities are categorized as **Critical**, **High**, **Medium**, **Low** and **Informational**.

Critical: Severe Impact on the affected application. They require immediate attention and resolution. Successful exploitation may provide the attacker **access to critical data**.

High: Severe Impact on the affected application. They require immediate attention. They are relatively easy for attackers to exploit and may provide them with **full control of the affected application.**

Medium: Moderate impact on the affected application. They are often **harder to exploit** and may not provide the same access to affected application.

Low: Limited impact on the affected application. They provide information to attackers that may assist them in mounting **subsequent attacks on the affected applications.** These should also be fixed in a timely manner, but are not as urgent as the other vulnerabilities.

Informational: It exposes information that target stake holders simply need to be aware of. These are for findings that are very difficult to exploit in practice.

1.5 *Summary Table*

The table below shows the summary of vulnerabilities disclosed during the Penetration Testing.

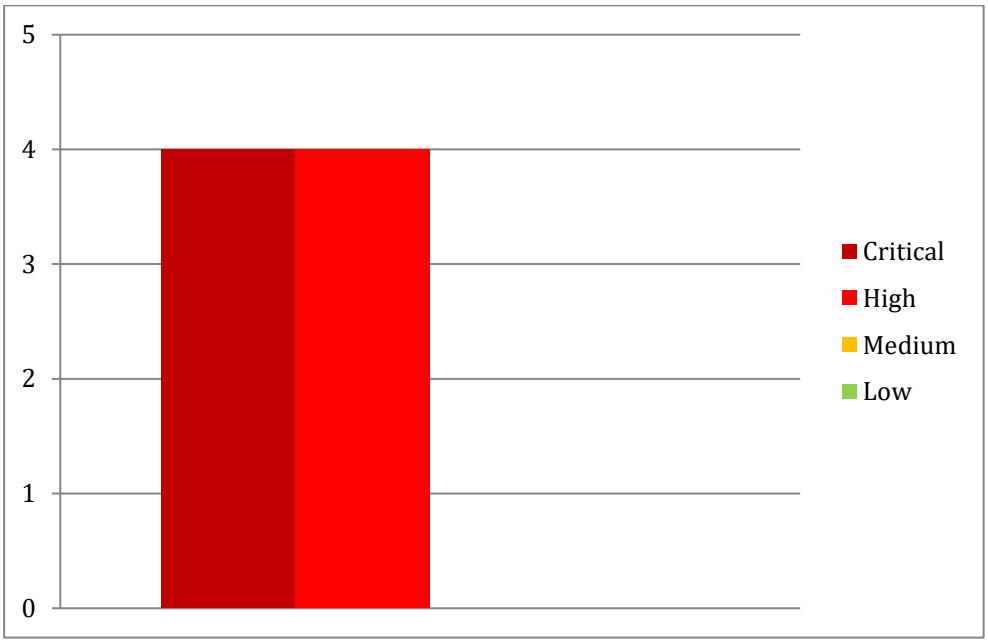
Mobile Application Penetration Testing

Critical	High	Medium	Low
4	4	-	-

1.6 *Summary Graph*

The following bar graph highlights the total number of vulnerabilities discovered during the penetration testing.

Figure 1 Application Penetration Testing



1.7 *Key Findings*

No.	Vulnerabilities Discovered	Platform	Severity Level
1	Hardcoding issues- Part 1	Android	**High**
		IOS	
2	Input Validation Issues – Part 1	Android	**Critical**
		IOS	
3	Input Validation Issues – Part 2	Android	**High**
		Windows	
4	Insecure Data Storage – Part 1	Android	**High**
		Windows	
5	Insecure Data Storage – Part 2	Android	**Critical**
		Windows	
6	Insecure Data Storage – Part 3	Android	**High**
		Windows	
7	Insecure Logging	Android	**Critical**
		Windows	
8	Access Control Issues - Part 1	Android	**Critical**
		Windows	

2 Conclusion

The penetration testing process revealed multiple critical and high-severity vulnerabilities within the application, indicating gaps in secure coding practices and defensive controls. These issues primarily relate to insecure data handling, weak input validation, improper access control, and unsafe logging mechanisms.

To strengthen the application's security posture, the identified vulnerabilities should be remediated according to their severity level, with priority given to critical findings followed by high-risk issues. The remediation guidance included in each section of this report is based on established industry best practices and should be applied where appropriate and technically feasible.

By implementing the recommended security measures and continuing to review the application against recognized standards such as OWASP Mobile Security, the overall resilience, reliability, and protection of user data can be significantly improved.

3 Methodology

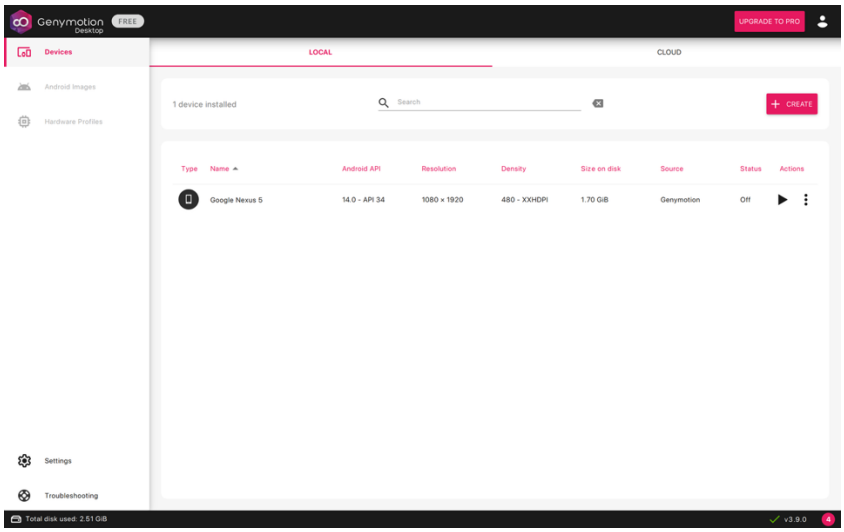
Our methodology on Mobile penetration testing is based on Mobile Open Web Application Security Project (OWASP); our assessment methodology to carry out the mobile penetration testing includes the following :

3.1 *Planning and Preparation:*

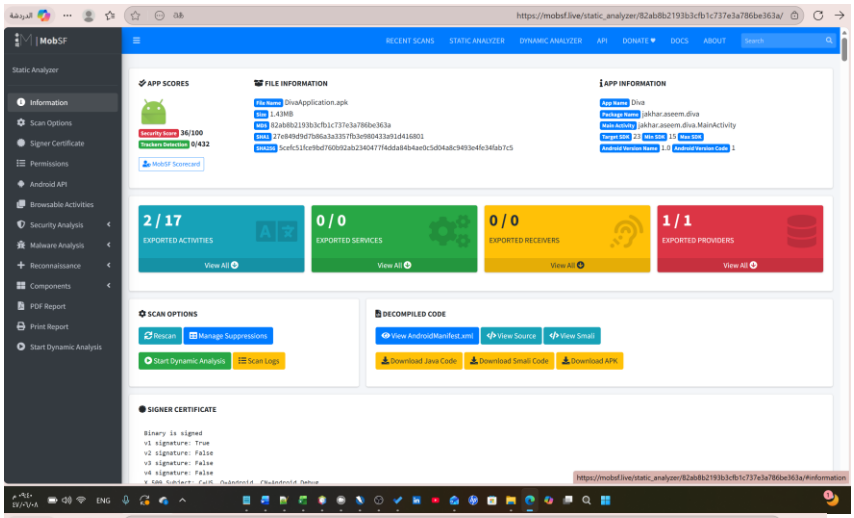
In this phase, we set up the environment required for penetration testing.

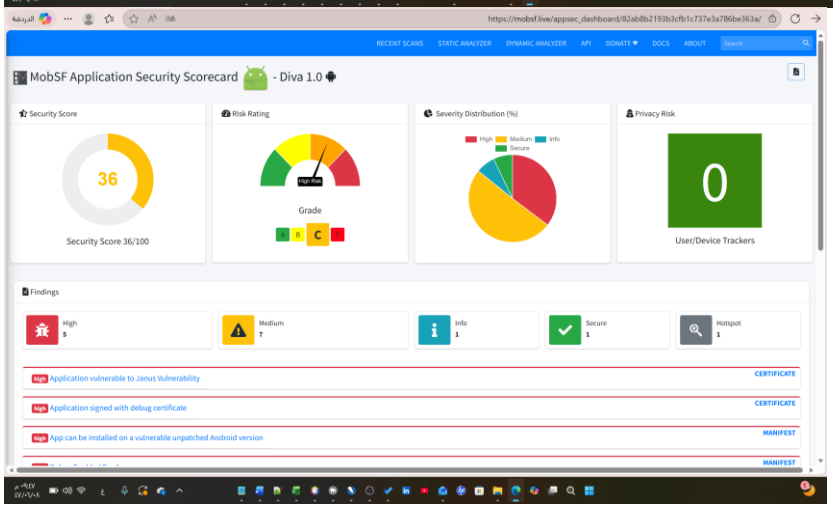
- **Genymotion**

Genymotion is an Android virtual device emulator that allows us to simulate a real Android environment. It was used to install and run the DIVA APK in a controlled virtual environment on macOS. The emulator provides device simulation capabilities that allow us to interact with the app as if it were running on a physical device.



- **MobSF (Mobile Security Framework)**

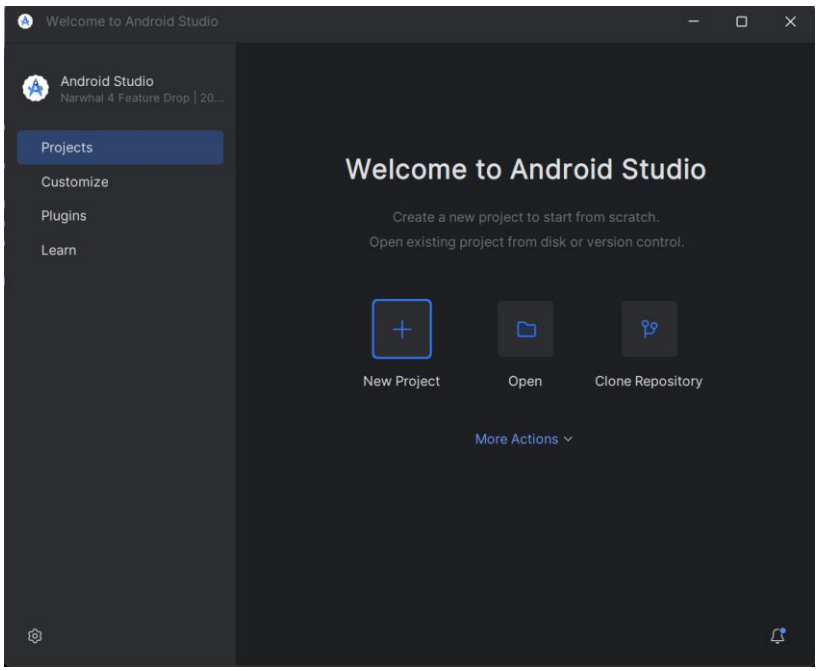


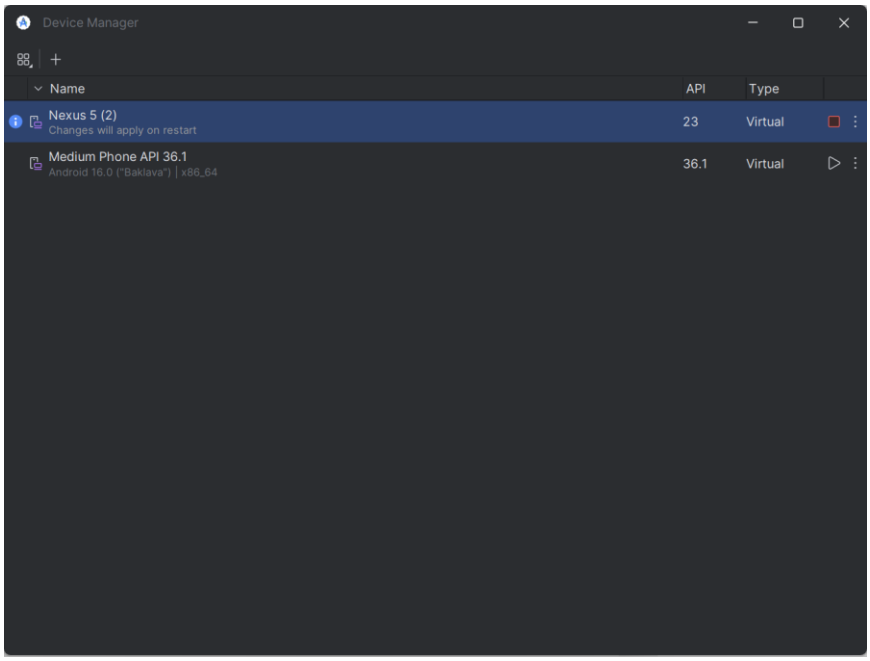


Mobile Security Framework (MobSF) is an automated security testing platform used for comprehensive static and dynamic analysis of mobile applications. We utilized the online MobSF Live instance (mobsf.live) to perform automated security scanning of the DIVA APK file. The tool provided in-depth vulnerability assessment, code analysis, and security scoring, complementing our manual testing efforts and helping identify potential security issues through automated pattern detection.

- **Android Studio Emulator**

Android Studio is an official Android development environment that includes a built-in emulator. It was used to install and run the DIVA APK on a virtual Android device directly on a Windows machine. The emulator allowed interaction with the app and supported the penetration testing process similarly to a real device.



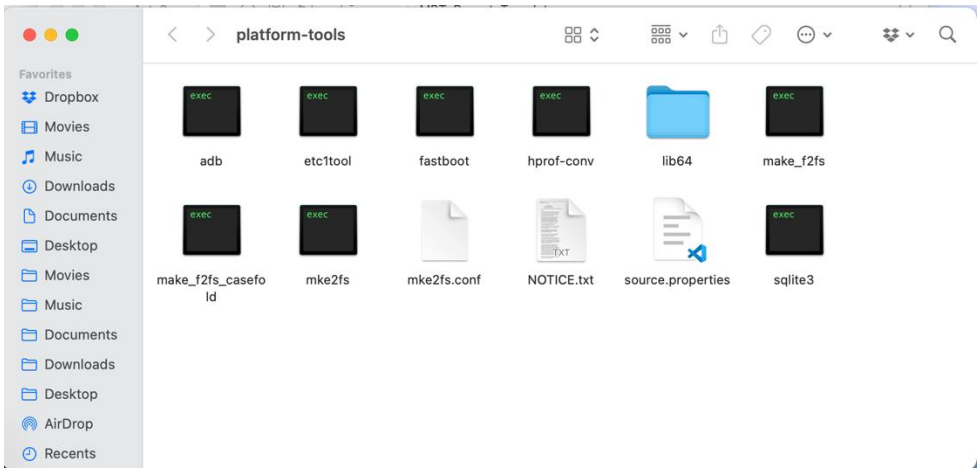


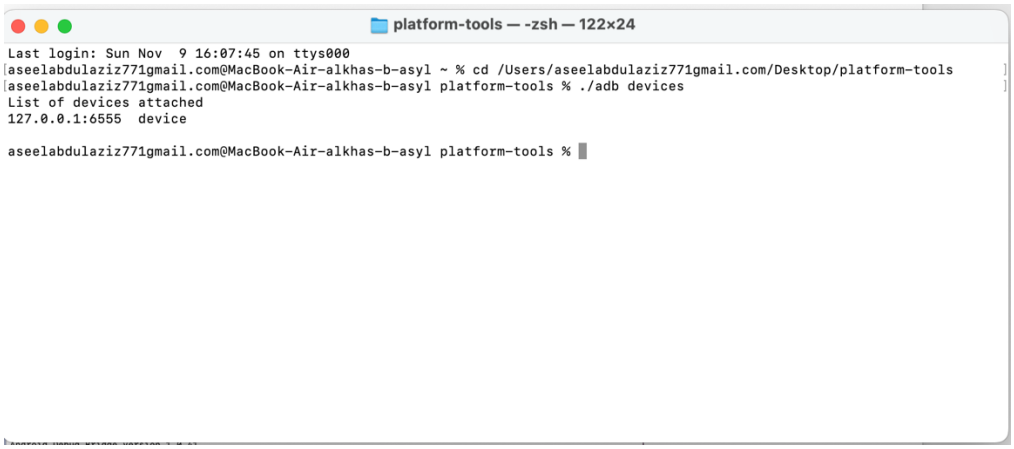
- **ADB (Android Debug Bridge)**

ADB is a command-line tool used to communicate with Android devices. We installed it using the Android SDK Platform Tools. It was used to access app files, run shell commands, and extract data such as logs or shared preferences.

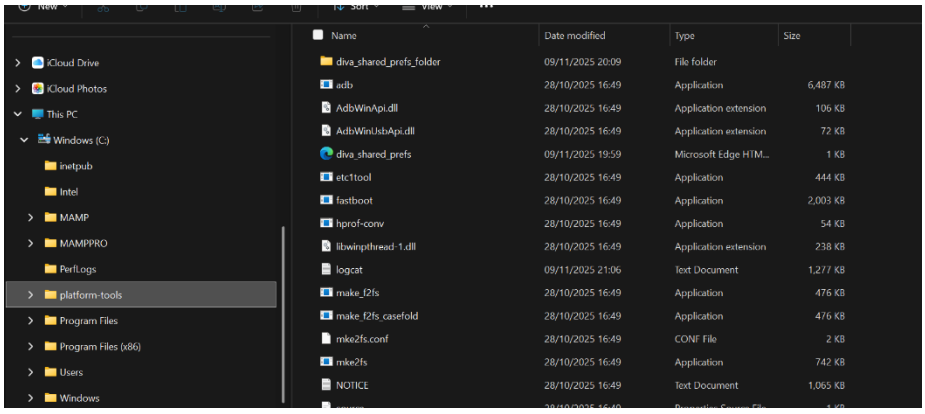
- Installation on macOS:

We downloaded the Android SDK platform-tools, opened the terminal in that directory, and connected to the Genymotion emulator using ./adb devices.

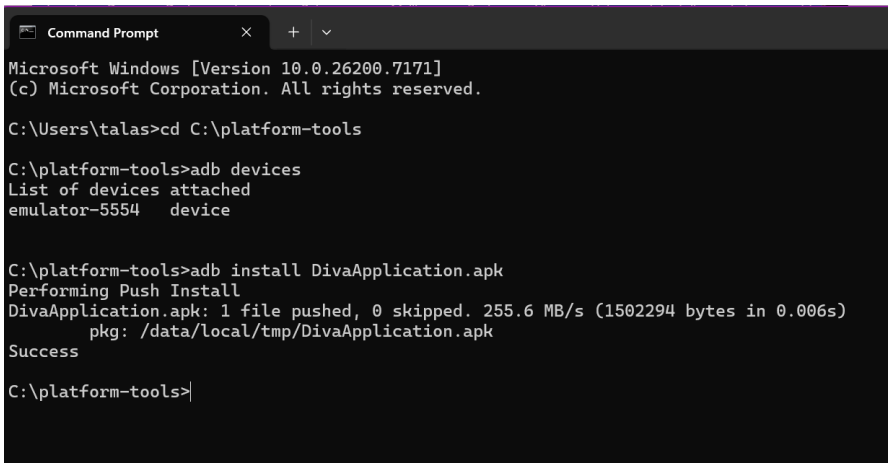




- Installation on Windows:



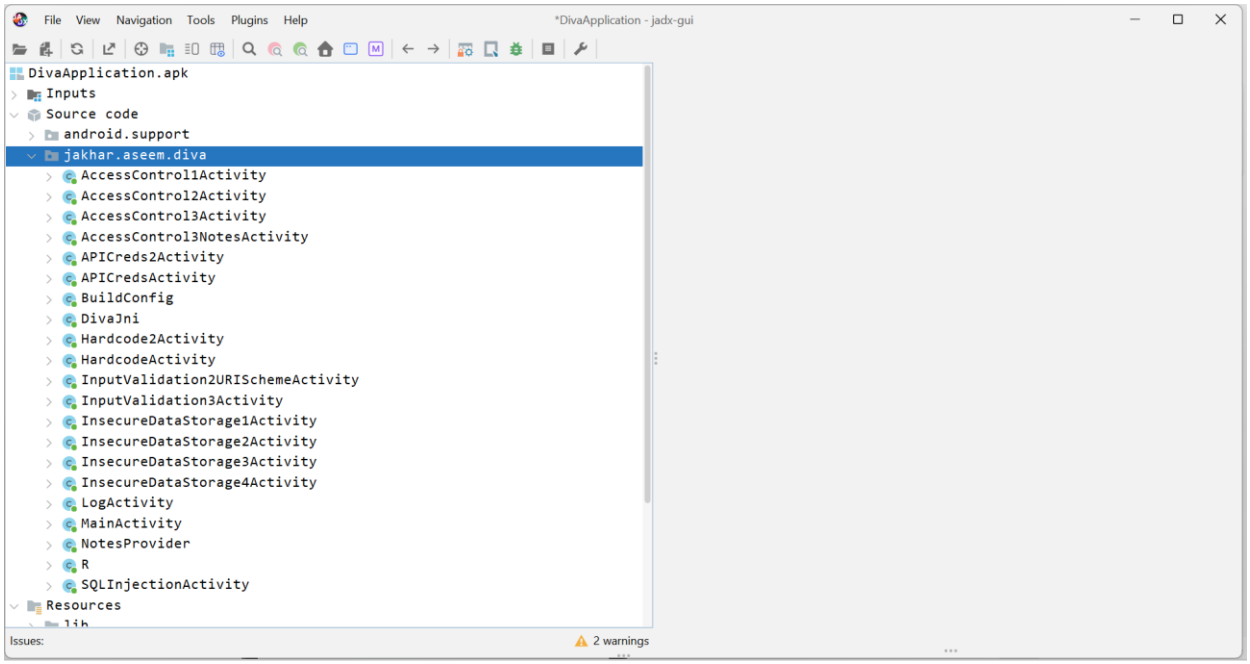
Extract platforms tools folder that Containing the ADB file.



Installing DivaApplication.apk using ADB commands via the Command Prompt.

- **JADX-GUI**

JADX-GUI is a decompilation tool used to extract and review the source code of the APK file. It was used to analyze the internal structure of the DIVA application and identify insecure coding practices and potential vulnerabilities.



3.2 *Information Gathering:*

During this phase, both static and dynamic analysis techniques were performed to understand how the DIVA application functions and handles data. Static analysis was conducted using **JADX-GUI** to decompile the APK and review key components such as the manifest file, activities, and source code structure. This helped identify elements including hardcoded credentials, URI handling, SQL query practices, and exported components.
Dynamic testing was carried out using the **Genymotion Emulator on macOS** and the **Android Studio Emulator on Windows**, allowing controlled execution of the application in different environments. In addition, the **Android Debug Bridge (ADB)** was used to interact with the application storage, extract files, monitor logs, and verify whether sensitive information could be accessed or manipulated.
Through these tools and testing methods, the team gathered evidence confirming several security weaknesses, including insecure data storage across multiple mechanisms, improper logging of sensitive information, weak input validation, and exposed components that could be accessed without authorization.

3.3 *Vulnerability Analysis:*

During this phase, we performed in-depth vulnerability identification and validation through comprehensive static and dynamic analysis approaches. Static analysis involved decompiling the APK using JADX-GUI to examine source code structure, review AndroidManifest.xml for security configurations, and identify insecure coding patterns such as hardcoded credentials and improper logging practices. This allowed us to understand the application's security posture from a code-level perspective. For dynamic validation, we utilized ADB commands to test real-world exploit scenarios. This included launching activities externally to validate access control issues and monitoring system logs to confirm sensitive information exposure. This combined approach ensured thorough vulnerability discovery and practical validation.

3.4 *Risk Modeling and Risk Rating:*

Once vulnerabilities were identified, each one was evaluated based on the defined scoring approach in the earlier sections of this report. The assessment considered the **technical impact** on confidentiality, integrity, and availability, as well as the **likelihood of exploitation**, including ease of execution and real-world feasibility. The risk categorization followed the OWASP-aligned model used in this project, resulting in each finding being classified as **Critical, High, Medium, or Low**. This prioritization supports clear remediation planning and reflects the potential severity to the application if exploited.

3.5 *Reporting:*

The final stage involved compiling the findings into a formal report structure. Each vulnerability was documented with its severity rating, technical description, observed impact, proof of concept, screenshots, affected platforms, and remediation recommendations. References to OWASP guidelines and relevant documentation were included to support corrective action. The report also presents summaries, visual graphs, and categorized tables to help stakeholders clearly understand the results and prioritize security improvements.

4 Detailed Findings

4.1 *Limitations*

During the Access Control Issues tasks, we encountered several limitations that constrained the depth of our analysis. One of the primary challenges was our limited understanding of certain Android internals—particularly how activities, permissions, and exported components interact through the manifest file. This knowledge gap made it harder to confidently assess how the application handles component exposure.

We also faced technical constraints related to the Android Debug Bridge (ADB) and emulator configuration. These issues occasionally prevented us from executing specific commands or fully testing component behavior, which slowed down our ability to validate some access control scenarios.

Additionally, because the DIVA application is a training environment that does not interact with real APIs or sensitive user data, it was difficult to fully assess the real-world impact of the vulnerabilities. The lack of actual backend systems or live data limited our ability to observe practical exploitation consequences or understand how these flaws would manifest in production-grade applications.

4.2 *Technical Description of Findings*

This section explains the details of the identified vulnerability along with technical impact, Proof of Concept, recommendations and references related to the vulnerability.

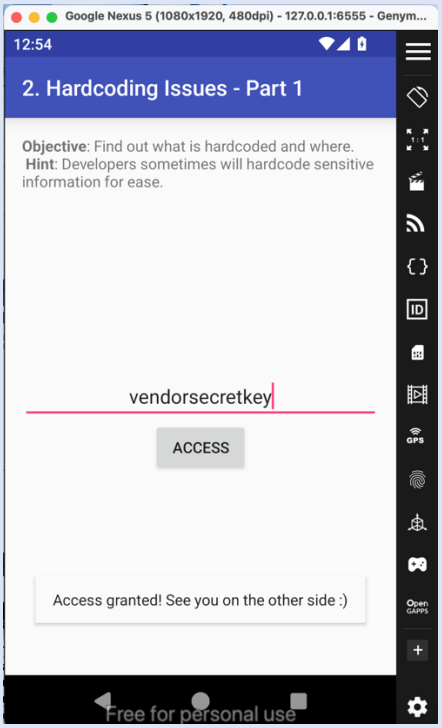
Severity Level	Risk Severity Level of the Vulnerability				
Technical Impact	Impact level	**Likelihood**		Popularity and Simplicity	
		Popularity	Popularity of the Exploit	**Simplicity**	Simplicity to Exploit
Observation	Our observation and description about the vulnerability				
Impact	Describes the possible Technical Impact if the vulnerability is Exploited				
Platform	Describe the platform				
Proof of Concept					
Evidence to support the finding					
Recommendation	High Level recommendation to mitigate the vulnerability				
OWASP Reference	OWASP Reference				
Reference	Reference related to the vulnerability				

4.2.1 Hardcoding issues – Part 1

<table>
<tr><td>Severity Level</td><td colspan="4">High</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">Hight</td><td>Likelihood</td><td colspan="2">Medium</td></tr>
<tr><td>Popularity</td><td>Hight</td><td>Simplicity</td><td>Medium</td></tr>
<tr><td>Observation</td><td colspan="4">A sensitive key (vendorsecretkey) was hardcoded directly into the application's source code. By exploring the HardcodeActivity.java class, we discovered that the key was hardcoded into the logic used for verifying access. The key was hardcoded in plaintext.</td></tr>
<tr><td>Impact</td><td colspan="4">An attacker with access to the APK can extract the hardcoded credentials and use them to gain unauthorized access to sensitive areas of the application.</td></tr>
<tr><td rowspan="2">Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="4">IOS</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
</table>

HardcodeActivity

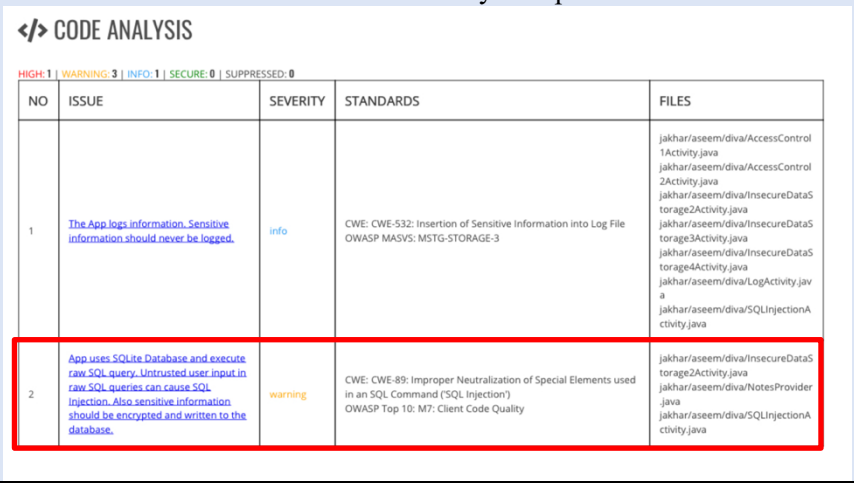
```
package jakhar.aseem.diva;

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.
    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();
        }
    }
}
```



We used the hardcoded key to gain an access, and it worked

Recommendation	Avoid hardcoding sensitive information such as keys, credentials, or tokens within the source code. Instead, use secure storage mechanisms like Android Keystore or retrieve it dynamically from a secured backend server after proper authentication and authorization or use a properly secured configuration file .
Reference	**https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage** **https://owasp.org/www-project-mobile-top-10/2023-risks/m1-improper-credential-usage**

4.2.2 Input Validation Issues – Part 1

Severity Level	**Critical**				
Technical Impact	Hight	**Likelihood**		Hight	
		Popularity	Hight	**Simplicity**	Hight
Observation	The app fails to validate input properly. User input is directly concatenated into SQL queries, as shown in the **SQLInjectionActivity.java** class. This allows attackers to inject SQL code that can extract sensitive user information from the database without proper authorization.				
Impact	Attackers may retrieve sensitive information (e.g., usernames, passwords, credit card numbers) or alter and delete records. It compromises confidentiality, integrity, and availability.				
Platform	**Android**				
	IOS				
Proof of Concept					
MobSF static analysis report 					

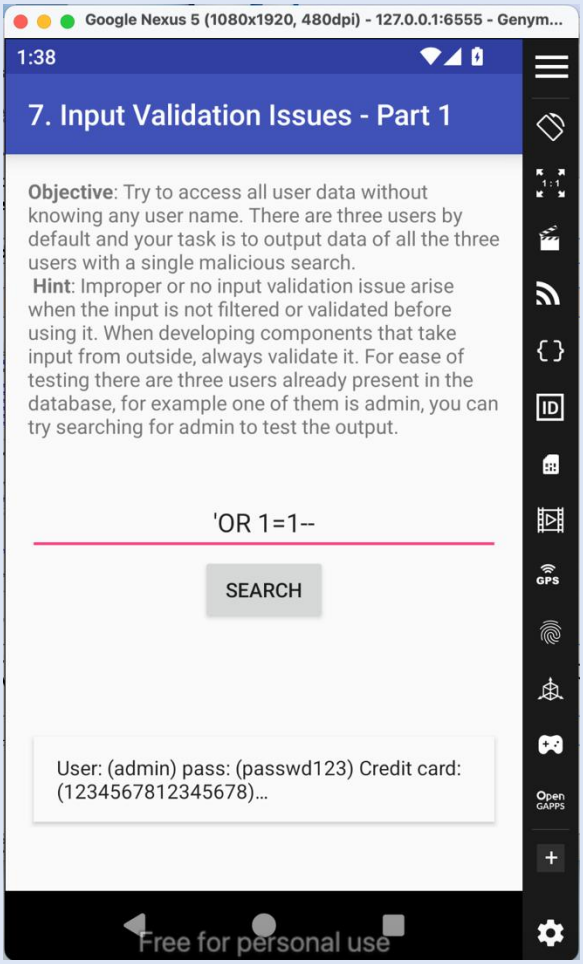
</> CODE ANALYSIS

HIGH: 1 | WARNING: 3 | INFO: 1 | SECURE: 0 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/LogActivity.java jakhar/aseem/diva/SQLInjectionActivity.java
2	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/NotesProvider.java jakhar/aseem/diva/SQLInjectionActivity.java

```
SQLInjectionActivity
protected void onCreate(Bundle savedInstanceState) throws SQLException {
    super.onCreate(savedInstanceState);
    try {
        this.mDB = openOrCreateDatabase("sqli", 0, null);
        this.mDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
        this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
        this.mDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'passwd123', '1234567812345678');");
        this.mDB.execSQL("INSERT INTO sqliuser VALUES ('diva', 'p@ssword', '1111222233334444');");
        this.mDB.execSQL("INSERT INTO sqliuser VALUES ('john', 'password123', '5555666677778888');");
    } catch (Exception e) {
        Log.d("Diva-sqli", "Error occurred while creating database for SQLi: " + e.getMessage());
    }
    setContentView(R.layout.activity_sqlinjection);
}

public void search(View view) {
    EditText srchtxt = (EditText) findViewById(R.id.ivi1search);
    try {
        Cursor cr = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);
        StringBuilder strb = new StringBuilder("");
        if (cr != null && cr.getCount() > 0) {
            cr.moveToFirst();
            do {
                strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")\n");
            } while (cr.moveToNext());
        } else {
            strb.append("User: (" + srchtxt.getText().toString() + ") not found");
        }
        Toast.makeText(this, strb.toString(), 0).show();
    } catch (Exception e) {
        Log.d("Diva-sqli", "Error occurred while searching in database: " + e.getMessage());
    }
}
}
```

The application does not perform any input validation it takes the user input as it is and insert it in the SQL query which will be then interpreted by the database engine as code



We entered the command ‘OR 1=1—and the query returned all records in sqliuser table

Recommendation	• Use prepared statements • Always validate and escape user inputs to make sure it doesn’t contain SQL syntax • Use stored procedures
Reference	https://owasp.org/www-community/attacks/SQL_Injection

4.2.3 Insecure Data Storage – Part 1

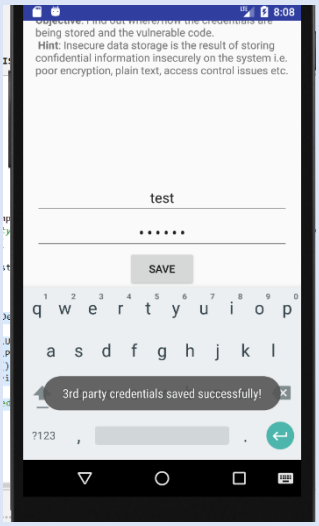
<table>
<tr><td>Severity Level</td><td colspan="4">High</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">Hight</td><td>Likelihood</td><td colspan="2">Medium</td></tr>
<tr><td>Popularity</td><td>Hight</td><td>Simplicity</td><td>Hight</td></tr>
<tr><td>Observation</td><td colspan="4">We unpacked the APK with JADX to inspect the application internals and discovered insecure handling of third-party credentials. In InsecureDataStorage1Activity the app writes the username and password into SharedPreferences using putString() without any encryption. To validate, we ran the app, entered test credentials, and used adb (run-as) to list and extract files from the app’s private storage. The pulled preferences file (jakhar.aseem.diva_preferences.xml) contained the submitted credentials in cleartext confirming a plaintext storage vulnerability that any local adversary with file access can exploit.</td></tr>
<tr><td>Impact</td><td colspan="4">• If an attacker gains physical or root access to the device, they can easily retrieve and use the stored credentials.
• This could result in unauthorized account access, data theft, or identity impersonation.
• It weakens user trust and violates secure coding standards recommended by OWASP.</td></tr>
<tr><td rowspan="2">Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="4">Windows</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
</table>

NO	ISSUE	SEVERITY	STANDARDS	FILES
3	Debug configuration enabled. Production builds must not be debuggable.	high	CWE: CWE-919: Weaknesses in Mobile Applications OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-RESILIENCE-2	jakhar/aseem/diva/BuildConfig.java
4	App creates temp file. Sensitive information should never be written into a temp file.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage3Activity.java
5	App can read/write to External Storage. Any App can read data written to External Storage.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage4Activity.java

MobSF flags insecure handling of sensitive data through temporary file storage.

```
/data/user/0/jakhar.aseem.diva/shared_prefs/: 1 file pulled, 0 skipped. 0.0 MB/s (152 bytes in 0.004s)
PS C:\platform-tools> # pull (optional if you already have file) then check and print summary
PS C:\platform-tools> .\adb.exe -s emulator-5554 shell "run-as jakhar.aseem.diva cat /data/user/0/jakhar.aseem.diva/shared_prefs/jakhar.aseem.diva_preferences.xml" > .\temp_prefs.xml
PS C:\platform-tools> if (Select-String -Path .\temp_prefs.xml -Pattern 'name="user"','name="password"') {
>>   Write-Output "Credentials found in plaintext in SharedPreferences XML file."
>>   Select-String -Path .\temp_prefs.xml -Pattern 'name="user"','name="password"'
>> } else {
>>   Write-Output "No plaintext credentials found in prefs file."
>> }
Credentials found in plaintext in SharedPreferences XML file.

temp_prefs.xml:5:    <string name="user">test</string>
temp_prefs.xml:7:    <string name="password">123456</string>


PS C:\platform-tools>
```

ADB access reveals stored values under /data/user/Jakhar.aseem.diva/shared_prefs/



The application stores sensitive credentials in plaintext within SharedPreferences (jakhar.aseem.diva_preferences.xml).
This insecure storage method allows unauthorized access on rooted devices, during debugging, or in misconfigured environments, leading to credential disclosure.

Recommendation	• **Avoid storing credentials or any sensitive data in plain text within SharedPreferences.** • **Use EncryptedSharedPreferences or Android KeyStore for secure local storage.** • **Apply encryption before saving credentials, and clear them after logout.**
Reference	• **OWASP Mobile Security Testing Guide — MSTG-STORAGE-1, MSTG-STORAGE-2** • **Android Developers — Data and File Storage Overview** • **OWASP Mobile Top 10 (2023) — M2: Insecure Data Storage** • **AndroidX Security Library — EncryptedSharedPreferences Documentation**

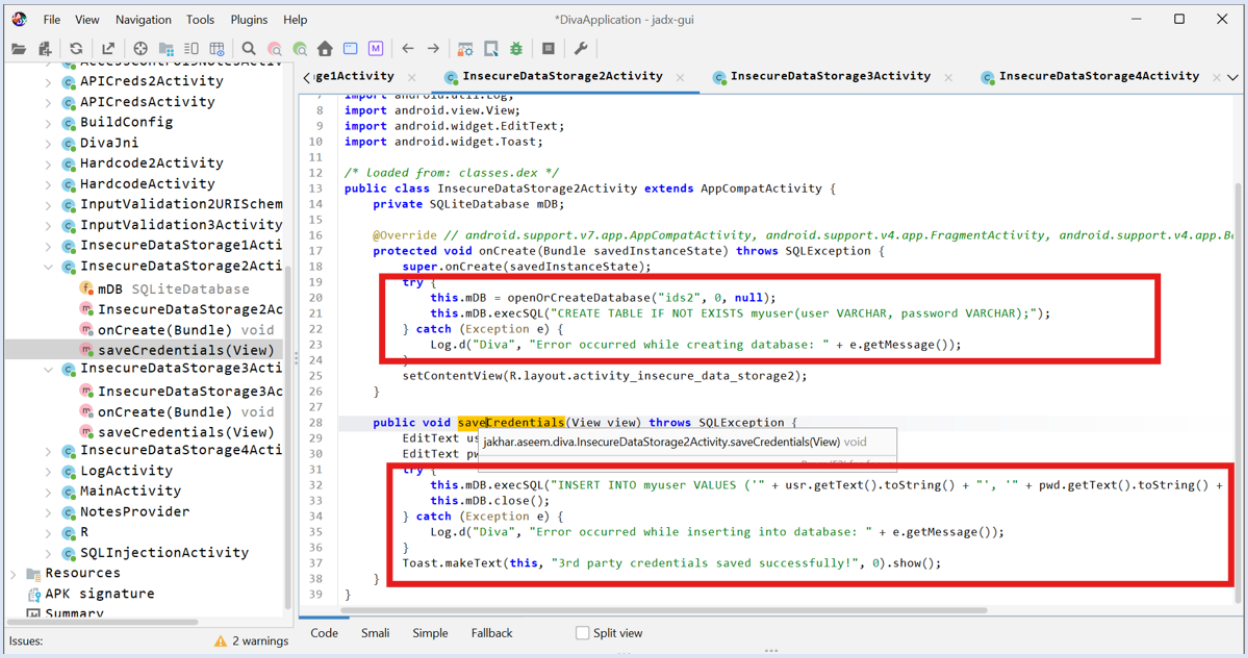
4.2.4 Insecure Data Storage – Part 2

Severity Level	**Critical**				
Technical Impact	High	**Likelihood**		**High**	
		Popularity	High	**Simplicity**	High
Observation	The application stores usernames and passwords in plaintext inside a local SQLite database (ids2) using openOrCreateDatabase() with insecure SQL concatenation, weak schema, no encryption, and no access controls.				
Impact	Credentials can be extracted through ADB, emulator access, backups, or rooted devices, enabling credential theft, SQL injection abuse, and potential account takeover.				
Platform	**Android**				
	Windows				
Proof of Concept					

CODE ANALYSIS

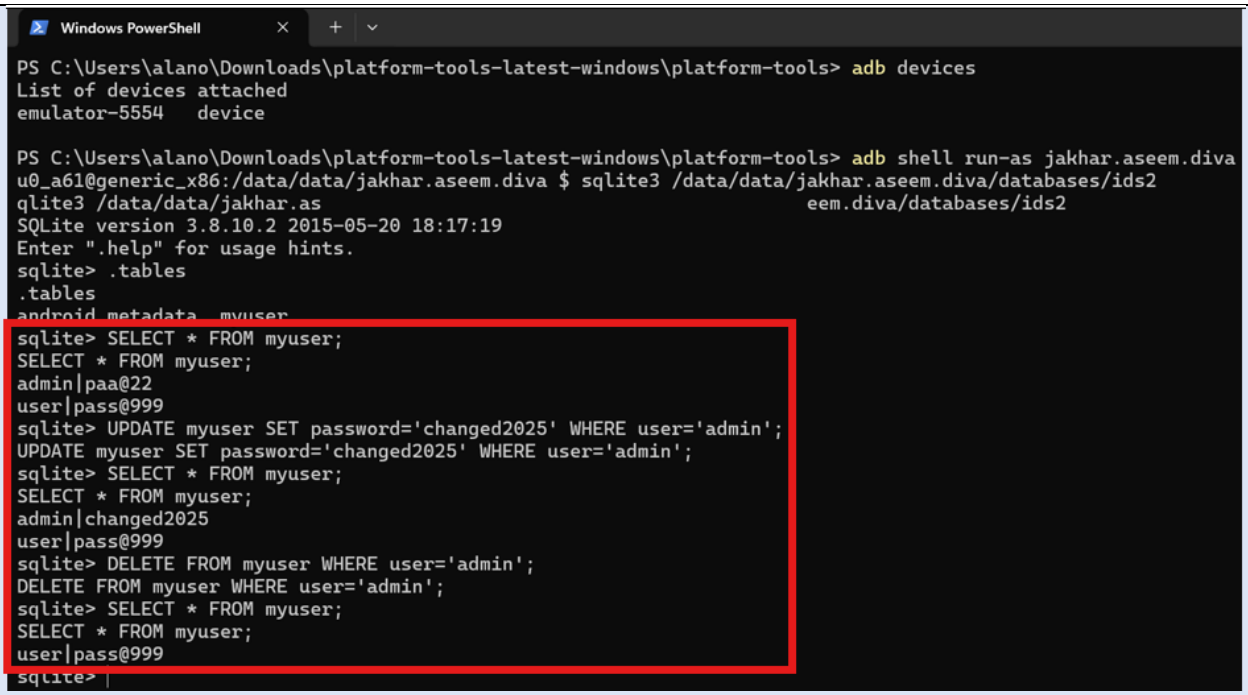
HIGH: 1 | WARNING: 2 | INFO: 1 | SECURE: 0 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/LogActivity.java jakhar/aseem/diva/SQLInjectionActivity.java
2	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/NotesProvider.java jakhar/aseem/diva/SQLInjectionActivity.java



Vulnerable code (the risky lines)

```
this.mDB = openOrCreateDatabase("ids2", 0, null);
this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
...
this.mDB.execSQL("INSERT INTO myuser VALUES ('" + usr.getText().toString() + "', '" +
pwd.getText().toString() + "');");
this.mDB.close();
```



The database was accessed using adb shell run-as on an Android emulator, demonstrating that the application stores credentials in plain text and allows direct modification without requiring root privileges.

This weakness exists because user input is directly concatenated into SQL statements and stored without protection. Implementing parameterized queries and encrypting stored credentials would prevent unauthorized database access and eliminate the SQL injection risk.

Recommendation	• Use parameterized queries. • Avoid storing plaintext credentials. • Encrypt the database (SQLCipher) and protect keys using Android Keystore.
Reference	**OWASP Mobile Top 10** **OWASP Mobile Security Testing Guide (MASTG) – MSTG-STORAGE-2** https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#logs https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04h-Testing-Code-Quality.md#injection-flaws-mstg-arch-2-and-mstg-platform-2

4.2.5 Insecure Data Storage – Part 3

Severity Level	**High**				
Technical Impact	High	**Likelihood**		**Medium**	
		Popularity	Medium	**Simplicity**	Medium
Observation	The application stores login credentials in a temporary file inside the application directory using File.createTempFile(). The file is written in plaintext and explicitly marked as readable and writable, making it easily accessible through ADB, emulator access, or on rooted devices.				
Impact	An attacker can retrieve stored credentials, modify or reuse them, access backups containing the file, or perform account takeover. The readable/writable permissions increase the exposure risk.				
Platform	**Android**				
	Windows				

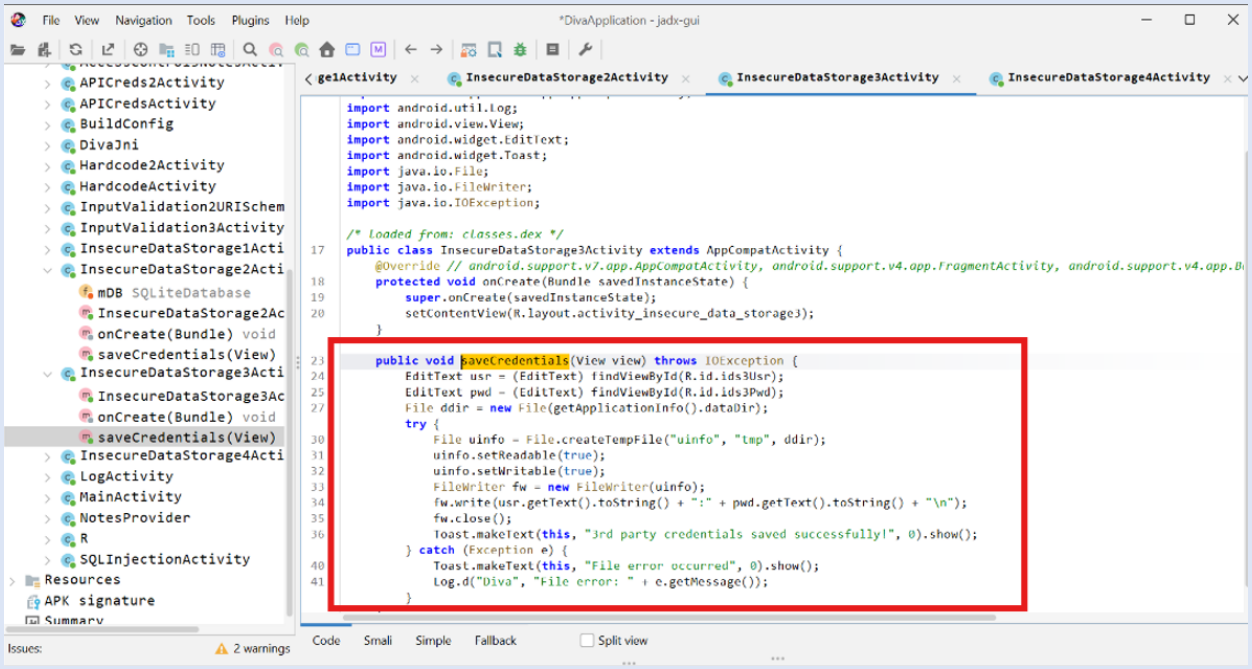
Proof of Concept

CODE ANALYSIS

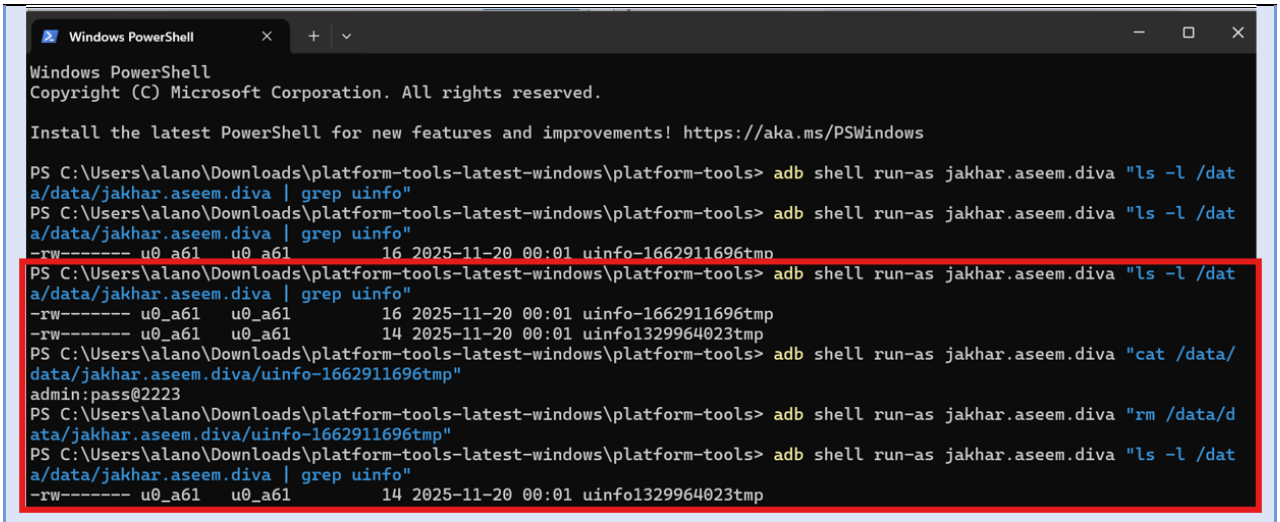
HIGH: 1 | WARNING: 3 | INFO: 1 | SECURE: 0 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/LogActivity.java jakhar/aseem/diva/SQLInjectionActivity.java

NO	ISSUE	SEVERITY	STANDARDS	FILES
3	Debug configuration enabled. Production builds must not be debuggable.	high	CWE: CWE-919: Weaknesses in Mobile Applications OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-RESILIENCE-2	jakhar/aseem/diva/BuildConfig.java
4	App creates temp file. Sensitive information should never be written into a temp file.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage3Activity.java
5	App can read/write to External Storage. Any App can read data written to External Storage.	warning	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	jakhar/aseem/diva/InsecureDataStorage4Activity.java



- Creates a temporary file inside the app's data directory.
- Explicitly marks the file readable and writable (no owner-only flag).
- Writes the plaintext username:password into the file.
- Shows a toast on success or error.



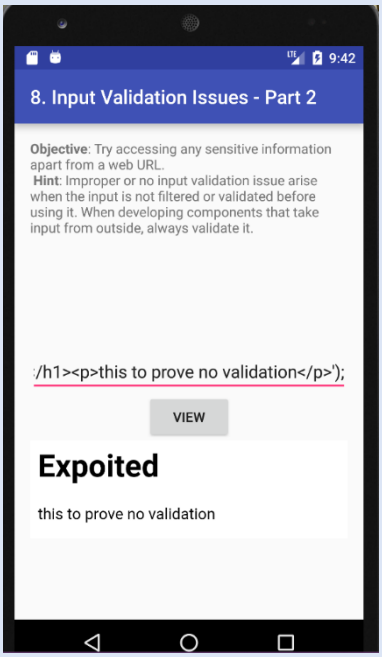
The output shows that plaintext credentials are stored in uinfo*.tmp. The file can be accessed, read, and deleted using adb shell run-as, confirming that unauthorized local access is possible due to insecure storage and improper file permissions.

The issue occurs because sensitive data is written to temporary files in plaintext and exposed through permissive file settings. Using secure encrypted storage mechanisms and avoiding globally readable permissions would prevent credential leakage and unauthorized file manipulation.

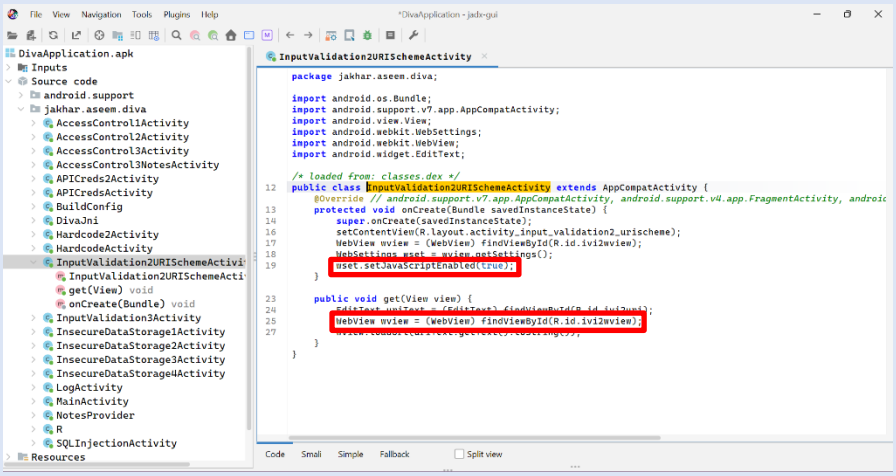
Recommendation	• Avoid writing plaintext credentials to disk. • Do not mark files as readable or writable globally. • Use encrypted storage mechanisms (EncryptedFile / EncryptedSharedPreferences). • Use server-issued tokens instead of storing raw credentials
Reference	**OWASP Mobile Top 10** **OWASP Mobile Security Testing Guide (MASTG) – MSTG-STORAGE-2** https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md#logs

4.2.6 Input Validation Issue – Part 2

Severity Level	**High**				
Technical Impact	Hight	**Likelihood**		**Hight**	
		Popularity	High	**Simplicity**	Hight
Observation	**In InputValidation2URISchemeActivity.java, user input is directly passed to WebView.loadUrl() without any URI scheme validation.**				
Impact	**URIs can be rendered in the app, proving lack of input validation. Could be used for content spoofing or local disclosure.**				
Platform	**Android**				
	Windows				
Proof of Concept					



User input was rendered directly without sanitization, confirming a lack of input validation controls.



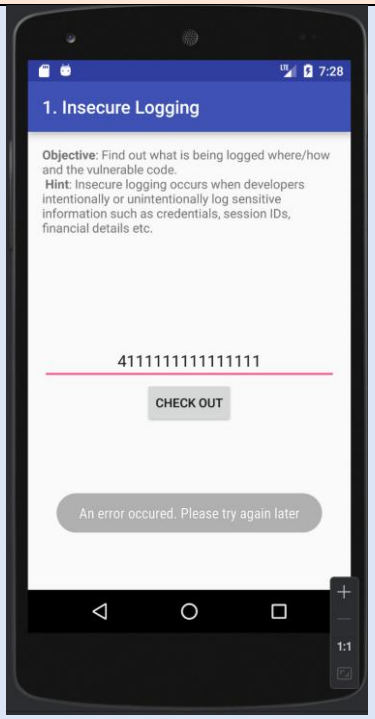
The application fails to validate user-supplied URLs, allowing redirection to untrusted websites. This open redirect vulnerability can be exploited by attackers to trick users into visiting malicious sites, posing risks such as phishing or malware attacks.

Recommendation	**• Use of Prepared Statements: To prevent SQL injection by separating SQL code from user input, ensuring queries are executed safely.** **• Input Validation: helps detect and block malicious patterns before they reach the database, reducing the risk of injection attacks.** **• Least Privilege Principle: limits each user's database permissions to the minimum necessary, reducing potential damage from exploited vulnerabilities.**
Reference	https://owasp.org/www-project-top-ten/

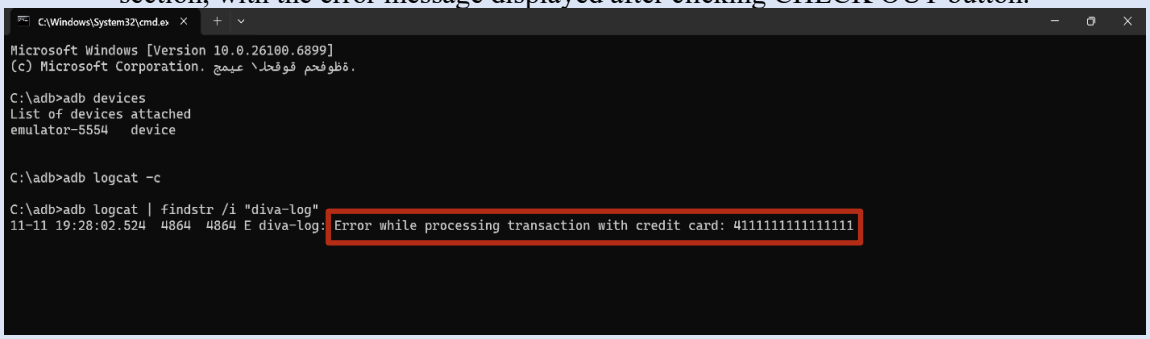
4.2.7 Insecure Logging

Severity Level	**Critical**				
Technical Impact	HIGH	**Likelihood**		HIGH	
		Popularity	HIGH	**Simplicity**	HIGH
Observation	The application logs sensitive credit card information to system logs using Log.e() method. During transaction errors, credit card numbers are written to				

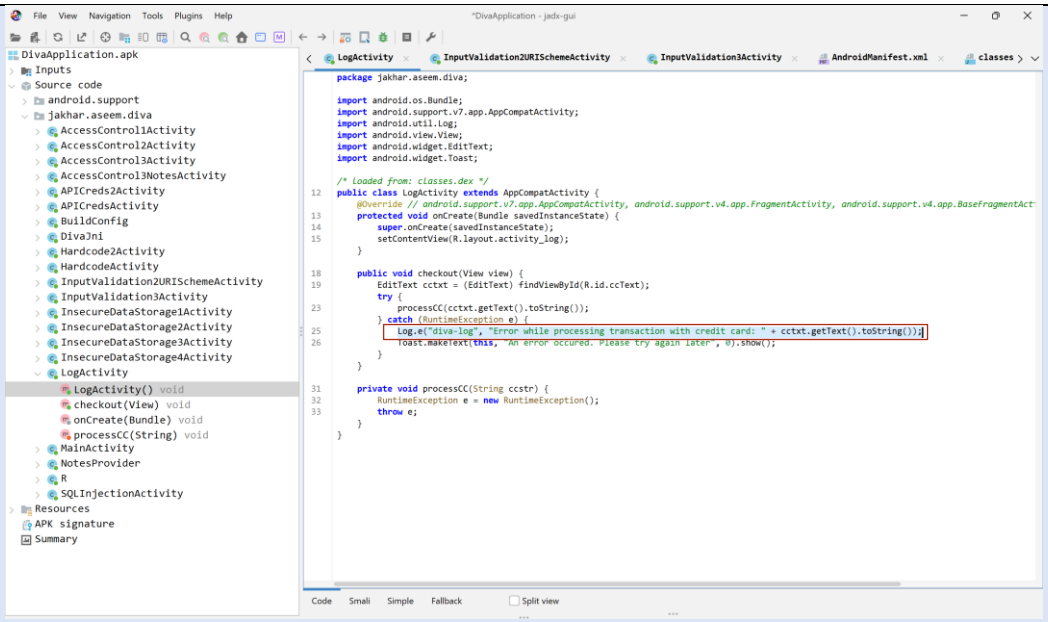
	logcat and become immediately accessible through ADB, exposing payment data to anyone with USB debugging access.
Impact	**The insecure logging practice in *LogActivity.java* exposes sensitive credit card information to the system logs. Through ADB (Android Debug Bridge), this data becomes accessible without requiring root privileges. This allows real-time capture of payment card data by anyone with USB debugging access, resulting in a serious confidentiality breach and potential financial fraud.**
Platform	Android
Proof of Concept	



The DIVA application interface showing a test credit card number entered in the insecure logging section, with the error message displayed after clicking CHECK OUT button.



ADB logcat output captured in terminal, clearly showing the exposed credit card number in the system logs, demonstrating the insecure logging vulnerability that leads to immediate data exposure.



JADX-GUI source code analysis highlighting the vulnerable logging statement in LogActivity() where sensitive credit card data is written to logs without protection.

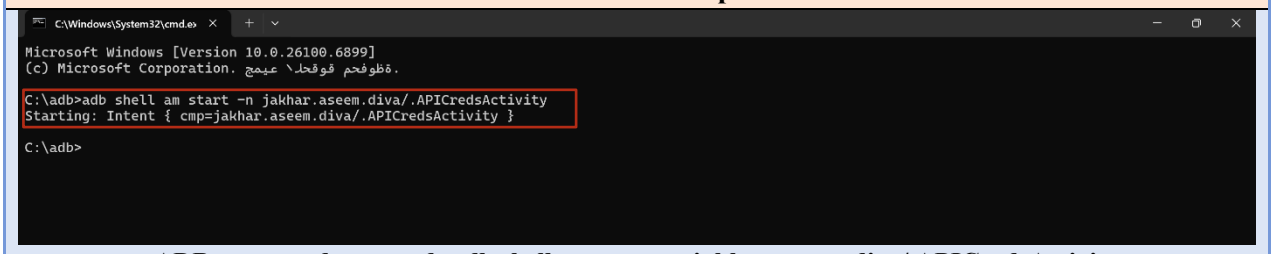
NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	jakhar/aseem/diva/AccessControl1Activity.java jakhar/aseem/diva/AccessControl2Activity.java jakhar/aseem/diva/InsecureDataStorage2Activity.java jakhar/aseem/diva/InsecureDataStorage3Activity.java jakhar/aseem/diva/InsecureDataStorage4Activity.java jakhar/aseem/diva/LogActivity.java jakhar/aseem/diva/SQLInjectionActivity.java

MobSF tool detecting insecure logging vulnerability in LogActivity.java

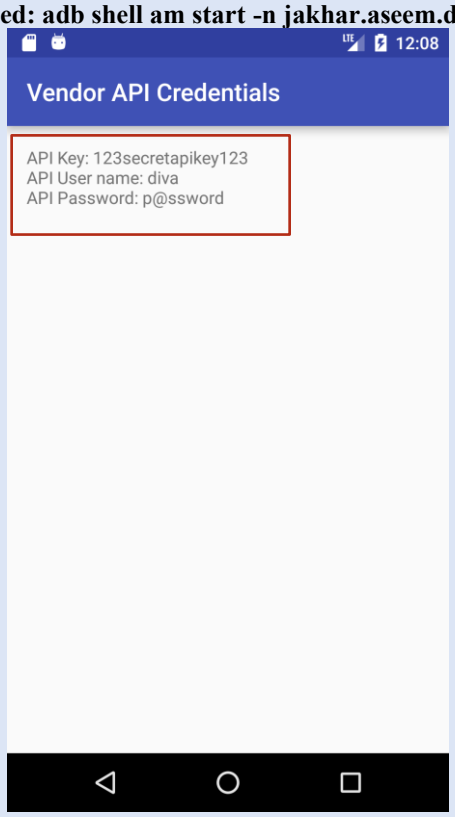
Recommendation	To address this specific logging vulnerability, modify the error handling in LogActivity.java by replacing the current Log.e() call that includes credit card numbers with a generic error message. Instead of logging the actual card data, use a secure alternative such as logging only a transaction ID or error code, and ensure all sensitive data is removed from log statements throughout the application.
OWASP Reference	**https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html**
Reference	https://developer.android.com/reference/android/util/Log **https://developer.android.com/privacy-and-security/risks/log-info-disclosure**

4.2.8 Access Control Issues – Part 1

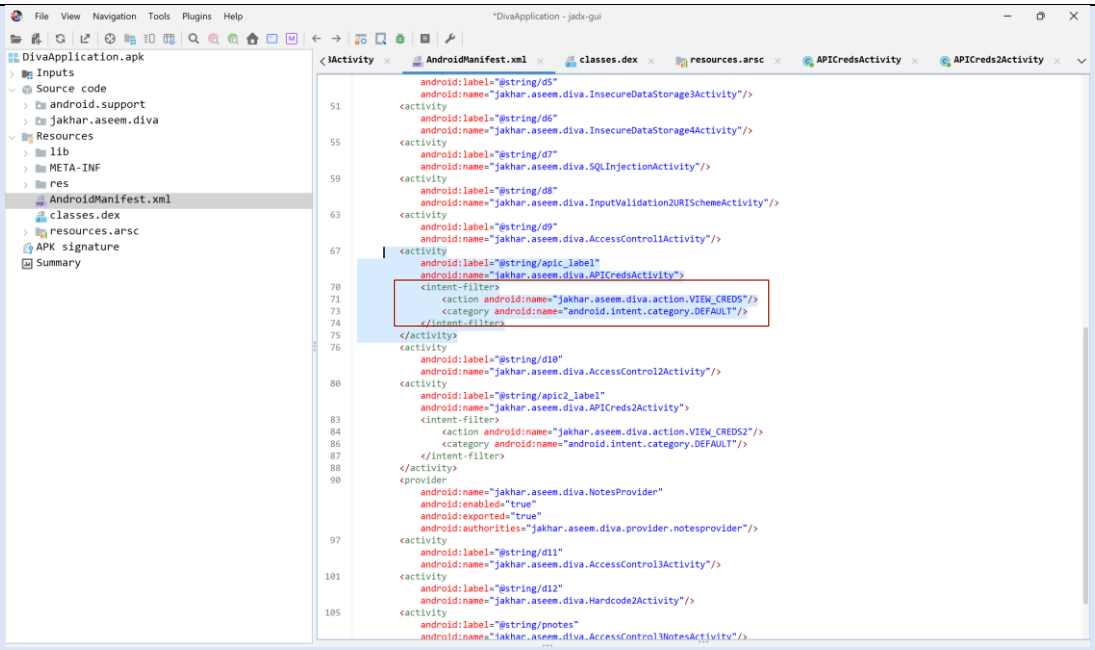
<table>
<tr><td>Severity Level</td><td colspan="4">Critical</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">HIGH</td><td colspan="2">Likelihood</td><td colspan="2">HIGH</td></tr>
<tr><td>Popularity</td><td>HIGH</td><td>Simplicity</td><td>HIGH</td></tr>
<tr><td>Observation</td><td colspan="4">The APICredsActivity is unintentionally exposed to external applications because it contains an intent-filter, which implicitly sets the component as exported. As a result, any external app can directly launch this activity and gain access to the hardcoded API credentials without authentication or user interaction.</td></tr>
<tr><td>Impact</td><td colspan="4">The exported APICredsActivity allows any external application to launch the activity and retrieve the hardcoded API credentials (Key: 123secretapikey123, Username: diva, Password: p@ssword). An attacker can use these credentials to impersonate the legitimate application when communicating with backend services, potentially gaining unauthorized access to protected API endpoints. This could allow data exposure, modification of user information, or other unauthorized operations depending on the privileges associated with the leaked credentials. The issue poses a high risk as it grants attackers the same level of backend access intended for the legitimate application.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
</table>



ADB command executed: adb shell am start -n jakhar.aseem.diva/.APICredsActivity



Activity successfully launched from external source without user interaction and sensitive API credentials (Key: 123secretapikey123, Username: diva, Password: p@ssword) exposed



AndroidManifest.xml revealing APICredsActivity is exported and accessible externally due to intent-filter declaration.

HIGH: 1 | WARNING: 4 | INFO: 0 | SUPPRESSED: 1

NO	ISSUE	SEVERITY	DESCRIPTION
1	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.
2	Application Data can be Backed up [android:allowBackup=true]	warning	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
3	Activity (jakhar.aseem.diva.APICredsActivity) is not Protected. An intent-filter exists.	warning	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
4	Activity (jakhar.aseem.diva.APICreds2Activity) is not Protected. An intent-filter exists.	warning	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.

MobSF security scan flagging APICredsActivity as exported due to unprotected intent-filter, creating an access control vulnerability that allows external application access

Recommendation	To resolve this specific vulnerability, add android:exported="false" attribute to the APICredsActivity declaration in AndroidManifest.xml. This will prevent external applications from directly launching this activity while maintaining its functionality within the DIVA application. Additionally, consider removing the intent-filter if external access is not required.
OWASP Reference	**https://owasp.org/www-project-mobile-top-10/2023-risks/m8-security-misconfiguration.html**
Reference	https://developer.android.com/guide/topics/manifest/intent-filter-element **https://developer.android.com/privacy-and-security/risks/android-exported**

Appendix A: About the Team

Team Code:		
Student Name	Serial Number	Role
Aseel Almubaddel	444202930	○ Hardcoding issues- Part 1 ○ Input Validation Issues – Part 1 ○ Introduction ○ Scope
Tala Alsheail	444200518	○ Input Validation Issues – Part 2 ○ Insecure Data Storage – Part 1 ○ Limitations
Norah Aldalal	443200118	○ Insecure Logging ○ Access Control Issues - Part 1 ○ Vulnerability Analysis ○ Methodology
Alanoud Aloradi	444201111	○ Insecure data storage- Part 2 ○ Insecure data storage- Part 3 ○ Conclusion ○ Methodology