جامعة الملك سعود
King Saud University
كلية علوم الحاسب والمعلومات
قسم تقنية المعلومات

King Saud University
College of Computer and Information Sciences
Department of Information Technology

IT371: Application Security
2nd Semester 1445 H

External Application Penetration Testing Technical Report

For AndroGoat

Supervised By: **L.** Nora Madi.

Document Info

Item	Description
Document Title	**External Application Penetration Testing Technical Report For AndroGoat .**
Requestor	**L. Nora Madi**
Author/s	**Hind Alhijailan – Wiam Baalahtar.**
Date Created	**7/5/2024**

Table of Content

Table of Table

Table of Figure

Executive Summary

1.1 Introduction

The purpose of this Mobile Penetration Testing (MPT) report is to deliver a thorough evaluation of the security measures, emphasizing five vulnerabilities within the designated Android package (APK) named AndroGoat.apk. This APK has been intentionally designed to contain multiple vulnerabilities, making it an ideal subject for comprehensive testing and analysis. Furthermore, to facilitate effective MPT, three key tools were employed: Jadx, MobSF, and the Android Debug Bridge (ADB). These tools collectively establish an environment conducive to mobile penetration testing. Throughout this project, our primary goal is to thoroughly document detailed observations, impacts, and recommendations pertaining to these vulnerabilities. We will utilize these tools, along with a structured methodology outlined in subsequent sections, to conduct our testing. Ultimately, our efforts will contribute to enhancing the overall security posture of the application, thereby fortifying it against potential cyber threats.

Throughout this report, we navigate through various sections, each contributing crucial insights to our MPT project. Firstly, in the "Scope" section, we define the specific scope of our project, outlining the applications under scrutiny. Moving forward, in "Risk Rating," we delve into an intricate assessment of the risks posed by identified vulnerabilities, evaluating their potential impact on organizational operations, and assigning overall risk ratings based on their exploitability by adversaries. Subsequently, in "Threat Security Level," vulnerabilities are meticulously categorized into critical, high, medium, low, and informational levels, providing a comprehensive understanding of their severity and implications for the organization's security posture. In "Summary Table" and "Summary Graph," we present succinct summaries of our findings, encapsulating the total number of vulnerabilities discovered and their distribution across different severity levels. Furthermore, "Key Findings" offers a distilled overview of the most significant discoveries. Transitioning to "Conclusion," we consolidate our findings and draw overarching conclusions regarding the security status of the tested applications, providing actionable recommendations for mitigation and improvement. In "Methodology," we elucidate our approach to Mobile Penetration Testing (MPT), drawing inspiration from the Mobile Open Web Application Security Project (OWASP) and outlining the steps followed in our testing procedures. Finally, in "Detailed Findings," we embark on a detailed exploration of identified vulnerabilities. This section is subdivided into "Limitation," where we delineate any constraints encountered during testing, and "Technical Description of Findings," where we provide comprehensive details of each vulnerability, including technical impacts, proofs of concept, recommendations, and relevant references. Through the meticulous navigation of each section, we aim to provide stakeholders with a holistic understanding of the vulnerabilities discovered, their potential implications, and actionable insights for enhancing the security posture of mobile applications.

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.	Windows	Black Box

Table 1:project scope

1.3 Risk Rating

The risk rating for the issues and their impact on the operation of the organization is explained in table 2 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 the 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 stakeholders 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	1	1	-

Table 3:Mobile Application Penetration Testing

1.6 Summary Graph

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

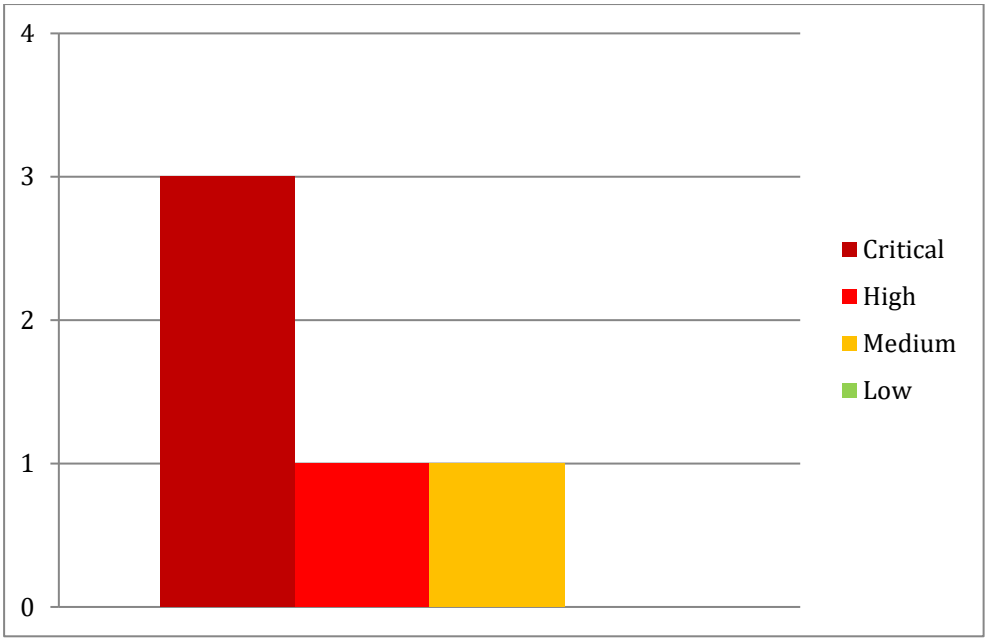


Figure 1: Application Penetration Testing

1.7 Key Findings

No.	Vulnerabilities Discovered	Platform	Severity Level
1	Hardcoded issue	Android	High
2	SQL injection	Android	Critical
3	Debuggable APK in Production	Android	Critical
4	Target SDK Version Exploit	Android	Critical
5	Unprotected Android Components	Android	Medium

Table 4:Key Findings

2 Conclusion

The Mobile Penetration Testing (MPT) assessment of the AndroGoat.apk revealed critical vulnerabilities that pose significant risks to the application's confidentiality and integrity. These vulnerabilities, including hardcoded issues, SQL injection, debuggable APK in production, target SDK version exploit, and unprotected Android components, were systematically identified, analyzed, and documented. Each vulnerability presents unique challenges and potential security implications for the application and its users.

Exploiting the hardcoded Promocode vulnerability allows attackers to bypass payment processes and obtain products for free, potentially leading to financial losses for the company. Furthermore, the exposure of sensitive data through debuggable APK in production and the exploitation of SQL injection vulnerabilities pose severe risks to user privacy and application security. Additionally, the target SDK version exploit exposes users to known vulnerabilities in outdated Android versions, increasing the risk of data breaches and malware infections. Additionally, the weak protection of Android components enables unauthorized access via ADB commands, Compromising user authentication.

Clearly, current security measures are not effective enough to address these vulnerabilities, and in order to mitigate these vulnerabilities, developers must implement robust security measures, such as avoiding hardcoded values for sensitive data, utilizing parameterized queries to prevent SQL injection, and disabling debuggable mode in production builds. Strengthening authorization controls and implementing proper access controls can prevent unauthorized access to sensitive activities and mitigate the risks associated with unprotected Android components. Increase the minimum SDK version in the app's (AndroGoat.apk) manifest file. Additionally, strengthening authorization controls to prevent unauthorized access to sensitive activities and mitigate the risks of unauthorized access via ADB commands is crucial. Before launching AccessControl1ViewActivity, ensure authenticated users have the necessary permissions.

In conclusion, addressing these vulnerabilities is essential to enhance the overall security posture of AndroGoat.apk and protect it against potential cyber threats. By following the recommendations outlined in this report and complying with secure development practices, developers can mitigate risks, protect user data, and ensure the long-term security and integrity of the mobile application.

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 4 phases:

- **planning and preparation:**
  In preparation for our Mobile Penetration Testing (MPT) project focused on the androGoat.apk application, we have diligently planned and organized our approach. Our

primary objective is to identify and document vulnerabilities within the androGoat.apk file, ultimately enhancing its security. To achieve this, we have identified crucial tools: Jadx, MobSF, and Android Debug Bridge (ADB). Additionally, we have incorporated Bluestacks. These tools are essential components of our testing, and we have ensured their download and installation as following:

- Bluestack5:

BlueStacks, an Android emulator, is a versatile addition to our toolkit for mobile penetration testing. It creates a virtual Android device on our computer, allowing us to test applications across various Android versions and device configurations Anyway, we downloaded BlueStacks through the link and then we uploaded Androgoat.apk within the program.

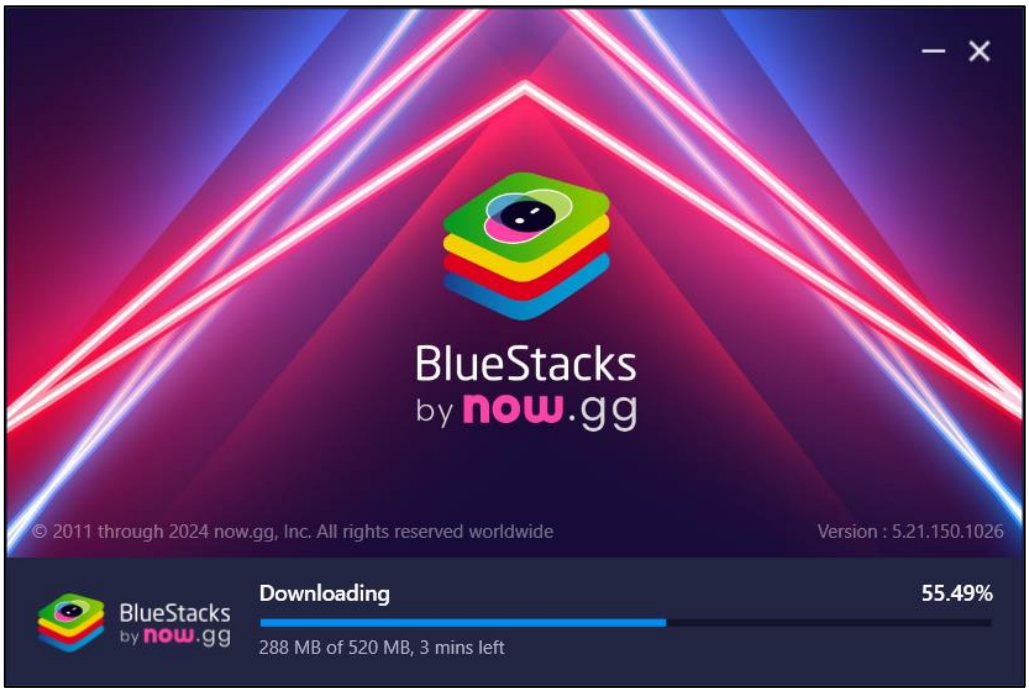


Figure 2: Downloading BlueStack.

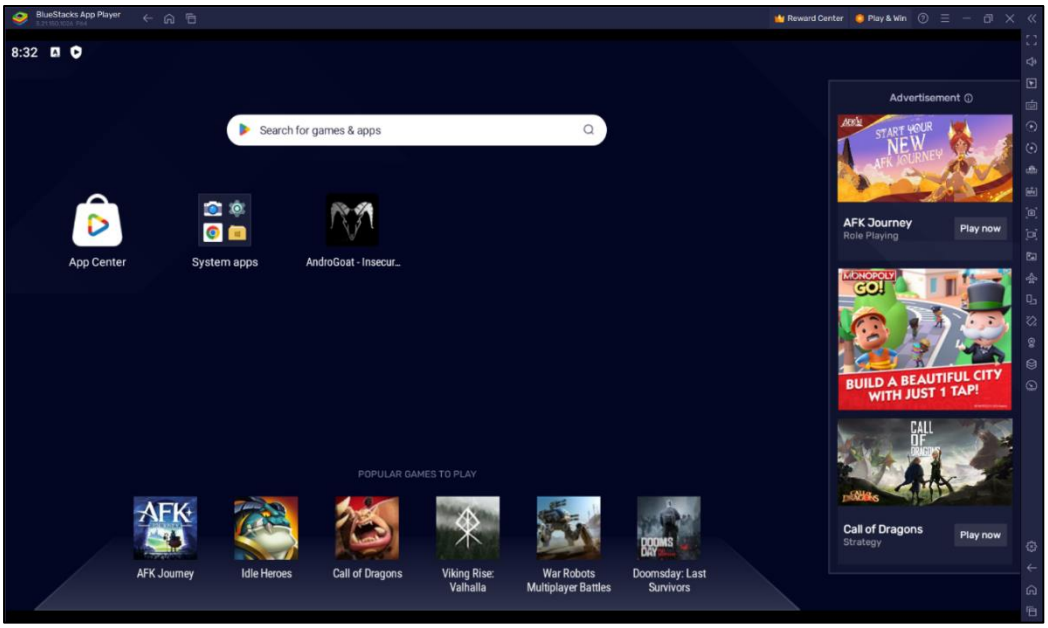


Figure 3:BlueStack Interface

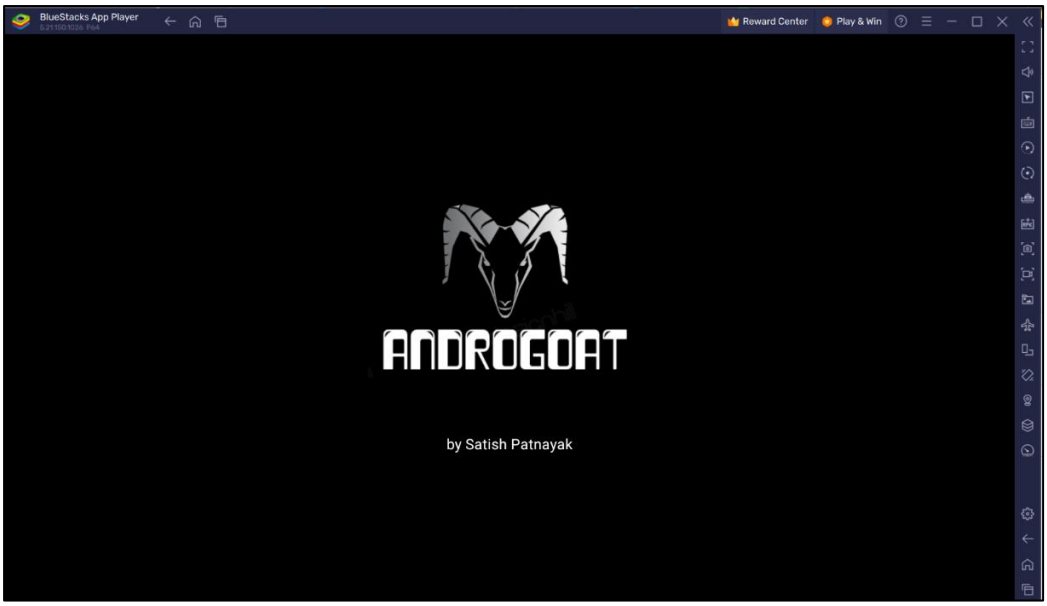


Figure 4:Open AndroGoat in BlueStack

- Jadx:

Because JADX is essential to the decompilation of APK files, we include it in our toolbox for mobile penetration testing. With the help of jadx, we can investigate androgoat.apk's internal workings and see the resources, source code, and other crucial elements. With the help of this technology, we can increase the entire security posture of mobile apps and find hidden https://github.com/skylot/jadx/releases/tag/v1.4.7.

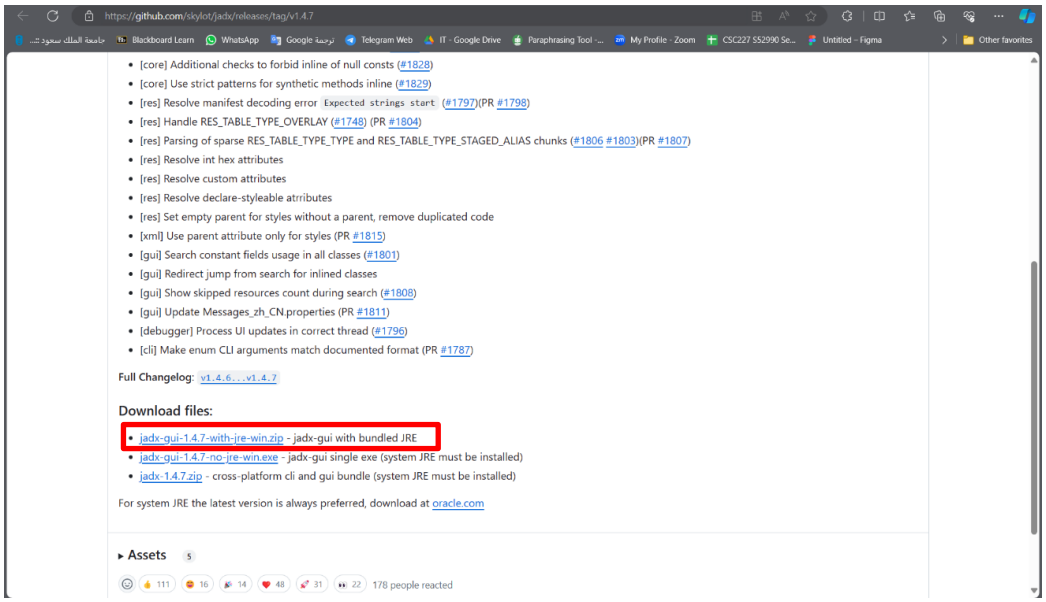


Figure 5:github repository for dowenload jadx

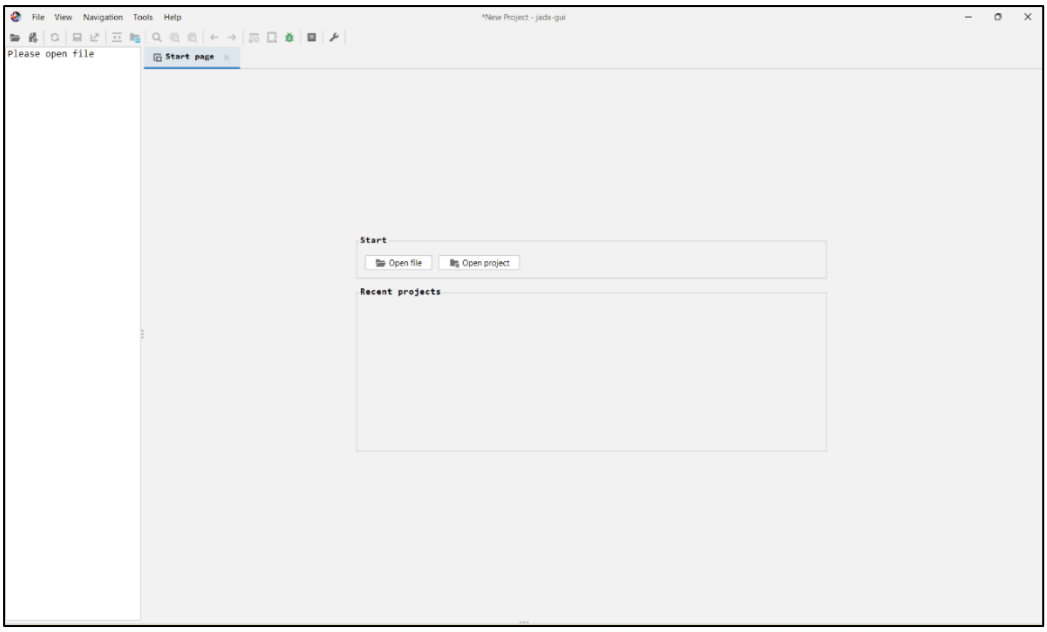


Figure 6:Jadx Interface

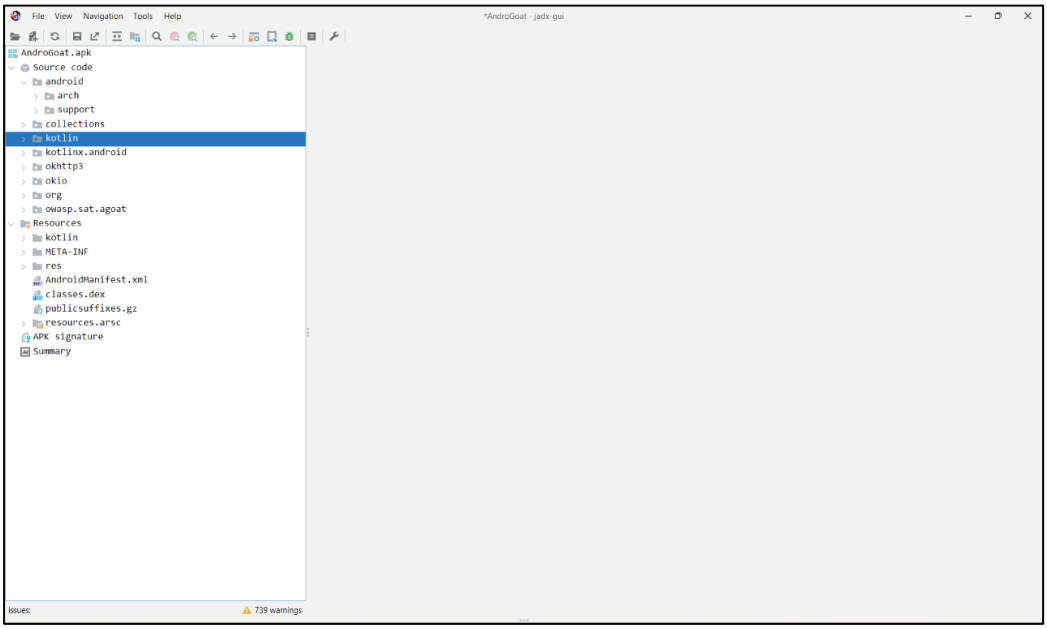


Figure 7:Open AndroGoat in Jadx.

- MobSF:

We used MobSF (Mobile Security Framework) as an integral part of our toolkit for mobile penetration testing due to its robust capabilities in conducting static analysis of mobile applications. MobSF offers a comprehensive suite of features designed to identify security vulnerabilities, assess the overall security posture, and generate detailed reports. By leveraging MobSF, we can scrutinize the androGoat.apk file thoroughly, examining its code, resources, and configurations for potential vulnerabilities. We obtain it from MobSF.live in the browser and then drag and drop the androGoat.

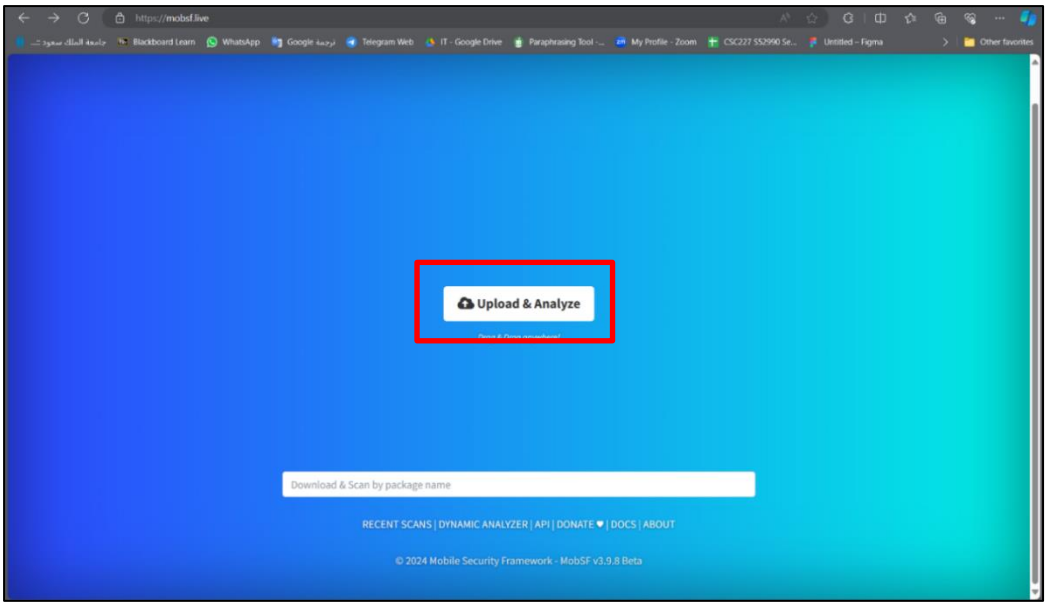


Figure 8:Browsing MobSF and uploading the AndroGoat.

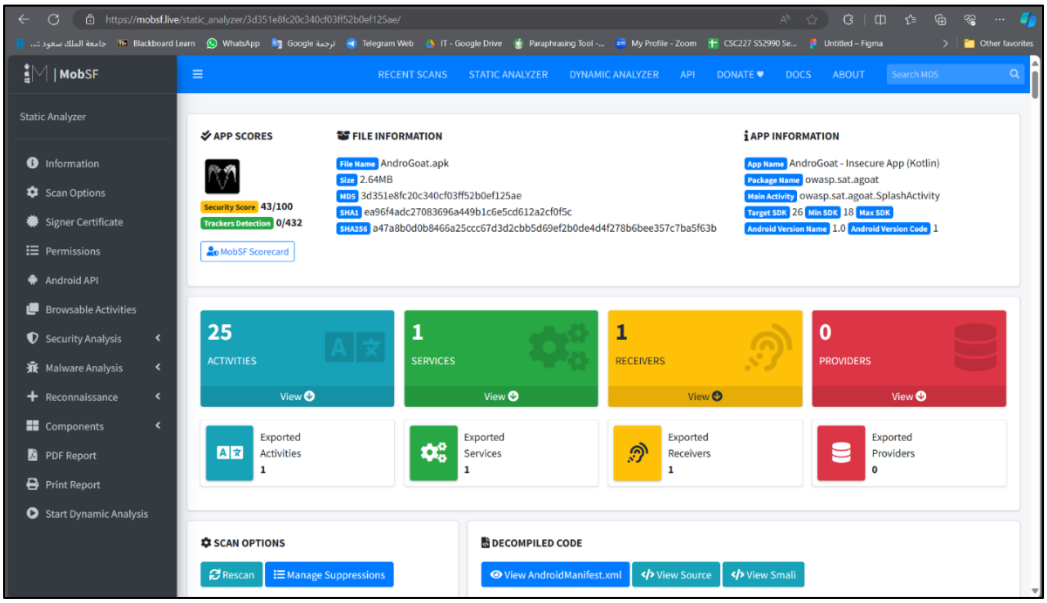


Figure 9:Analsis APK file in mobSF.

- Android Debug Bridge (ADB):

ADB is a command-line tool that enables developers to establish a connection with the emulators. It forms an essential part of the Android SDK (Software Development Kit) and is primarily used for debugging and testing purposes, by providing access to a Unix shell on the connected device or emulator, enabling developers to execute

commands directly on the device's operating system, interact with the file system, and perform various system-level tasks.

In our Mobile Penetration Testing (MPT) project, we leverage ADB as a critical tool for interacting with the emulators during testing. Specifically, we utilize ADB for tasks such as:

- Installing AndroGoat on the emulator.
- Interacting with the emulator’s file system to manipulate it.
- Executing shell commands on the device to exploit vulnerabilities.

Setting ADB up steps:

- Downloaded the Android SDK Platform-Tool on the host device through this link, then extracted the contents of the downloaded ZIP.
- From the setting of the blueStack we enabled "Android Debug Bridge".
- Opened the unzipped folder in the terminal and entered the command “./adb devices” then the emulator serial appeared on the screen which means the connection was success.
- open a shell session directly on the emulator by running the following command “./adb shell”, allowing us to execute commands within the emulator's environment.
- install the APK onto the emulator by running the following command “./adb install AndroGoat.apk”.

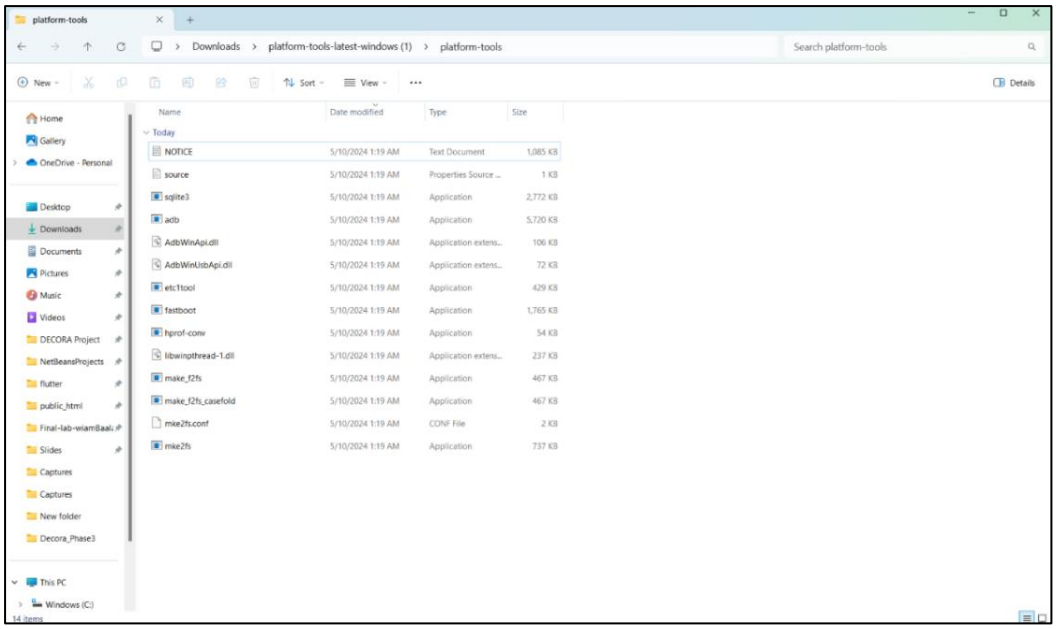


Figure10:Extact platforms tools folder that Containing the ADB file.

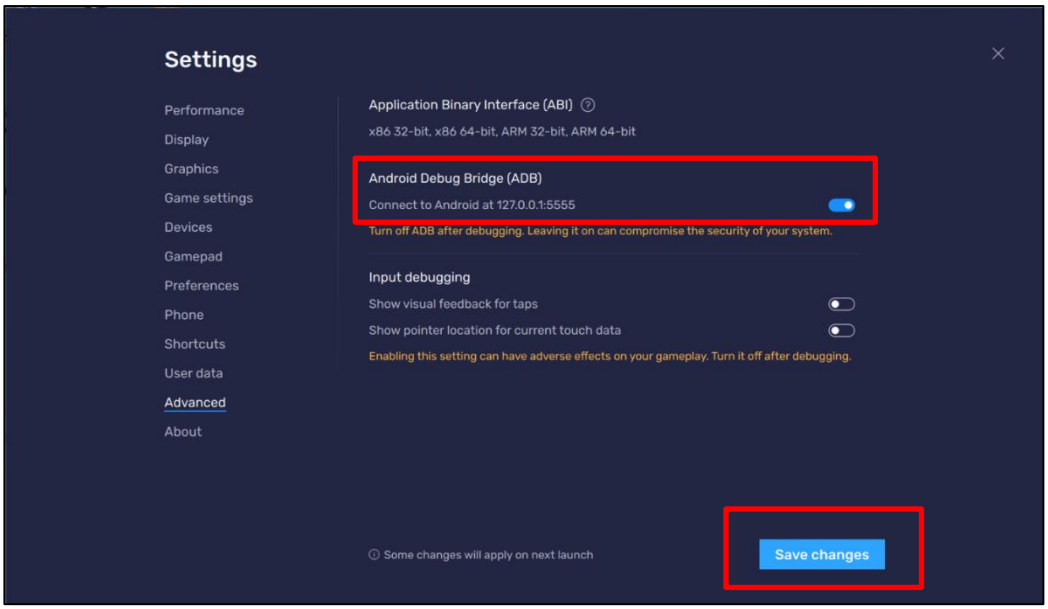


Figure 11:ADB settings.

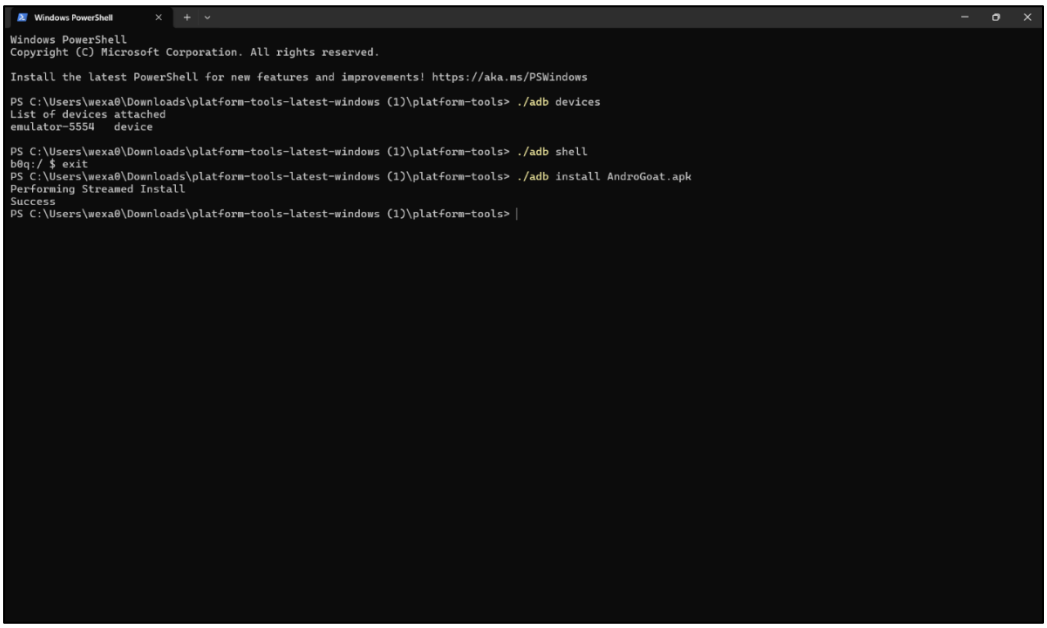


Figure 12:ADB commands showing device connection, shell access, and successful installation of AndroGoat.

- **Information gathering:**
  During this stage, our information gathering process was multifaceted, drawing insights from diverse sources to enrich our understanding of the target application, we extensively utilized educational content available on platforms like YouTube, leveraging tutorials, walkthroughs, and demonstrations to deepen our knowledge of mobile penetration testing techniques and methodologies, GitHub repositories, OWASP guidelines, and cybersecurity websites. In addition to that, we have gained a lot of knowledge regarding the vulnerabilities we are expected to find during scanning the source code and going through the MobSF listed vulnerabilities, which helps us understand in more depth how penetration testing is accomplished. These resources provided us with valuable insights into mobile penetration testing techniques, tools, and best practices. By leveraging diverse sources, we enhanced our understanding of the target application's architecture, functionality, and potential vulnerabilities, laying a strong foundation for our testing efforts, In addition to that we have gained a lot of knowledge regarding the vulnerabilities we are expected to find during scanning the source code and going through the mobsf listed vulnerabilities which help us understand in more depth how penetration testing is accomplished.

- **Risk modeling and risk rating:**
  In conducting our mobile penetration testing project on AndroGoat.apk, we employed a comprehensive risk modeling and rating methodology to evaluate the identified vulnerabilities. Our approach began with meticulous identification of key risk factors, encompassing technical impact and likelihood of exploitation. We carefully assessed each vulnerability against established criteria, including the CIA triad (Confidentiality, Integrity, Availability), popularity, and simplicity of exploitation. While some vulnerabilities were rated on the OWASP website, others were not, prompting us to conduct additional research to determine their real-world popularity and simplicity of attack. Through rigorous data collection and analysis, we gained insights into the severity and potential consequences of each vulnerability. Using a structured scoring system, we systematically assigned risk ratings to categorize vulnerabilities[1] into Critical, High, Medium, or Low, based on their impact on the application's security posture. Our methodology ensured a thorough assessment, providing actionable insights to prioritize mitigation efforts and enhance the overall security of AndroGoat.apk.

- **Reporting:**
  Employing a structured template, we provided a comprehensive overview of the assessment, including the executive summary, scope, methodology, detailed findings, practical recommendations, and conclusion. The executive summary briefly summarized key vulnerabilities and their severity, serving as a snapshot for stakeholders. A clear delineation of the assessment's scope and objectives defined the testing environment and approach. We detailed our methodology, drawing from frameworks like OWASP, to outline steps such as planning and preparation, information gathering, risk modeling and rating. In the detailed findings section, vulnerabilities were thoroughly described,

[1] OWASP Risk Rating Methodology | OWASP Foundation

supported by proofs of concept and mitigation measures. Actionable recommendations were tailored to address specific security gaps. The conclusion summarized key insights and recommendations, providing a roadmap for improvement. Precise review and approval processes ensured accuracy and alignment with organizational standards. Our objective was not only to conduct a comprehensive Mobile Penetration Testing (MPT) assessment on the androGoat.apk application but also to enhance its overall security posture through the identification, analysis, and mitigation of vulnerabilities. Moreover, addressing these vulnerabilities and providing actionable recommendations aim to contribute to our growth in the field of MPT.

4 Detailed Findings

4.1 Limitations

During our mobile penetration testing project, we encountered several challenges that impacted the depth and thoroughness of our assessment. Initially, connecting ADB to the emulator proved to be a hurdle, compounded by errors encountered in the command line interface (CLI). Additionally, many commands used for attacks were unrecognized by the CLI, necessitating troubleshooting efforts. We addressed these issues by rectifying path discrepancies and seeking external resources such as internet searches and tutorials on platforms like YouTube.

Determining the severity level of each attack was also to some extent confusing, requiring meticulous examination and assessment. This aspect highlighted the importance of adaptability and resourcefulness in navigating the complexities of mobile penetration testing. Furthermore, some assessed vulnerabilities might have dependencies on other attacks or system components. Due to scope constraints, we may not have elaborated on these interconnected vulnerabilities fully. This limitation may restrict the thoroughness of the assessment and potentially overlook certain security risks.

Time constraints were another factor that impacted our assessment as it may have been limited by time constraints, impacting the selection and prioritization of vulnerabilities for analysis. Moreover, some tools encountered errors during dynamic attacks, preventing thorough exploration of certain vulnerabilities. As a result, the assessment team may not have been able to thoroughly review every vulnerability or conduct dynamic attacks on all identified weaknesses due to time limitations. Consequently, the assessment may not have prioritized the most critical vulnerabilities or fully explored their potential impact.

These limitations underscore the complexity of mobile penetration testing and highlight the need for careful planning, resource management, and adaptability to effectively navigate the assessment process.

4.2 Technical Description of Findings

4.2.1 Hardcoded issue

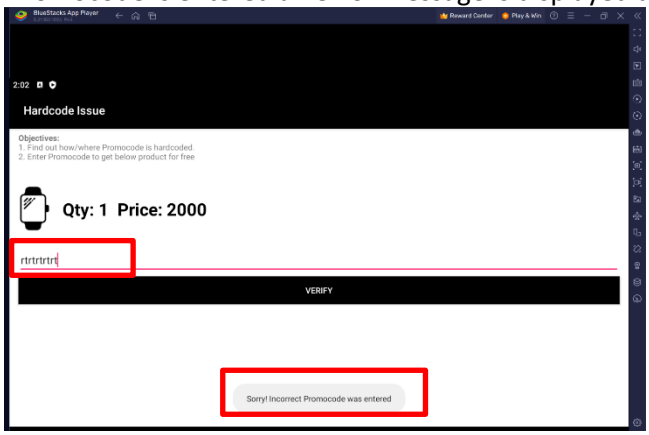
<table>
<tr><td>Severity Level</td><td colspan="4">High, considering that Promocode is not critical data, but it effects the confidentiality of code, and the likelihood is high since it is very popular and simple to exploit.</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">Medium</td><td colspan="2">Likelihood</td><td>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 absence of dynamic handling of Promocodes makes the application more vulnerable to exploitation, because with just a simple search process, an attacker can uncover this vulnerability, gaining access to the hard-coded Promocode, and bypass payment processes and obtain the product for free.</td></tr>
<tr><td>Impact</td><td colspan="4">Exploiting the hardcoded Promocode could cause significant technical risks to the application's security and possibly integrity. An attacker might get products for free that he's not supposed to, which could potentially lead to financial losses for the company. If the attacker utilizes reverse engineering tools, it will be very easy to extract hardcoded data from other places in the source code that might contain more sensitive data than the Promocode.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
<tr><td colspan="5">In order to get the product for free we need to have the Promocode. Using "BlueStack5" emulator, If incorrect Promocode is entered an error message is displayed as shown:
 

Figure 13: incorrect Promocode.

Using “jadx” search feature, we tried to search the source code for some related keywords, thinking we might find the promocode hard coded, and we will probably recognize it. So we searched the word “code” and couldn’t find it and then we searched” PromoCode” and we found two possible results</td></tr>
</table>

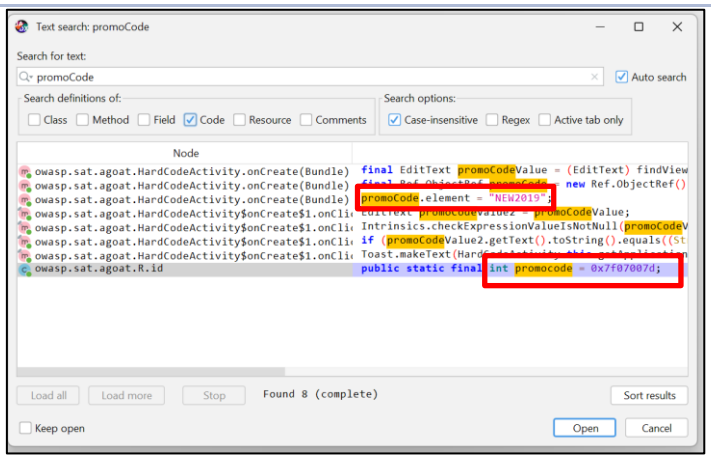


Figure 14:Search "Jadx" feature.

We tried the first one and it did work, and the product become free.

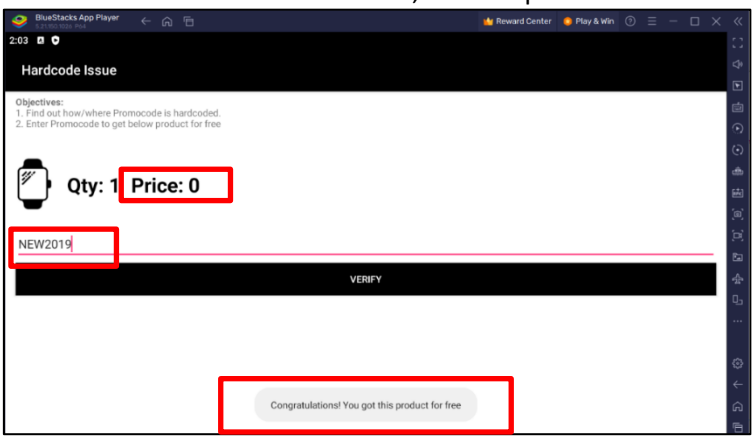


Figure 15: Successfully Attack-Hardcoded issue.

Recommendation	To mitigate the hardcoded Promocode vulnerability, developers should avoid hardcoded values for sensitive data, such as Promocodes, within the application's source code. Instead, utilize secure storage mechanisms, such as encrypted databases and server-side validation, to manage Promocodes dynamically. Implement input validation checks to ensure that only valid Promocodes are accepted, reducing the risk of exploitation.
OWASP Reference	**Use of hard-coded password**
Reference	https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets

4.2.2 SQL injection

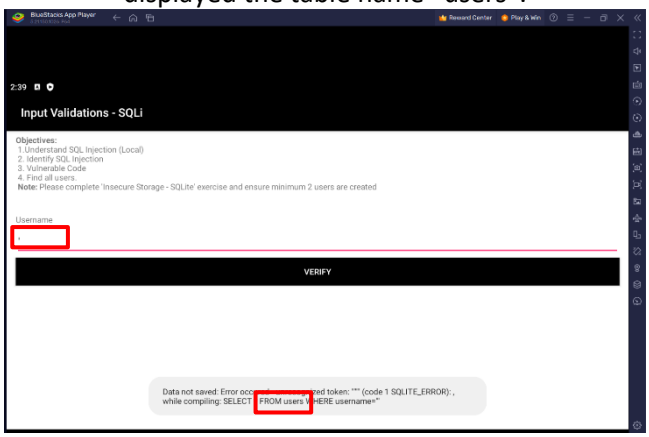
<table>
<tr><td>Severity Level</td><td colspan="4">Critical, since the vulnerability may impact conditionality, availability and integrity "CIA" of application and its lead to the access of critical data, since it is very simple to implement and consider very popular we rated the likelihood as high.</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">This vulnerability arises due to the improper construction of SQL queries, where user-controlled input, obtained from the "username" field, is directly concatenated into the query without validation or sanitization, and if any person enters special chars, then it will get reflected into database or data will be retrieved from database and the error message should not be as. Moreover, the application responds with an error message, indicative of susceptibility to SQL injection.</td></tr>
<tr><td>Impact</td><td colspan="4">returning all records from the users table, which could lead to unauthorized access and data leakage.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
<tr><td colspan="5">When we initially entered a single apostrophe using the "BlueStack5" emulator, we got an error message which often indicates that the input field is vulnerable to SQL injection. This is because the apostrophe may disrupt the SQL query's syntax, leading to a database error. The error message displayed the table name “users”.
 

Figure 16:Vulnerable input filed.

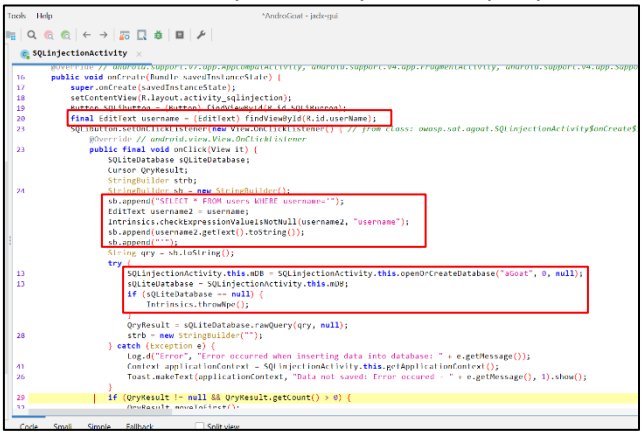
Using "Jadx" to read the “source” code, this code is vulnerable to SQL injection since it constructs an SQL query by concatenating user-controlled input directly into the query without proper validation or sanitization. By appending the value of “username” which is retrieved using “getText().toString()” from the user’s input directly into the query.
 

Figure 17:SqLinjectionActivity class.</td></tr>
</table>

In this injected query, the condition '1'='1' will always evaluate to true, effectively bypassing username-based authentication checks and returning all records from the users table.

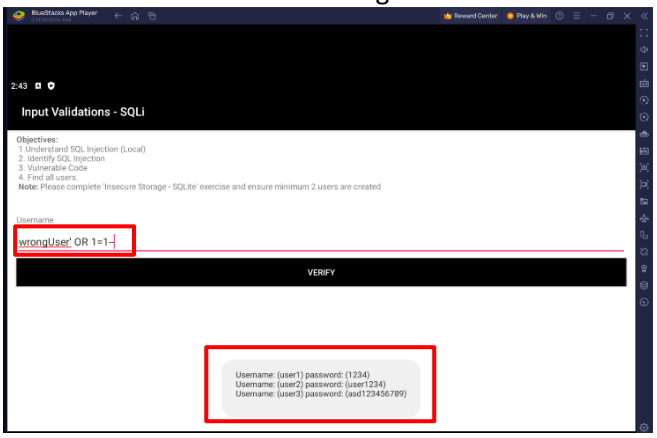


Figure 18:Find all users.

Recommendation	To prevent SQL injection vulnerabilities, developers should use parameterized queries or prepared statements with placeholders instead of concatenating user input directly into SQL queries. Additionally, input validation and proper error handling should be implemented to detect and mitigate potential injection attempts.
OWASP Reference	SQL Injection
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

4.2.3 Debuggable APK in Production

Severity Level	**Critical,** this assessment aligns with OWASP's rating, categorizing it as common and easy to exploit, with a severe technical impact, and can have significant consequences for application security and user privacy. Therefore, immediate attention and mitigation measures are crucial to address this critical vulnerability effectively.				
Technical Impact	High	**Likelihood**		High	
		Popularity	High	**Simplicity**	Medium
Observation	The MobSF analysis showed "Debuggable APK in Production" vulnerability which describes a situation where an Android application's production build is unintentionally configured to be debuggable. After reviewing the apk file in jadx, we note that BuildConfig.java is compiled with debugging enabled.				
Impact	When an application is debuggable, it compromises user privacy and application security. It allows debuggable builds, exposing sensitive information, leading to data breaches and unauthorized access, and undermining user trust.				
Platform	Android				
Proof of Concept					
We were to find and investigate the vulnerabilities in the AndroGoat.apk file by using the "mobSF" application to assist with Penetration Testing. "Debuggable APK in Production" is one of these vulnerabilities that we discovered.					

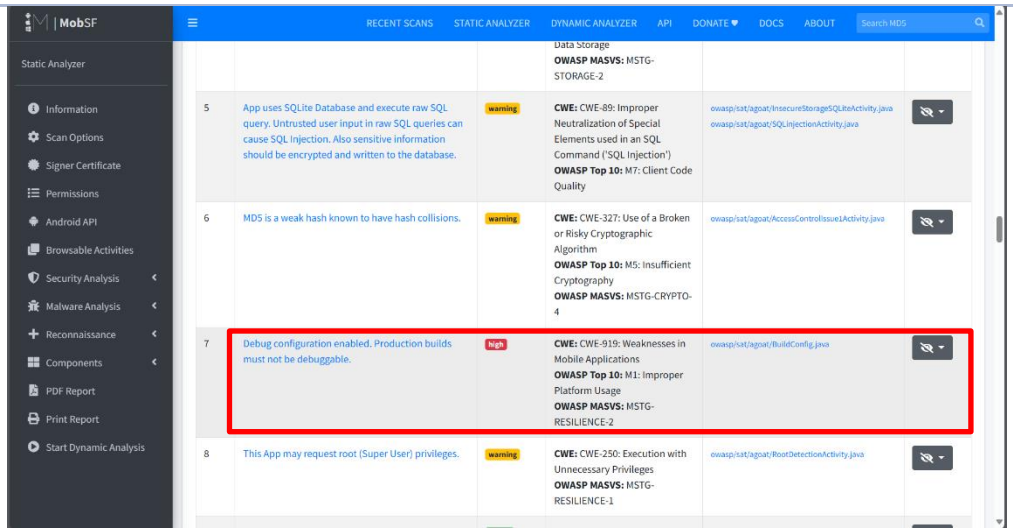


Figure 19:MobSF vulnerability analysis.

By using "jadx" to read the “source” code, the presence of DEBUG = "true" in the BuildConfig class means that the application's production build is compiled with debugging enabled.

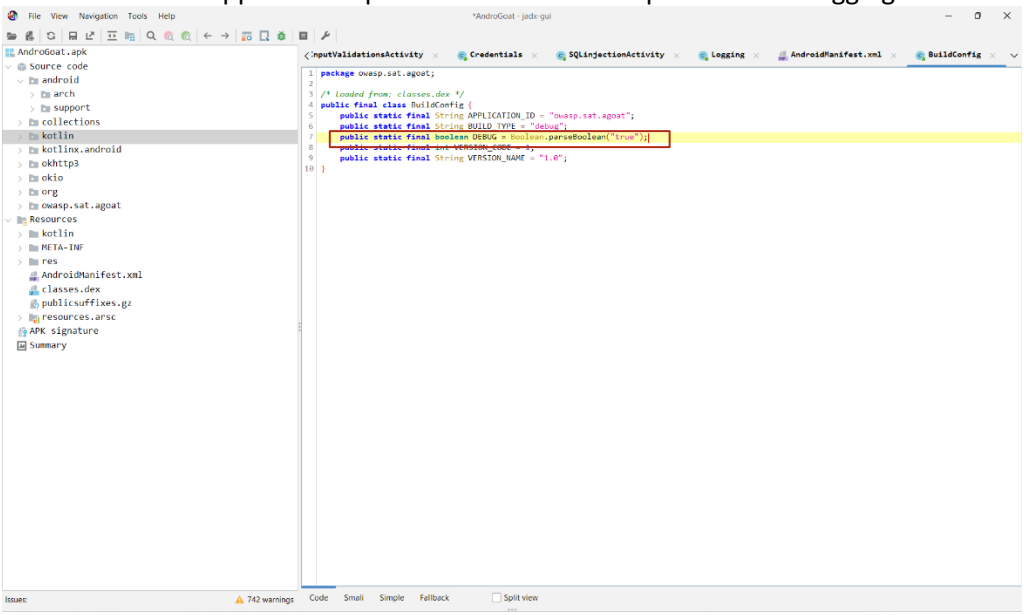


Figure 20:BulidConfig class

Recommendation	To mitigate this vulnerability, developers need to ensure that debuggable mode is disabled (ex. DEBUG = ''False'') in production builds and adhere to secure development practices to prevent unauthorized access and exploitation of vulnerabilities.
OWASP Reference	Improper Platform Usage
Reference	https://www.infosecinstitute.com/resources/application-security/android-hacking-security-part-6-exploiting-debuggable-android-applications/ https://security.stackexchange.com/questions/74209/what-are-the-possible-exploits-over-a-debuggable-apk

4.2.4 Target SDK Version Exploit

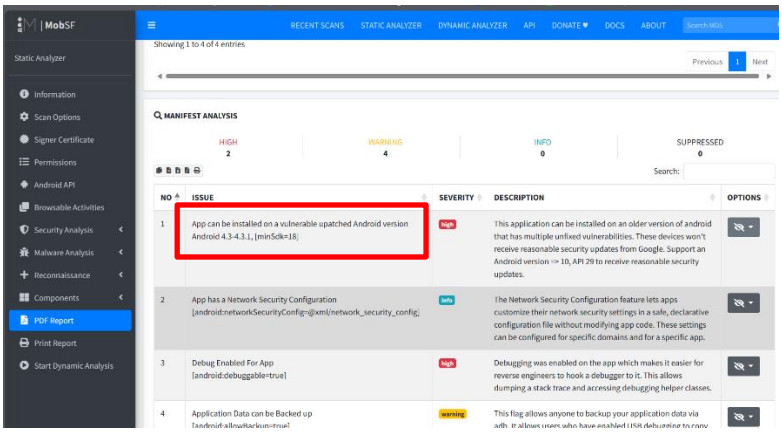
<table>
<tr><td>Severity Level</td><td colspan="4">Critical, it poses a severe impact on the affected application, requires immediate attention, and can potentially provide attackers with full control over the application and access to critical data.</td></tr>
<tr><td rowspan="2">Technical Impact</td><td rowspan="2">High</td><td colspan="2">Likelihood</td><td>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 observed vulnerability, termed "Target SDK Version Exploit," manifests in the application's ability to be installed on outdated and vulnerable Android versions 4.3 to 4.3.1 (API level 18), despite potential security risks associated with these versions. By targeting such obsolete SDK versions, the application bypasses crucial security enhancements and patches introduced in newer Android releases, leaving users exposed to known vulnerabilities.</td></tr>
<tr><td>Impact</td><td colspan="4">A low targetSdkVersion with a high minSdkVersion like this app (targeting 4.3 but allowing install on 4.3-4.3.1) creates a security risk. The app exposes itself to vulnerabilities patched in newer Android versions and might lack the security features of those newer versions, making it easier for attackers to steal data or install malware on vulnerable devices.</td></tr>
<tr><td>Platform</td><td colspan="4">Android</td></tr>
<tr><td colspan="5">Proof of Concept</td></tr>
<tr><td colspan="5">We will be able to find and investigate the vulnerabilities in the AndroGoat.apk file by using the mobSF application to assist with a Pentest. Target SDK Version Exploit is one of these vulnerabilities that we discovered.
 

Figure 21:MobSF vulnerability analysis.

By using "jadx" to decompile Apk, we noticed that the provided Android manifest file (AndroidManifest.xml) sets the minSdkVersion to 18 (Android 4.3), allowing the app installation on older devices. This creates a vulnerability known as Target SDK Version Exploit.</td></tr>
</table>

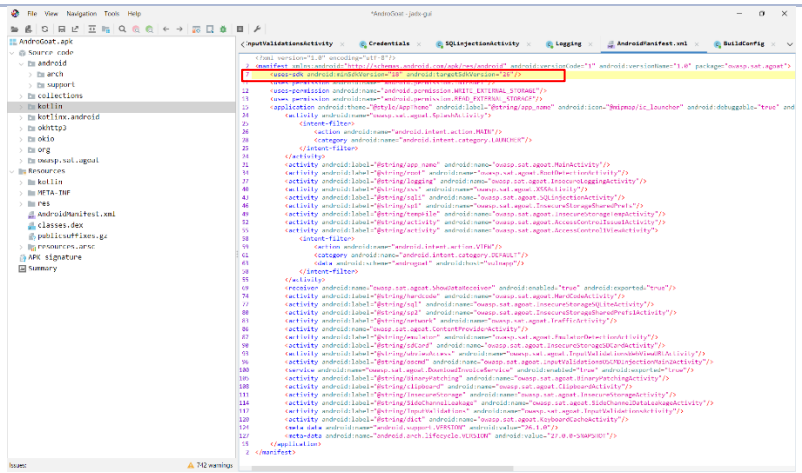


Figure 22:AndroidManifest.xml.

Recommendation	To mitigate the vulnerability, the developer should increase the minSdkVersion in the app's (AndroGoat.apk) manifest file. This will prevent the app from installing on outdated and insecure devices (like Android 4.3-4.3.1) that lack security patches, reducing the app's exposure to known exploits.
OWASP Reference	https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
Reference	https://developer.android.com/ndk/guides/sdk-versions#:~:text=Using

4.2.5 Unprotected Android Components

Severity Level	Medium, it is considered as simple since it requires just a basic level of knowledge with ADB commands and tools, and it is less common compared to other vulnerabilities. Bypassing the authentication factor, the attacker will only be able to access a single activity.				
Technical Impact	Medium.	**Likelihood**		Medium.	
		Popularity	Medium	**Simplicity**	Medium
Observation	The weak protection of Android components, as shown by the absence of robust validation checks or access controls, allowed unauthorized users to take advantage of ADB commands, bypassing intended security measures. Exploiting this vulnerability allows attackers to directly access activities without undergoing the required PIN authentication, which in itself is a weak method of user authentication.				
Impact	Enables unauthorized access to sensitive files, potentially compromising the confidentiality and integrity of the application's data.				
Platform	Android				
Proof of Concept					
We first generate a PIN using the "BlueStack" emulator, which is then used to log in and download an invoice file.					

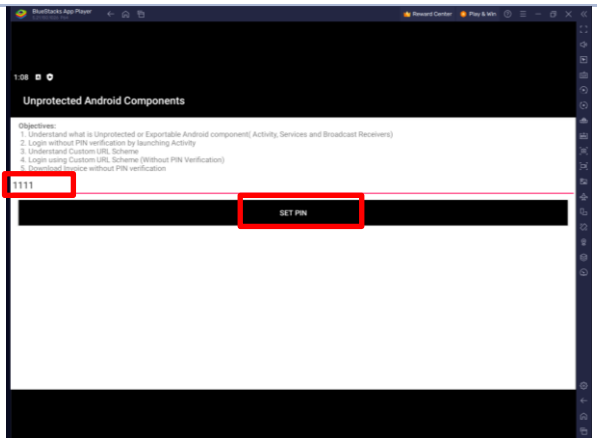


Figure 23:Set PIN.

You have to verify the PIN that you have previously created if you want access the invoice file.

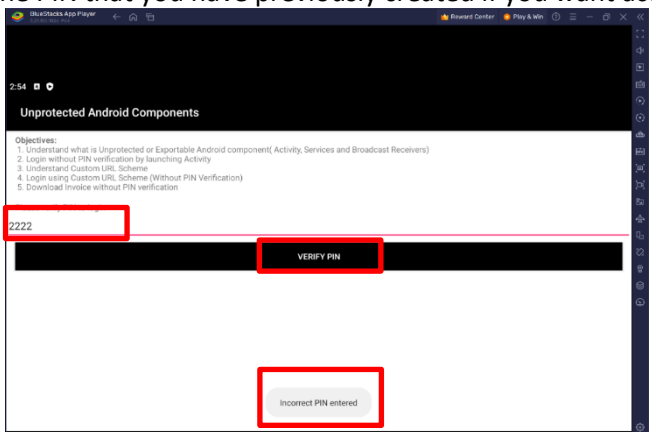


Figure 24:Incorrect PIN entered.

This code indicates that the "AccessControl1ViewActivity" activity will be triggered if the correct PIN is entered. "Jadx" is used to read the "source" code of AndroGoat.

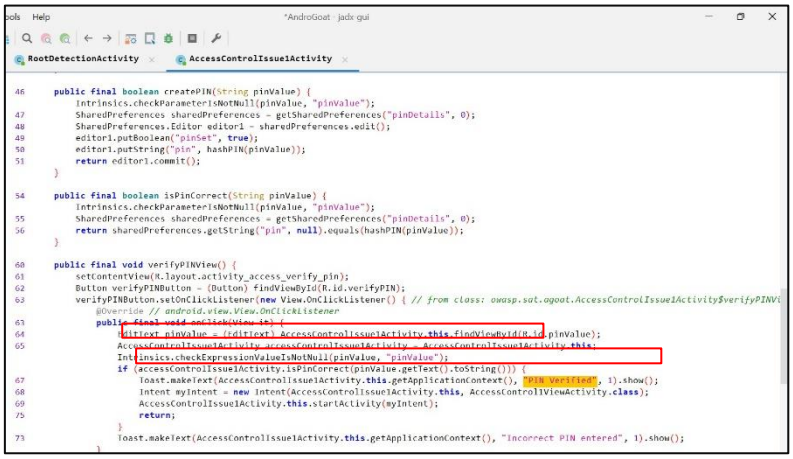


Figure 25:AccessControl1ViewActivity class.

We have connected the emulator with the CLI using 'ADB', and used ". **/**adb " shell am start -n owasp.sat.agoat/.AccessControl1ViewActivity" command to access the activity that has the invoice immediately without having to verify the PIN.

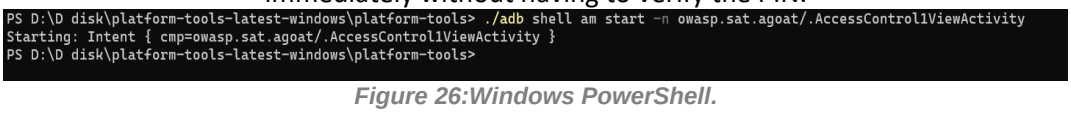


Figure 26:Windows PowerShell.

After executing the command successfully, the following activity that has the invoice will be accessed skipping the verify step:

Figure 27: Successfully attack - Unprotected Android Components.

Recommendation	Strengthen authorization controls to prevent unauthorized access to sensitive activities, and to mitigate risks of unauthorized access via ADB commands. Before launching AccessControl1ViewActivity, ensure authenticated users have the necessary permissions.
OWASP Reference	Insecure Authorization
Reference	https://medium.com/@baldraider/start-an-activity-with-adb-9f8edbe5d652#:~:text=You%20can%20use%20the%20start,as%20defined%20in%20the%20manifest.

Appendix A: About the Team

Team Code:	**A11**	
Student Name	Serial Number	Role
Wiam Baalahtar	443200416	Introduction – planning and preparation – Risk Modeling and risk rating– Information gathering – 'Target SDK Version Exploit' table – 'Debuggable APK in Production' table- Limitations.
Hind Alhijailan	443200971	Conclusion – Risk modeling and risk rating- planning and preparation – Reporting – Limitations – "Hardcoded issue" – "SQL injection" – "Unprotected Android Components".