King Saud University

College of Computer and Information Sciences

Information Technology Department

Application Security (IT 371)

Semester two 2024

External Application Penetration Testing Technical Report

For AndroGoat

Prepared By

Haya Alhawaimel

Sarah Aldbasi

Supervised by

L.Noura Al-Madi

Document Info

Item	Description
Document Title	External Application penetration Testing Technical report for Client Name.
Requestor	L.Noura Al-Madi for IT371 course project
Author/s	Haya Alhawaimel – Sarah Aldbasi
Date Created	4/3/2024

Table of Contents:

1 EXECUTIVE SUMMARY 4
- 1.1 Introduction 4
- 1.2 Scope 4
- 1.3 Risk Rating 4
- 1.4 Threat Security Level 5
- 1.5 Summary Table 5
- 1.6 Summary Graph 6
- 1.7 Key Findings 6

2 *CONCLUSION* 7

3 *METHODOLOGY* 7

4 DETAILED FINDINGS 13
- 4.1 Limitations 13
- 4.2 Technical Description of Findings 13
  - 4.2.1 Allow debugging for app (Reverse Engineering). 14
  - 4.2.2 Insecure version. 15
  - 4.2.3 Hardcoding sensitive data in source code. 16
  - 4.2.4 Insecure Clipboard Use. 18
  - 4.2.5 Insecure Logging. 20

APPENDIX A: ABOUT THE TEAM 22

1 Executive Summary

1.1 *Introduction*

The technical report details the methodology and results of an external breach conducted to test the security of the customer's application. This penetration test was part of a security course project, at the request of the course trainer, Noura Al-Madi. The primary goal was to uncover vulnerabilities that could be exploited by an attacker without prior knowledge of the APK. The report utilizes various tools—jadx, mobsf, and adb—and we also use the BlueStacks emulator, as detailed in the methodology section, to perform the testing.

The executive summary defines the scope of penetration testing and ranks risks based on their technical impact—confidentiality, integrity, and availability—and their likelihood, considering factors such as popularity and simplicity. It also includes the threat security level, a summary table, and a graph showing the vulnerabilities discovered during testing.

The main body of the report provides a comprehensive examination of the top five vulnerabilities discovered, complete with comments and recommendations for mitigating these vulnerabilities. The Results section offers a detailed technical description of the identified vulnerabilities and discusses limitations encountered during testing.

The methodology section outlines the stages of planning and preparation, information gathering, risk modeling and assessment, and the reporting process.

Furthermore, there is a detailed findings section that explains the limitations encountered and provides technical details on the identified vulnerabilities.

Finally, the report concludes with a summary of the findings and key weaknesses identified.

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
AndroGoat	Android	1.0	Windows	Black box penetration testing

Table1.Project Scope.

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 2. 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.

Critical	High	Medium	Low
3	**2**	**-**	**-**

Table3. Mobile Application Penetration Testing.

1.6 *Summary Graph*

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

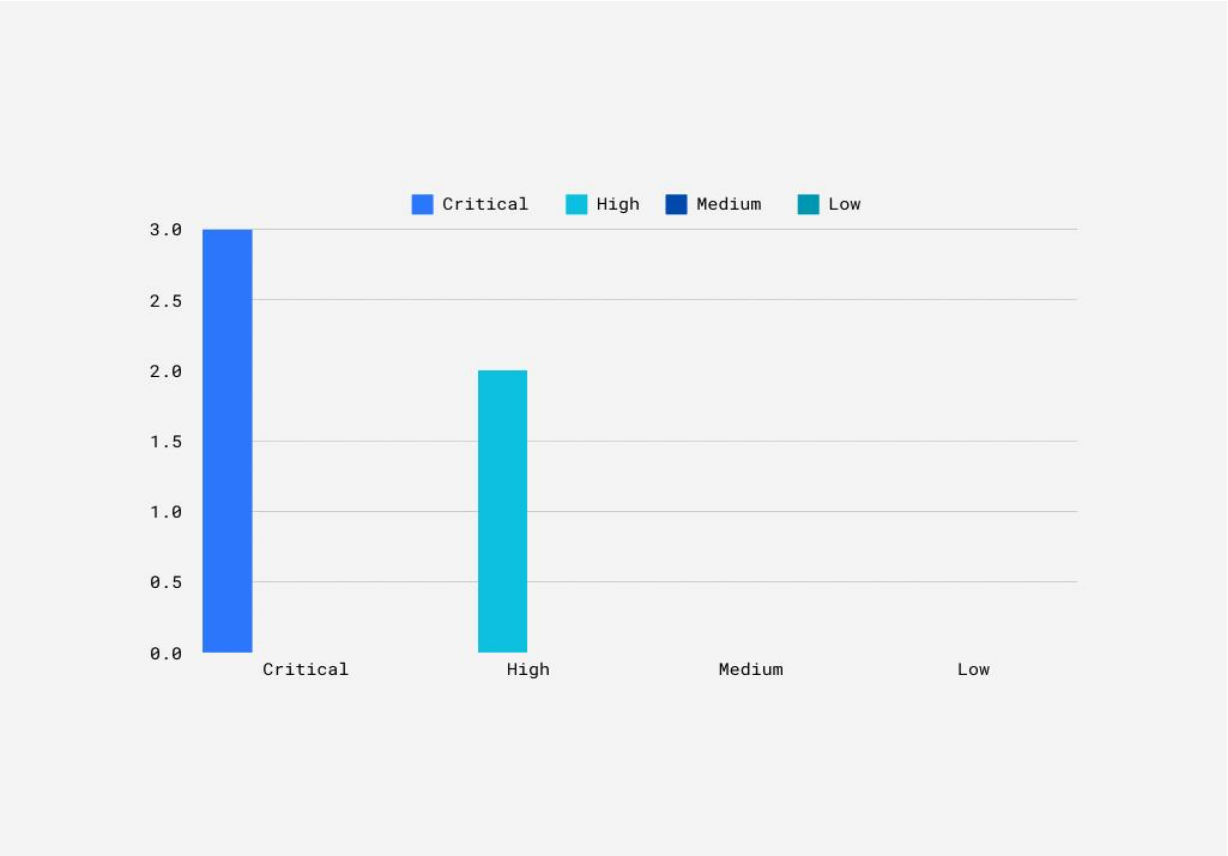


Figure 1. Application Penetration Testing.

1.7 *Key Findings*

No.	Vulnerabilities Discovered	Platform	Severity Level
1	Allow debugging for app (reverse engineering).	Android	**High**
2	Insecure version.	Android	**High**
3	Hardcoding sensitive data in source code.	Android	**Critical**
4	Insecure Clipboard use.	Android	**Critical**
5	Insecure Logging.	Android	**Critical**

Table 4. Key Findings.

2 *Conclusion*

During our application penetration testing, we discovered five vulnerabilities that attackers could exploit to gain unauthorized access to sensitive data or perform unauthorized actions. To improve app security, we suggest you follow our tips on how to do this.
Mitigating these weaknesses:

• Hardcoding sensitive data in source code:
Usernames and passwords are easy to find as plain text. in the source code, which represents a serious security vulnerability. To mitigate this vulnerability, store sensitive data outside of code, restrict access to it, and protect passwords.

• Insecure Logging: a security vulnerability that leads to username or password leakage.
To mitigate this vulnerability, sensitive data must be securely logged using appropriate encryption. Storage to prevent unauthorized access and tampering. Limit the amount of recorded data to reduce Possibility of exposure.

• Allow debugging of the application (reverse engineering):
Attackers can exploit debugging tools to gain access. Sensitive data or information.
To mitigate this vulnerability, android:debuggable must be set to false.

• Insecure Clipboard use:
To mitigate this vulnerability, we must prevent the application from automatically storing sensitive information such as passwords, credit card numbers, or other confidential data on the clipboard. and implement a mechanism to automatically clear the contents of the clipboard after a certain period or after copying or pasting sensitive information.

• Insecure version:
To mitigate this vulnerability, we should Stay up to date: Make sure the app and all its dependencies are running on the latest stable versions. This includes the operating system, web server, application framework, libraries, and plugins. Update and patch your software regularly to address known vulnerabilities.

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 X phases or any preparations:

To begin the process, we must first establish the necessary environment. This involves installing and configuring several tools, Bluestack 5, Jadx, MobSF, and ADB. and downloading the target APK, which is AndroGoat.

- **BlueStacks5:**

Bluestack 5 is an Android emulator that allows us to run the AndroGoat APK on our Windows computers, enabling us to explore its features and test for potential vulnerabilities.
We downloaded the app from their official website: [Fastest & Lightest Android App Player for PC - BlueStacks 5]

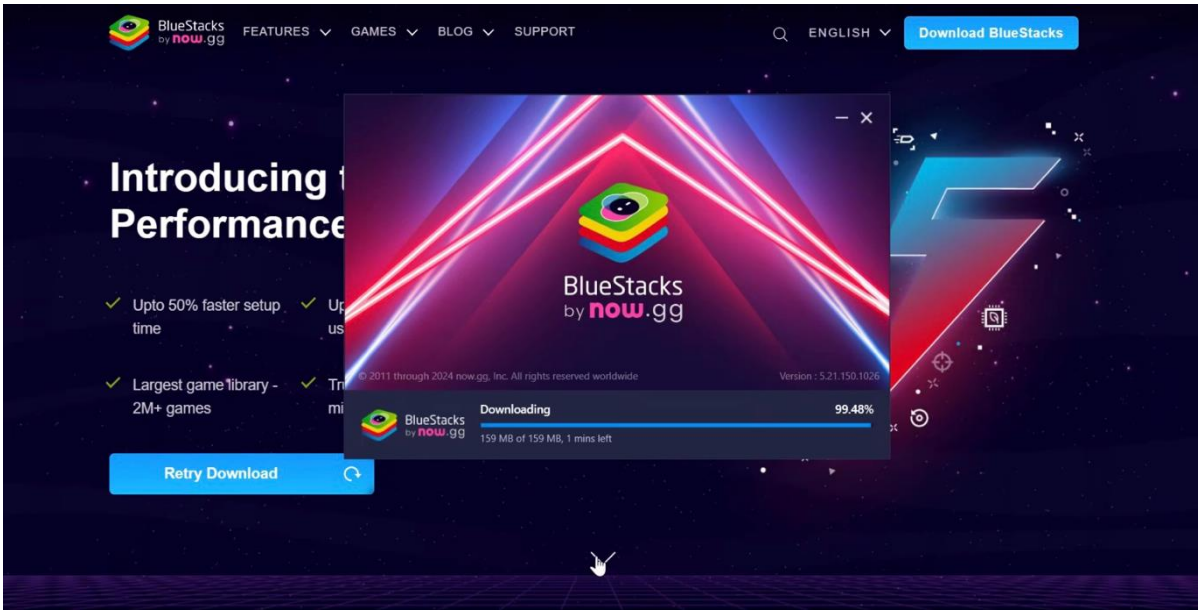


Figure 2. BlueStack downloading.

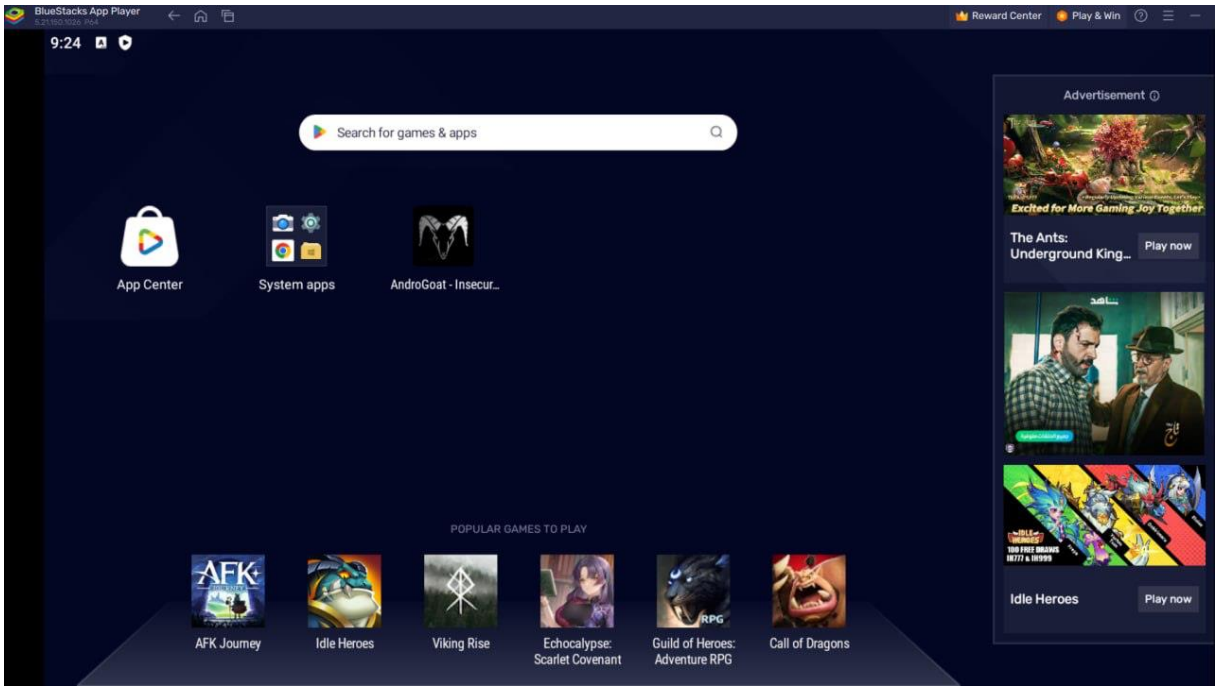


Figure 3. BlueStack interface.

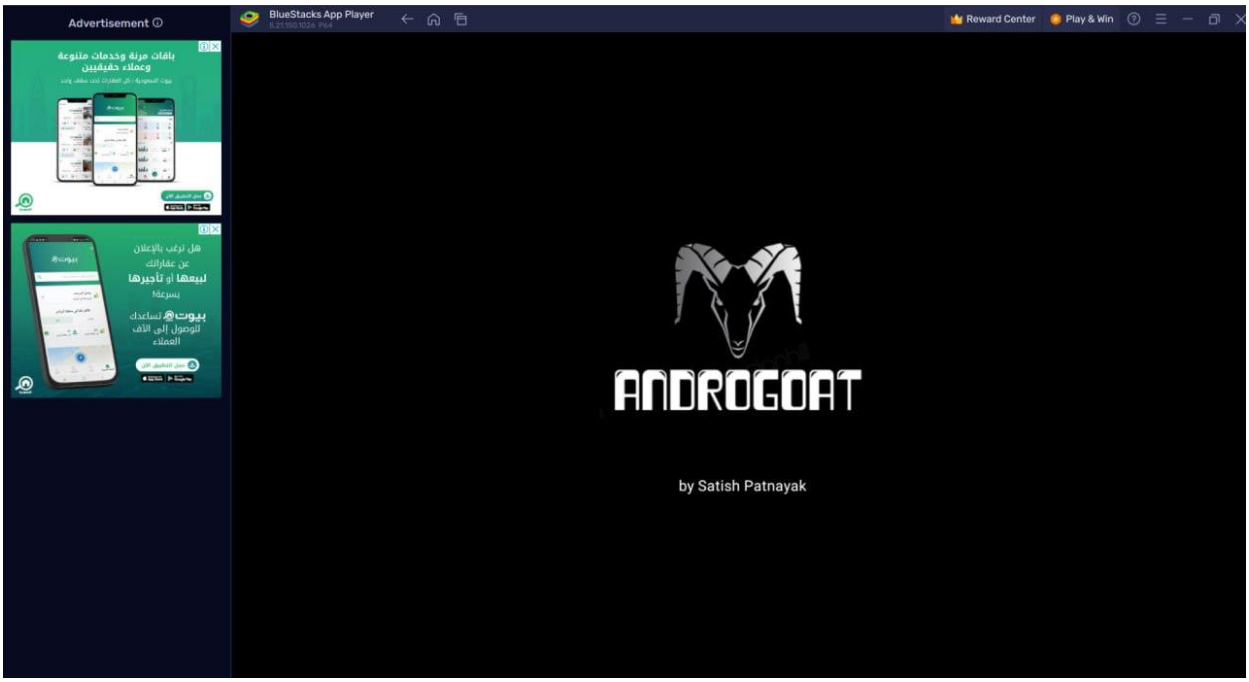


Figure 4: Open AndroGoat in BlueStack.

- **Jadx:**

Jadx is an open source decompiler used for reverse engineering Android applications. It allows developers and security researchers to analyze compiled APK files and extract the original Java source code, resources, and other assets. By decompiling an Android app with JADX, developers can gain insights into the inner workings of the application, understand its functionality, and potentially identify vulnerabilities or security issues.
We are downloading the app: https://sourceforge.net/projects/jadx.mirror/

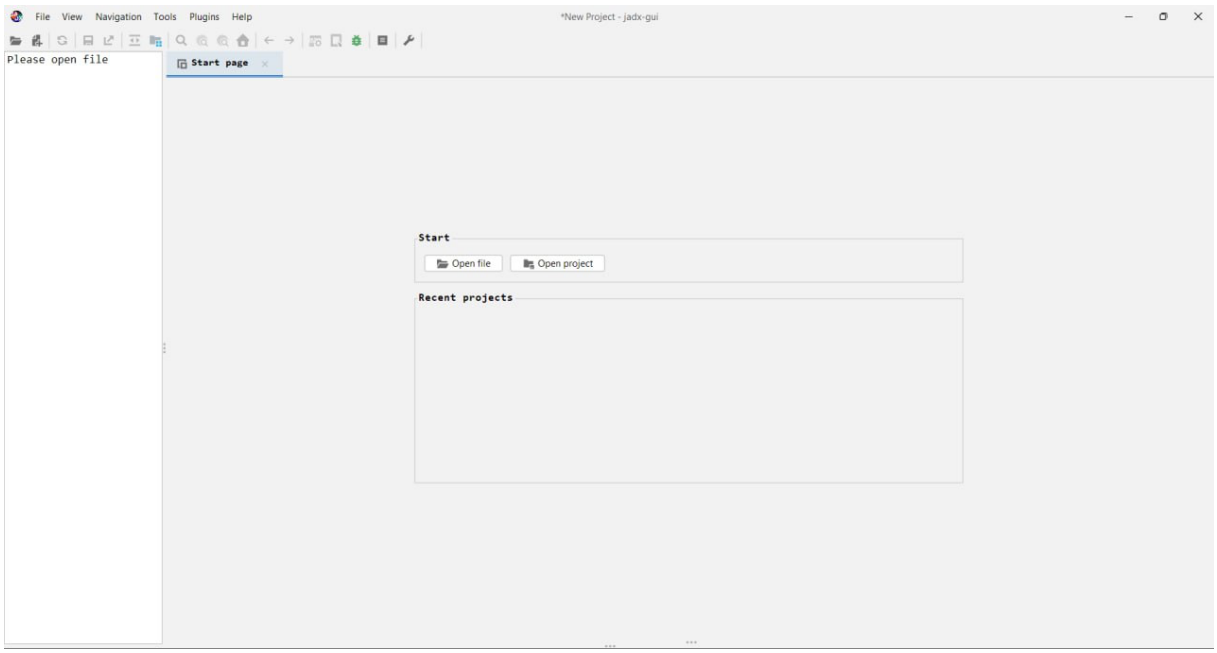


Figure 5. Home page for Jadx GUI.

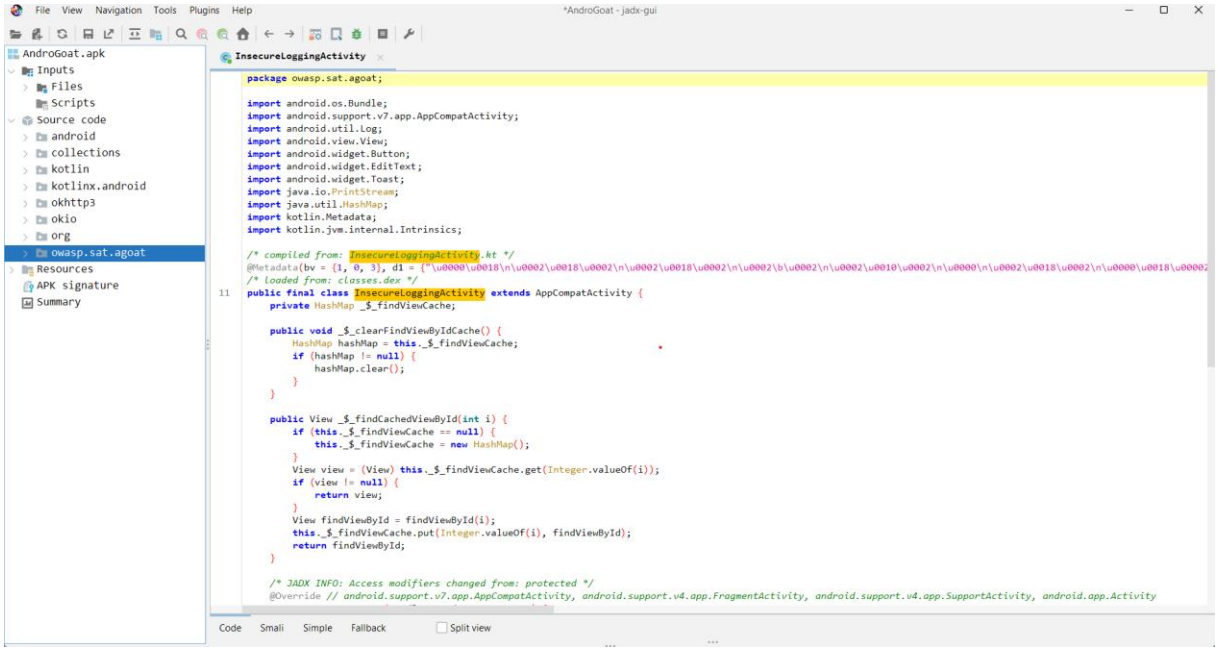


Figure 6. AndroGoat source code in Jadx.
We upload AndroGoat File from "open file" option.

- ## MobSF (Mobile Security Framework):

MobSF, the Mobile Security Framework, is an open-source testing framework for assessing the security of mobile applications. It offers both static and dynamic analysis capabilities, allowing users to analyze source code, detect malware, assess API security, and generate detailed reports. With its integration and extensibility features, MobSF is widely used by security researchers and developers to identify vulnerabilities, ensure the security of mobile apps, and enhance overall mobile application security.

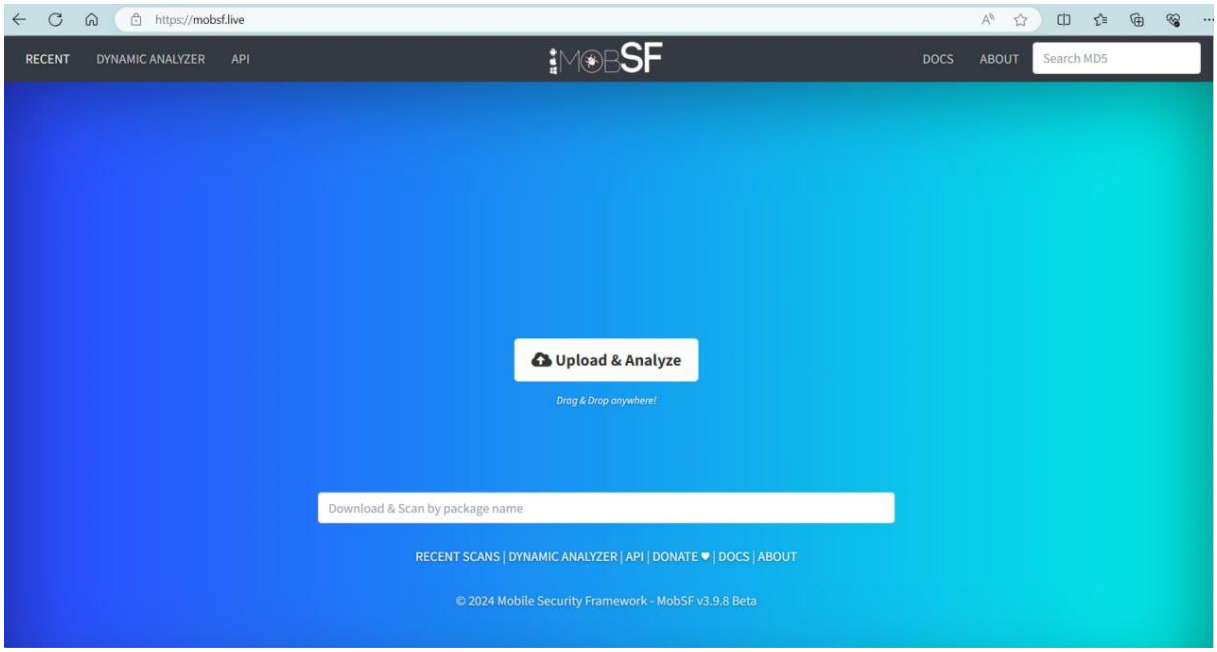


Figure 7. Browsing MobSF and uploading the file.

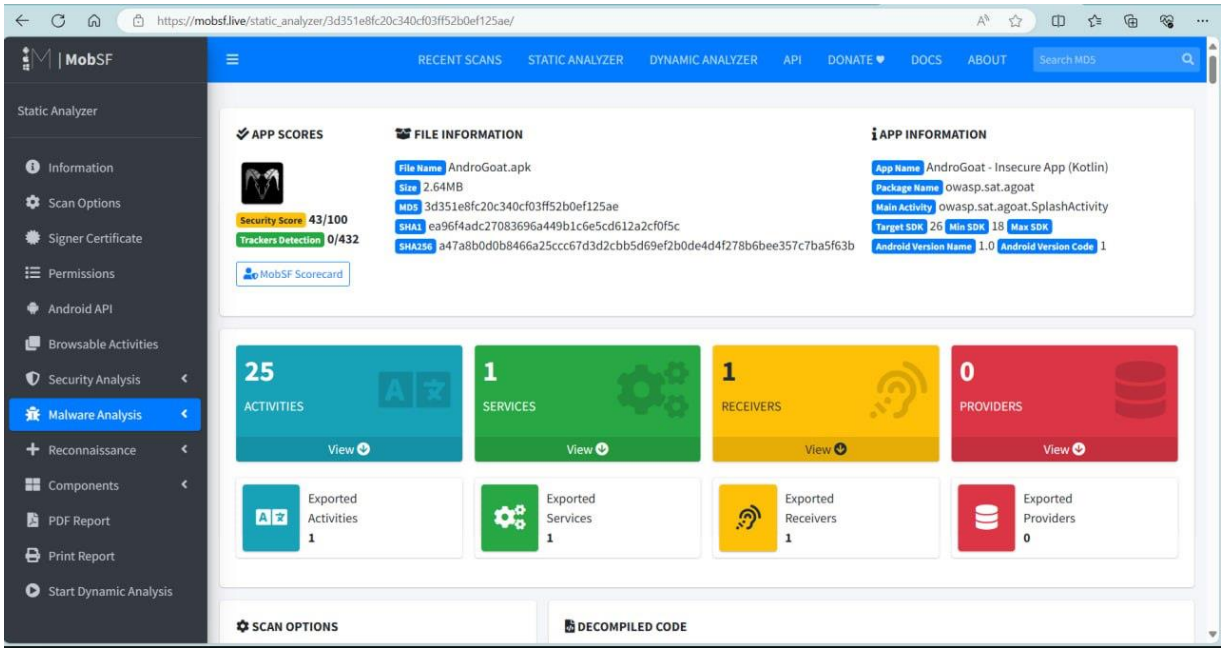


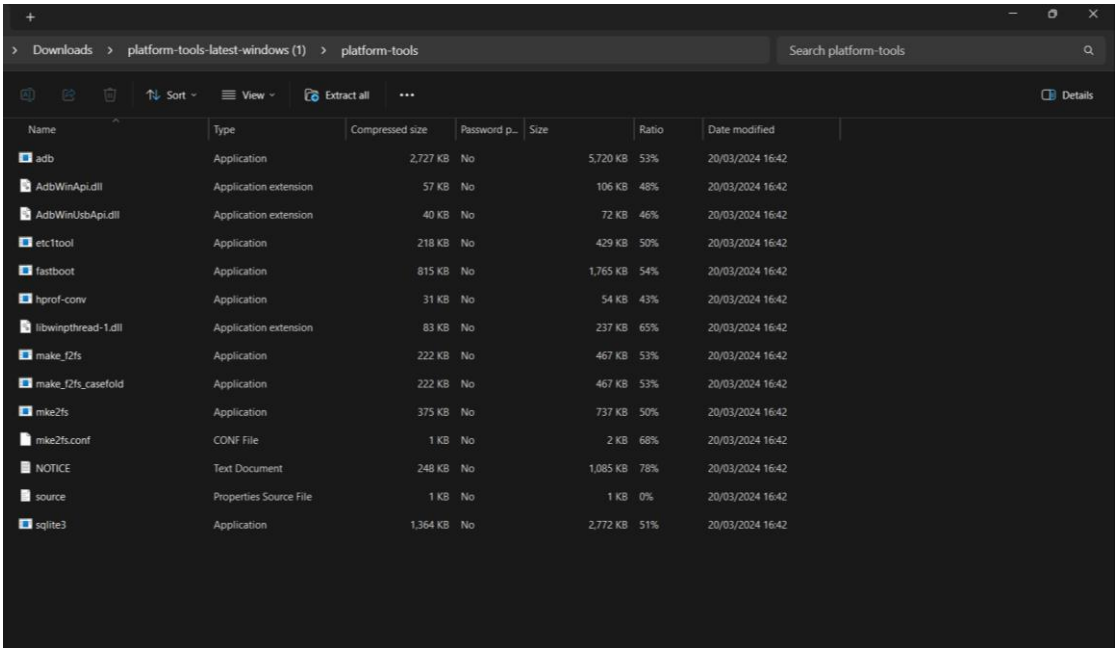
Figure 8: Vulnerability classified.

- ## ADB (Android Debug Bridge):

  One command-line utility that comes with the Android SDK (Software Development Kit) is called ADB. We utilized it because it enables communication between the Bluestack emulator and our computers.

  running the Android development tools to perform various tasks, such as installing and debugging.

  applications, accessing logs, and running shell commands. To connect the ADB to the emulator, we first needed to enable the ADB option in the emulator from Setting-Advanced. Then, we downloaded the Android SDK Platform Tools on our computers from this link: https://dl.google.com/android/repository/platform-tools-latest windows.zip, then we opened the folder in the terminal, and entered the ". /adb devices" command to launch the ADB and the emulator number appeared on the screen. Lastly, we entered ". /abd shell" command so we could run any ADB command on the emulator.



Name	Type	Compressed size	Password p...	Size	Ratio	Date modified
adb	Application	2,727 KB	No	5,720 KB	53%	20/03/2024 16:42
AdbWinApi.dll	Application extension	57 KB	No	106 KB	48%	20/03/2024 16:42
AdbWinUsbApi.dll	Application extension	40 KB	No	72 KB	46%	20/03/2024 16:42
etc1tool	Application	218 KB	No	429 KB	50%	20/03/2024 16:42
fastboot	Application	815 KB	No	1,765 KB	54%	20/03/2024 16:42
hprof-conv	Application	31 KB	No	54 KB	43%	20/03/2024 16:42
libwinpthread-1.dll	Application extension	83 KB	No	237 KB	65%	20/03/2024 16:42
make_f2fs	Application	222 KB	No	467 KB	53%	20/03/2024 16:42
make_f2fs_casefold	Application	222 KB	No	467 KB	53%	20/03/2024 16:42
mke2fs	Application	375 KB	No	737 KB	50%	20/03/2024 16:42
mke2fs.conf	CONF File	1 KB	No	2 KB	68%	20/03/2024 16:42
NOTICE	Text Document	248 KB	No	1,085 KB	78%	20/03/2024 16:42
source	Properties Source File	1 KB	No	1 KB	0%	20/03/2024 16:42
sqlite3	Application	1,364 KB	No	2,772 KB	51%	20/03/2024 16:42

Figure 9. Platform tool folder – Containing the ADB file.

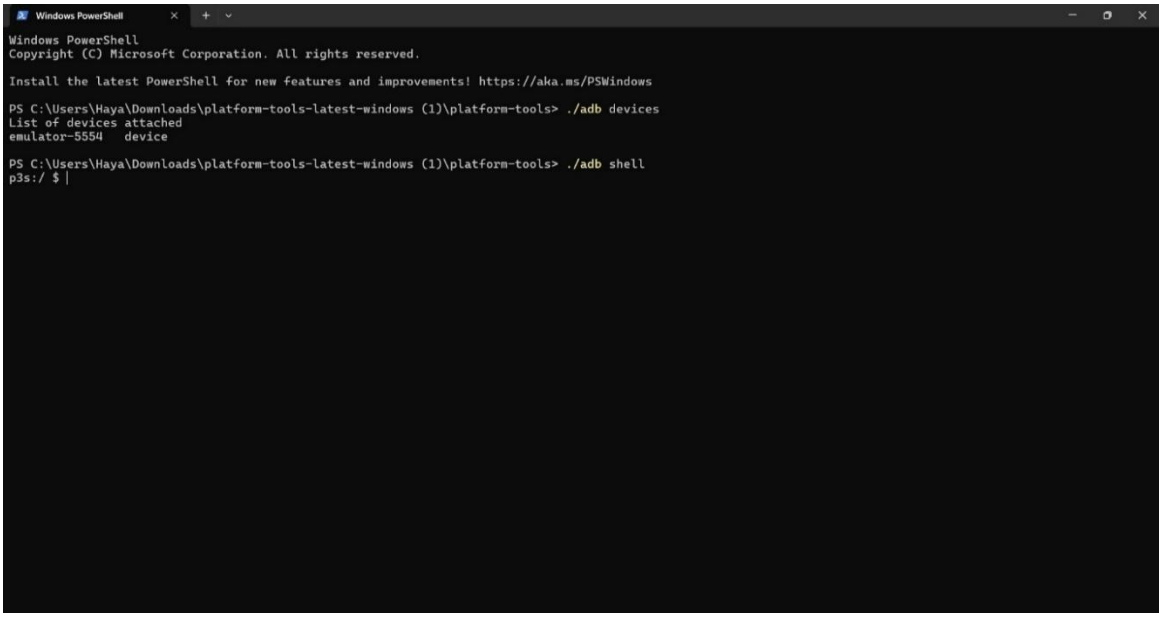


Figure 10. Windows PowerShell -Connecting the ADB with emulator and lunch the ADB shell.

4 Detailed Findings

4.1 *Limitations*

During penetration testing, we encountered several limitations that affected the process. Limited time was one of the major challenges, as the test had to be completed within a period. In addition, the weak processing and memory capabilities of laptops also affected the functionality of the gadgets. In addition, while testing and extending the testing procedures, we encountered some difficulties while finding vulnerabilities using Jadx.

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.

4.2.1 Allow debugging for app (Reverse Engineering).

<table>
<tr><td>Severity Level</td><td colspan="4">High</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">Medium</td><td>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">By using debugging tools, attackers are able to access sensitive data or information when android:debuggable="true" is set.
Debuggable code offers more information to aid developers in identifying and resolving flaws, making it easier to reverse engineer, but this information can also be exploited by attackers to find and take advantage of potential vulnerabilities.</td></tr>
<tr><td>Impact</td><td colspan="4">Attackers might have access to private information kept in the app's code or memory. In addition, the security of the app can be compromised if hackers have access to the code while it is operating. Plus, users of the app may be vulnerable to fraud or other online attacks like phishing if hackers gain access to their personal information or login details.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
</table>

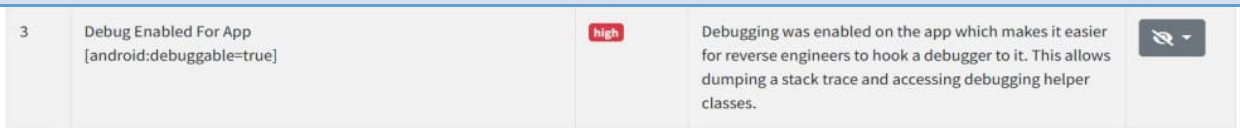


Figure 11. MobSF vulnerability analysis.

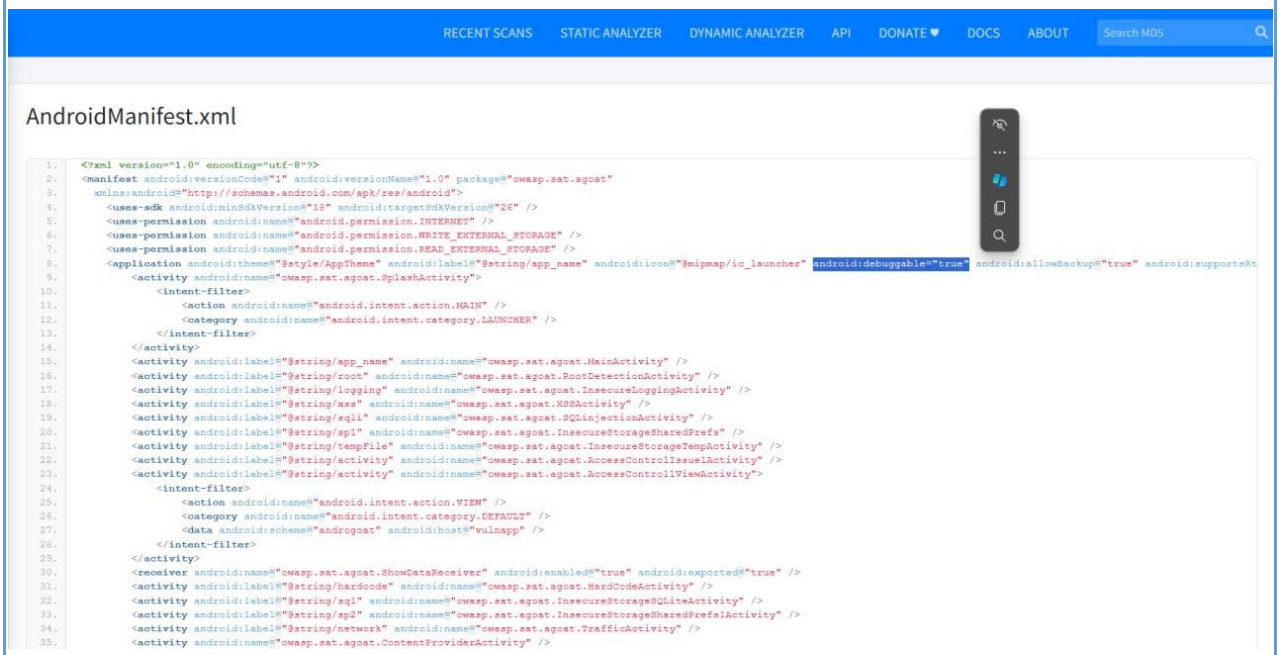


Figure 12. Manifest.xml

<table>
<tr><td>Recommendation</td><td colspan="4">1. Disable debugging on production versions of the application to prevent security vulnerabilities and data exposure.

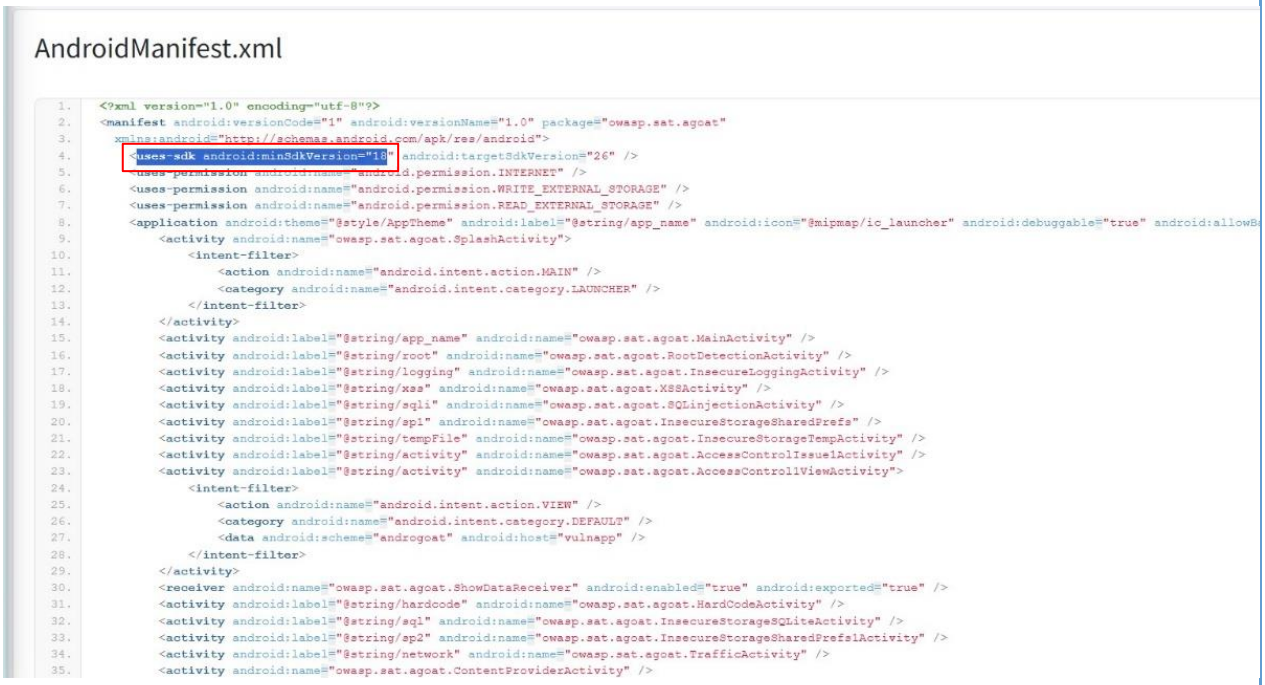
2. Limit access to debugging tools to authorized individuals only to minimize the risk of unauthorized access and data breaches.</td></tr>
<tr><td>OWASPReference</td><td colspan="4">https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering</td></tr>
<tr><td>Reference</td><td colspan="4">https://medium.com/@simonsruggi/decoding-the-magic-an-introductory-guide-to-reverse-engineering-229cd721269d</td></tr>
</table>

4.2.2 Insecure version.

Severity Level	**High**				
Technical Impact	**High**	**Likelihood**		**High**	
		Popularity	**Medium**	**Simplicity**	**Medium**
Observation	Installing apps on outdated, unpatched versions of Android like 4.3-4.3.1 poses significant security risks. These versions have known vulnerabilities that can be exploited by attackers, compromising user data, and enabling malware installation. Many users aren't aware of these risks or don't prioritize updates, increasing the likelihood of exploitation. To address this, collaboration between developers, manufacturers, and users is crucial. It requires timely updates, user education, and potentially regulatory measures to ensure devices remain secure.				
Impact	Installing apps on outdated and unpatched versions of Android, such as versions 4.3-4.3.1 (minSdk=18), carries significant consequences. These versions harbor known vulnerabilities, leaving users susceptible to various cyber threats. Data breaches become a looming risk, with personal information vulnerable to exploitation, potentially leading to identity theft or financial loss.				
Platform	**Android**				
Proof of Concept					

NO	ISSUE	SEVERITY	DESCRIPTION	OPTIONS
1	App can be installed on a vulnerable upatched Android version Android 4.3-4.3.1, [minSdk=18]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.	

Figure 13: MobSF vulnerability analysis.



AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="owasp.sat.agoat"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:minSdkVersion="18" android:targetSdkVersion="26" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:debuggable="true" android:allowB
        <activity android:name="owasp.sat.agoat.SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="@string/app_name" android:name="owasp.sat.agoat.MainActivity" />
        <activity android:label="@string/root" android:name="owasp.sat.agoat.RootDetectionActivity" />
        <activity android:label="@string/logging" android:name="owasp.sat.agoat.InsecureLoggingActivity" />
        <activity android:label="@string/xss" android:name="owasp.sat.agoat.XSSActivity" />
        <activity android:label="@string/sqli" android:name="owasp.sat.agoat.SQLinjectionActivity" />
        <activity android:label="@string/sp1" android:name="owasp.sat.agoat.InsecureStorageSharedPrefs" />
        <activity android:label="@string/tempFile" android:name="owasp.sat.agoat.InsecureStorageTempActivity" />
        <activity android:label="@string/activity" android:name="owasp.sat.agoat.AccessControlIssue1Activity" />
        <activity android:label="@string/activity" android:name="owasp.sat.agoat.AccessControlViewActivity">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="androgoat" android:host="vulnapp" />
            </intent-filter>
        </activity>
        <receiver android:name="owasp.sat.agoat.ShowDataReceiver" android:enabled="true" android:exported="true" />
        <activity android:label="@string/hardcode" android:name="owasp.sat.agoat.HardCodeActivity" />
        <activity android:label="@string/sql" android:name="owasp.sat.agoat.InsecureStorageSQLiteActivity" />
        <activity android:label="@string/sp2" android:name="owasp.sat.agoat.InsecureStorageSharedPrefs1Activity" />
        <activity android:label="@string/network" android:name="owasp.sat.agoat.TrafficActivity" />
        <activity android:name="owasp.sat.agoat.ContentProviderActivity" />
```

Figure14: Manifest.xml

Recommendation	The best practice to reduce security threats and guarantee device safety, updating to a supported version of Android is the best way to safeguard your data and device. To fix vulnerabilities right away, you should also enable automatic updates and updates installed apps on a regular basis.
OWASP Reference	**https://www.owasp.org/index.php/Testing_for_cookies_attributes_(OTG-SESS-002)**
Reference	https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/insecure-usage-of-version-1-guid/

4.2.3 Hardcoding sensitive data in source code.

Severity Level	**Critical**				
Technical Impact	**High**	**Likelihood**		**High**	
		Popularity	**High**	**Simplicity**	**High**
Observation	A promotional code is encoded in the app, creating a security vulnerability that enables users to obtain services or products for free by avoiding payment. The developer coded the products in class 'HardCodeActivity.java' and set 'promoCode.element' equal 'NEW2019'. If the attacker gains access to this code, they can easily get the products for free.				
Impact	1.Compromise of System Integrity: The exploitation may lead to unauthorized access to critical system resources, compromising the integrity of the entire system. Attackers could manipulate or corrupt data. 2. Data Breach: The vulnerability could allow attackers to gain access to sensitive data stored within the application or system. 3. Escalation of Privileges: Attackers could exploit the vulnerability to elevate their privileges within the system, gaining unauthorized access				
Platform	**Android**				

Proof of Concept

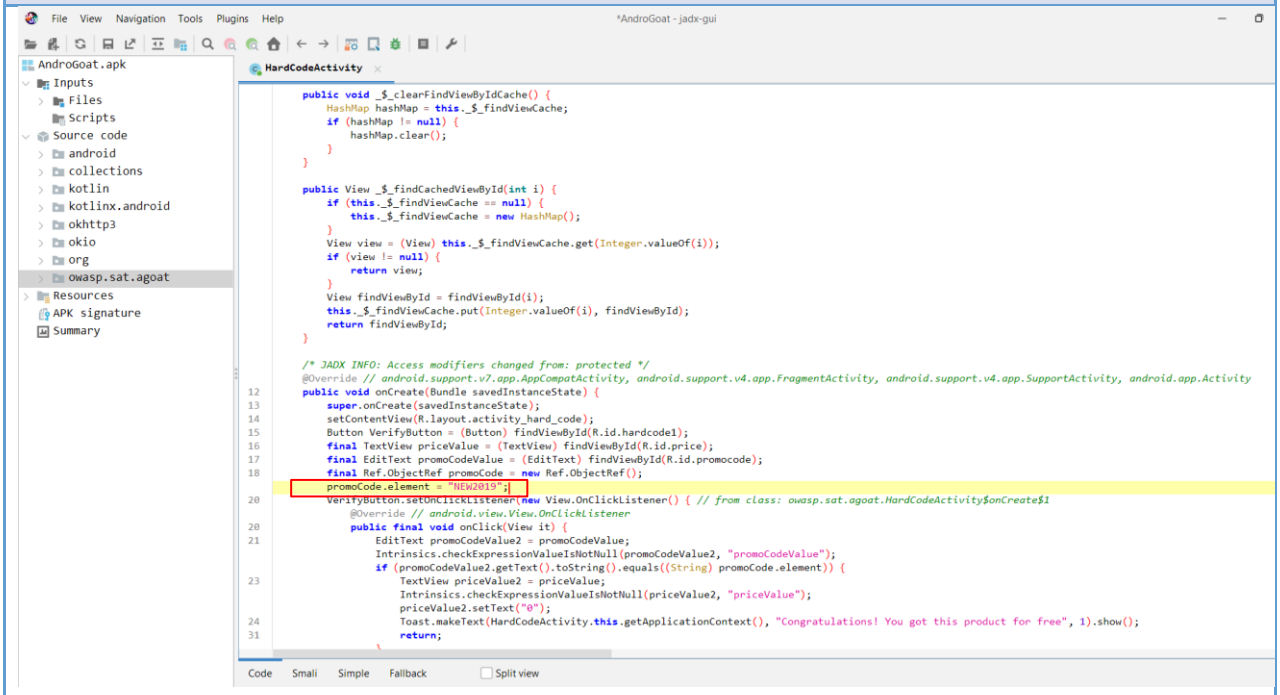


Figure 15. Class HardCodeActivity.java

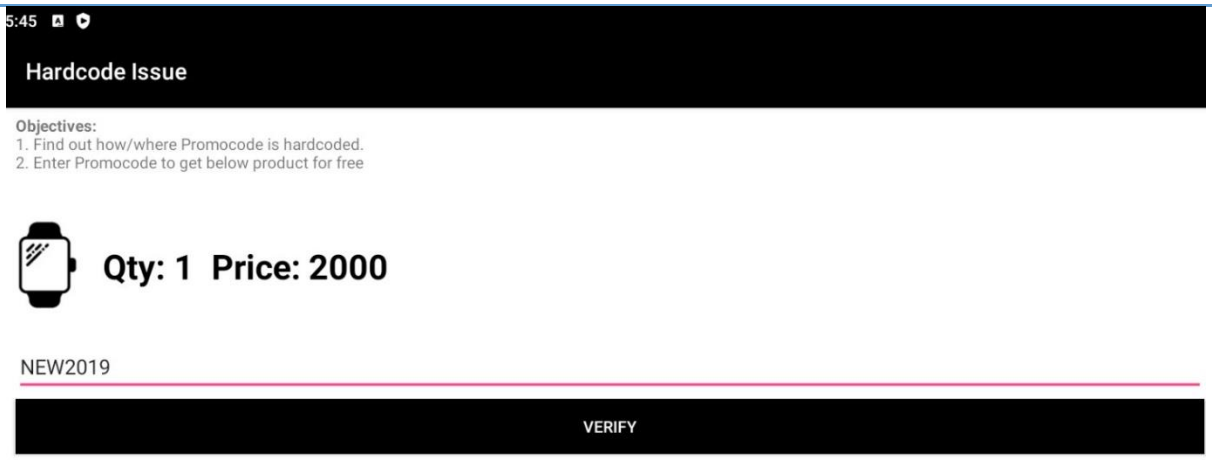


Figure 16. Hardcode issue before entering promCode.

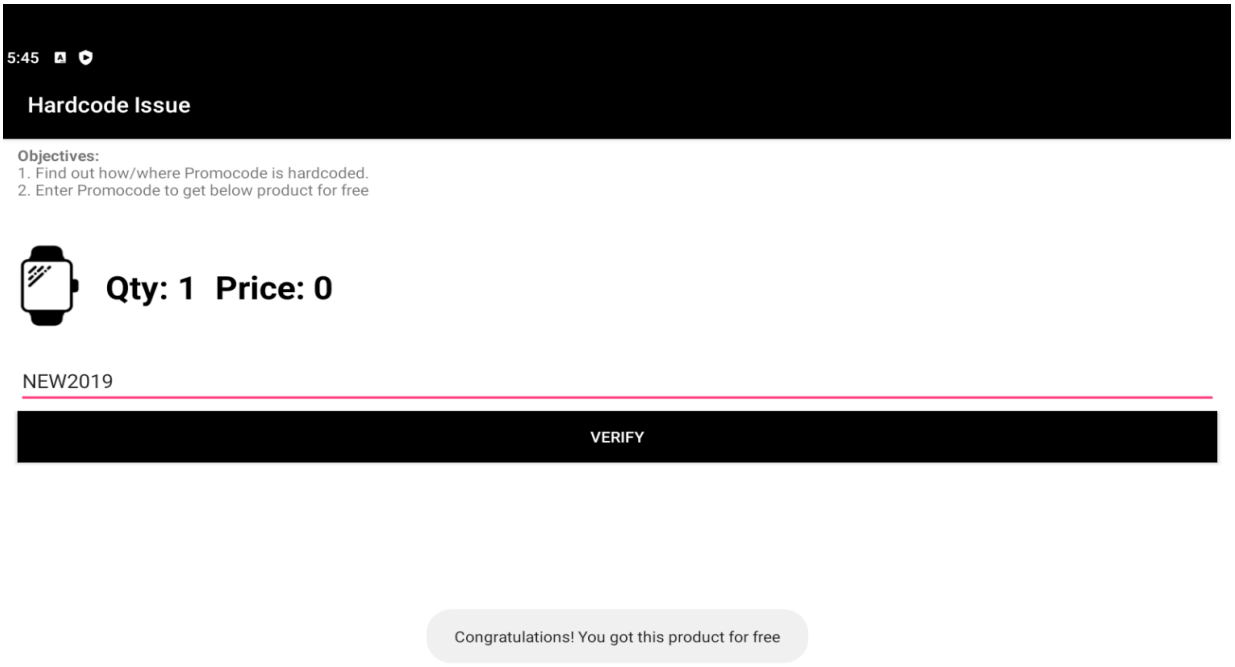


Figure 17: Hardcode issue after entering promCode.

Recommendation	1. Dynamic Promotional Codes: Instead of hardcoding promotional codes, use dynamic codes generated and validated server-side. This prevents attackers from easily exploiting hardcoded codes embedded within the application. 2. Monitor and Log Access: Implement logging and monitoring mechanisms to track access to sensitive functionalities and detect suspicious activities. Monitor for unauthorized access attempts or unusual patterns that may indicate a security breach.
OWASP Reference	https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password
Reference	https://www.synopsys.com/blogs/software-security/finding-hard-coded-secrets-before-you-suffer-a-breach.html

4.2.4 Insecure Clipboard Use.

<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>Medium</td><td>Simplicity</td><td>Medium</td></tr>
<tr><td>Observation</td><td colspan="4">The inadvertent display of OTPs presents a severe security risk, allowing potential unauthorized access, fraudulent transactions, and data breaches.
The developer has hard-coded the OTP in the source code in the class 'ClipboardActivity.java' by setting Toast.makeText(ClipboardActivity.this, "Created and copied OTP: " + otp, 1).show();`,if the attacker types the credit card number, they can access the OTP.</td></tr>
<tr><td>Impact</td><td colspan="4">1. Financial Loss: Unauthorized transactions resulting from the exposure of OTPs could lead to significant financial losses for both users and the organization responsible for the system.

2. Data Breach: The exposure of OTPs and potential unauthorized access to user accounts represent a breach of sensitive data, compromising user privacy and confidentiality.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
</table>

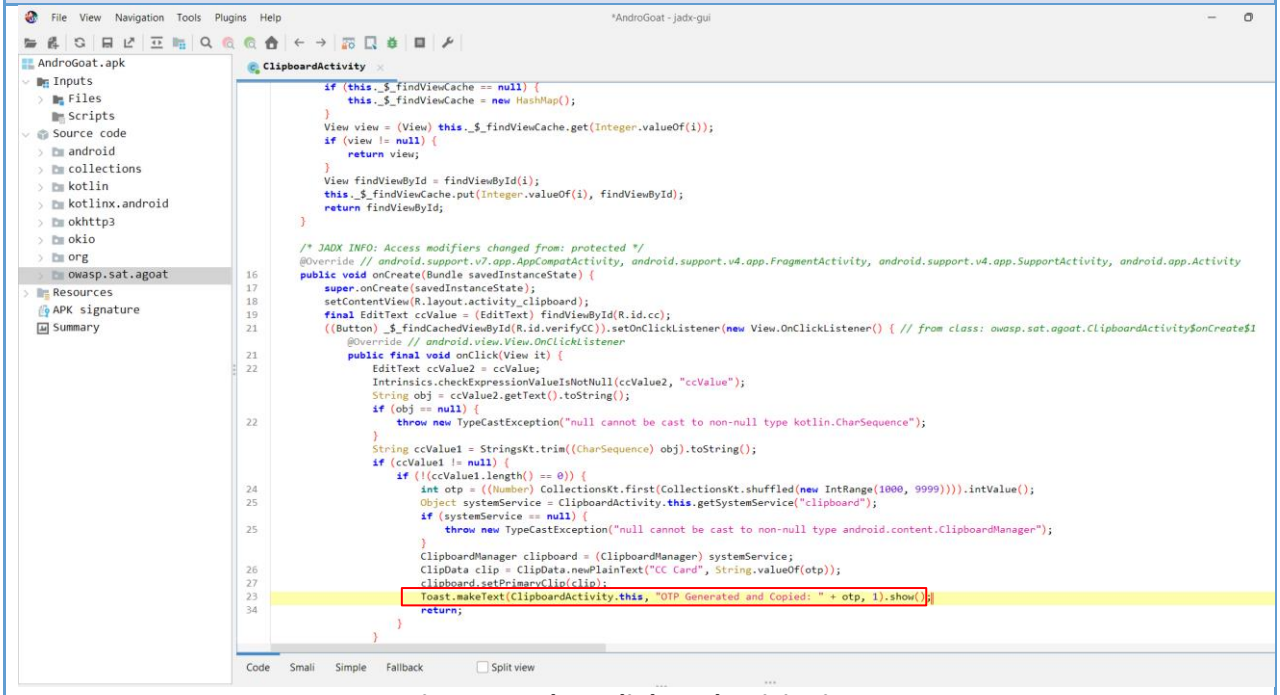


Figure 18. Class ClipboardActivity.java

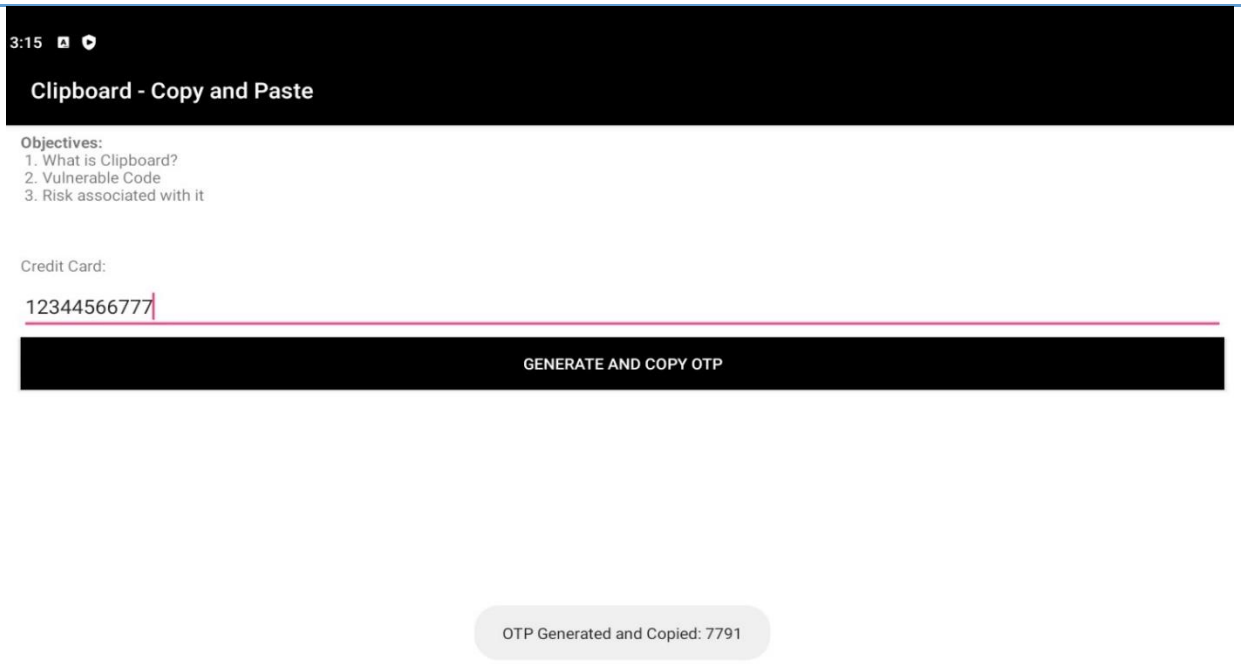


Figure 19. Clipboard-Copy and Paste.

Recommendation	1. Avoid putting sensitive data in the clipboard: Before copying and pasting any sensitive data such as passwords or credit card numbers, make sure to secure the clipboard first. 2. Use password manager applications: Use secure password storage applications where data is encrypted and secured well, providing secure interfaces for clipboard-related operations.
OWASP Reference	**https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure**
Reference	**https://www.packetlabs.net/posts/clipboard-data-security/**

4.2.5 Insecure Logging.

Severity Level	Critical				
Technical Impact	High	Likelihood		High	
		Popularity	Medium	Simplicity	Medium
Observation	When we have upload the apk file on jadx, we have seen this vulnerability in the source code which is InsecureLoggingActivity shows that the username and password that we apply on the app are being leaked into logcat.				
Impact	After the credentials have been logged it will be leaked so we can access it easily be connecting the adb with the emulator then any username and password will be exposed in the log to the attacker which is information discourse and violate confidentially.				
Platform	Android				

Proof of Concept

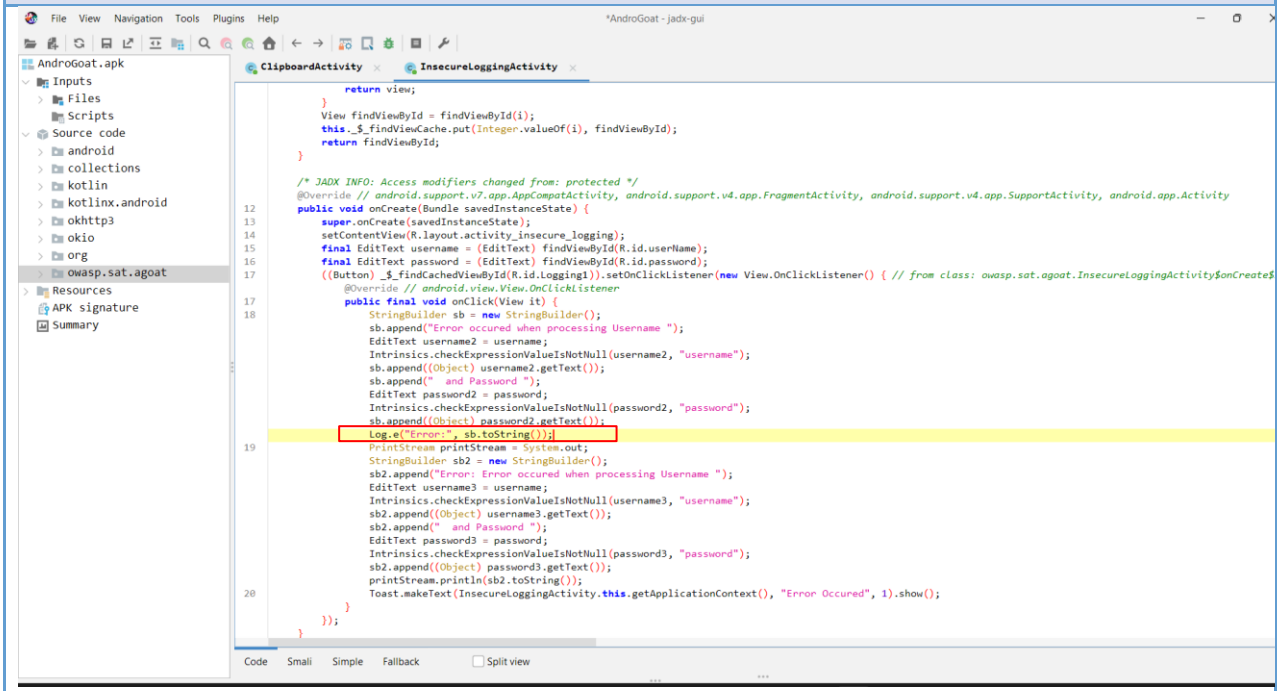


Figure 20. Class InsecureLoggingActivity.java

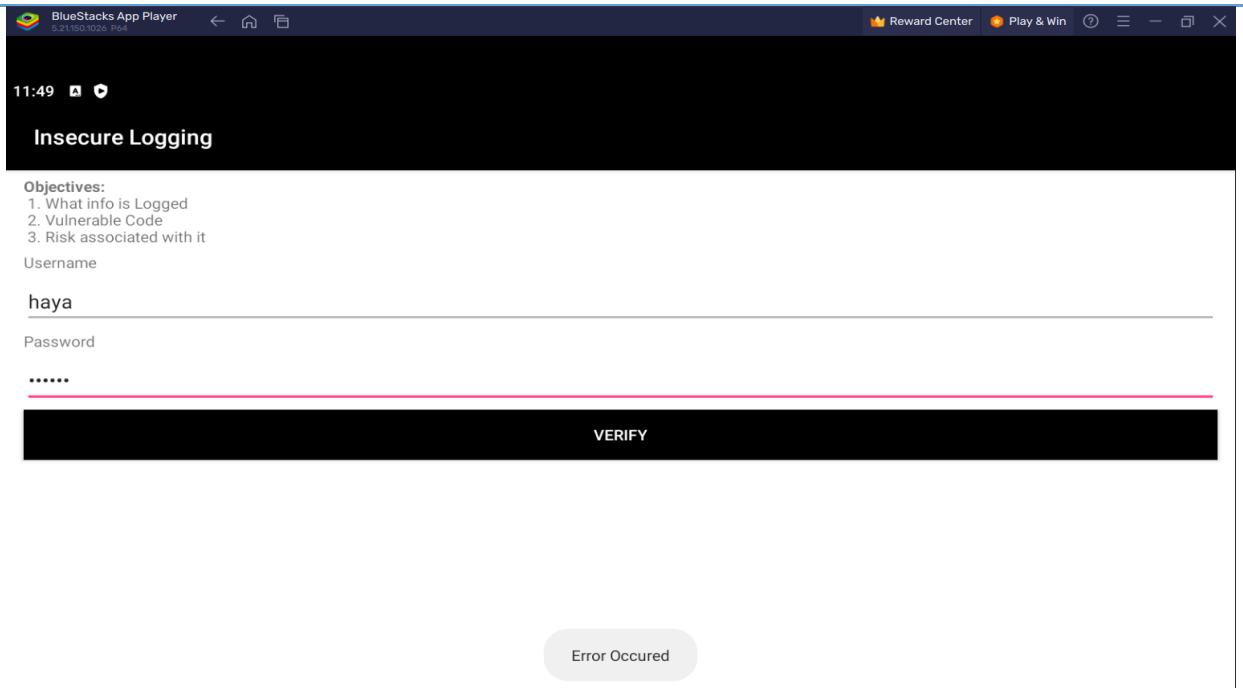


Figure 21. Enter insecure Username and Password.

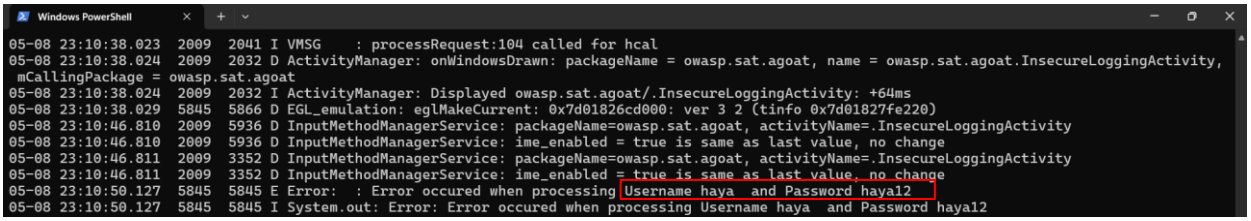


Figure 22. Windows PowerShell show username and password as plaintext.

Recommendation	1. Use secure logging methods: Logging sensitive data should only be done with proper. encryption and secure storage mechanisms. Logs must be protected from unauthorized access and potential tampering. 2. Limit the amount of data logged: To reduce the likelihood of sensitive data being exposed, only log the data necessary for diagnostic purposes.
OWASP Reference	https://tyk.io/blog/res-owasp-api-security-10-insufficient-logging-monitoring/#:~:text=Insufficient%20Logging%20%26%20Monitoring%20is%20not,the%20impact%20of%20the%20attack
Reference	https://docs.cycubix.com/application-security-series/web-application-security-essentials/solutions/sensitive-data-exposure/insecure-login-2

Appendix A: About the Team

Team Code: ATeam05		
Student Name	**Serial Number**	**Role**
Haya Alhawaimel	443200917	All work done together
Sarah Aldbasi	443200520	All work done together