KLEAP

CYBERSECURITY

Mobile Application Security Report

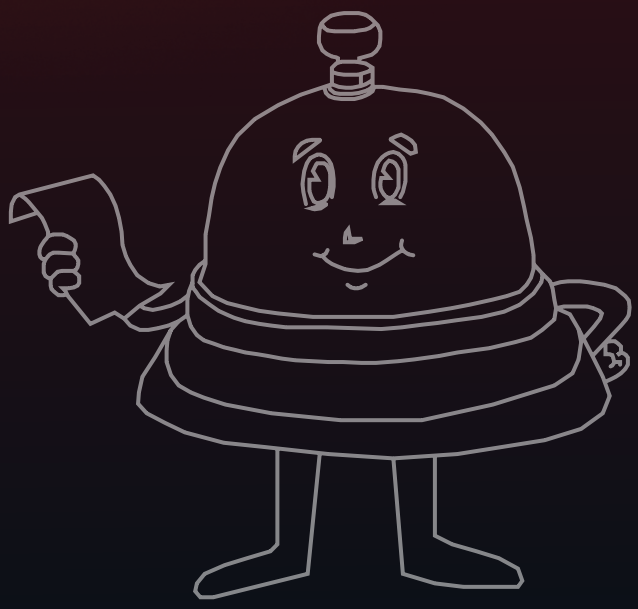


Table of Contents

Statement of Confidentiality

Engagement Contacts

Executive Summary

- Scoping and Time Limitations
- Testing Summary
- Vulnerability Status
- Recommendation

Scope Summary

- In-Scope Assets
- Out-of-Scope Assets

Methodology

- Engagement Phases
  1. Reconnaissance
  2. Scanning and Enumeration
  3. Vulnerability Assessment
  4. Exploitation
  5. Reporting
- Vulnerability Classification & Severity

Findings Summary

- Findings Overview
- Findings Overview as per OWASP Standards

Technical Findings Details

- 01: SQL Injection
- 02: Cross-Site Scripting
- 03: Directory Traversal
- 04: Server Version Disclosure

Statement of Confidentiality

This pentest report contains confidential and proprietary information belonging to KLEAP Technologies Pvt. Ltd. and Client. It is intended solely for the use of the Client and KLEAP Technologies Pvt. Ltd. The information provided within this report should not be disclosed, distributed, or shared with any third parties without the explicit written consent of both KLEAP Technologies Pvt. Ltd. and Client. Any unauthorized use or disclosure of this information is strictly prohibited and may result in legal action.

Executive Summary

Client engaged KLEAP Technologies Pvt. Ltd. to perform penetration testing of the APIs. The primary goal of this API penetration testing project was to identify any potential areas of concern associated with the API in its current state and determine the extent to which the system may be breached by an attacker possessing a particular skill and motivation. The assessment was performed in accordance with the “best-in-class” practices as defined by ISECOM's Open Source Security Testing Methodology Manual (OSSTMM) and Open Web Application Security Project (OWASP).

KLEAP Technologies Pvt. Ltd. conducted the penetration testing during the period of March 20th, 2024 to April 11th, 2024. All testing activities were performed on the staging environment provided by the customer and completely isolated from the production data. While performing the testing activities, KLEAP Technologies Pvt. Ltd. emulated an external attacker without prior knowledge of the environment. To test the user-authenticated area and privilege escalation vulnerabilities, the customer supplied KLEAP Technologies Pvt. Ltd. credentials for several registered user and admin accounts.

Scoping and Time Limitations

Scoping during the engagement did not permit denial of service or social engineering across all testing components.

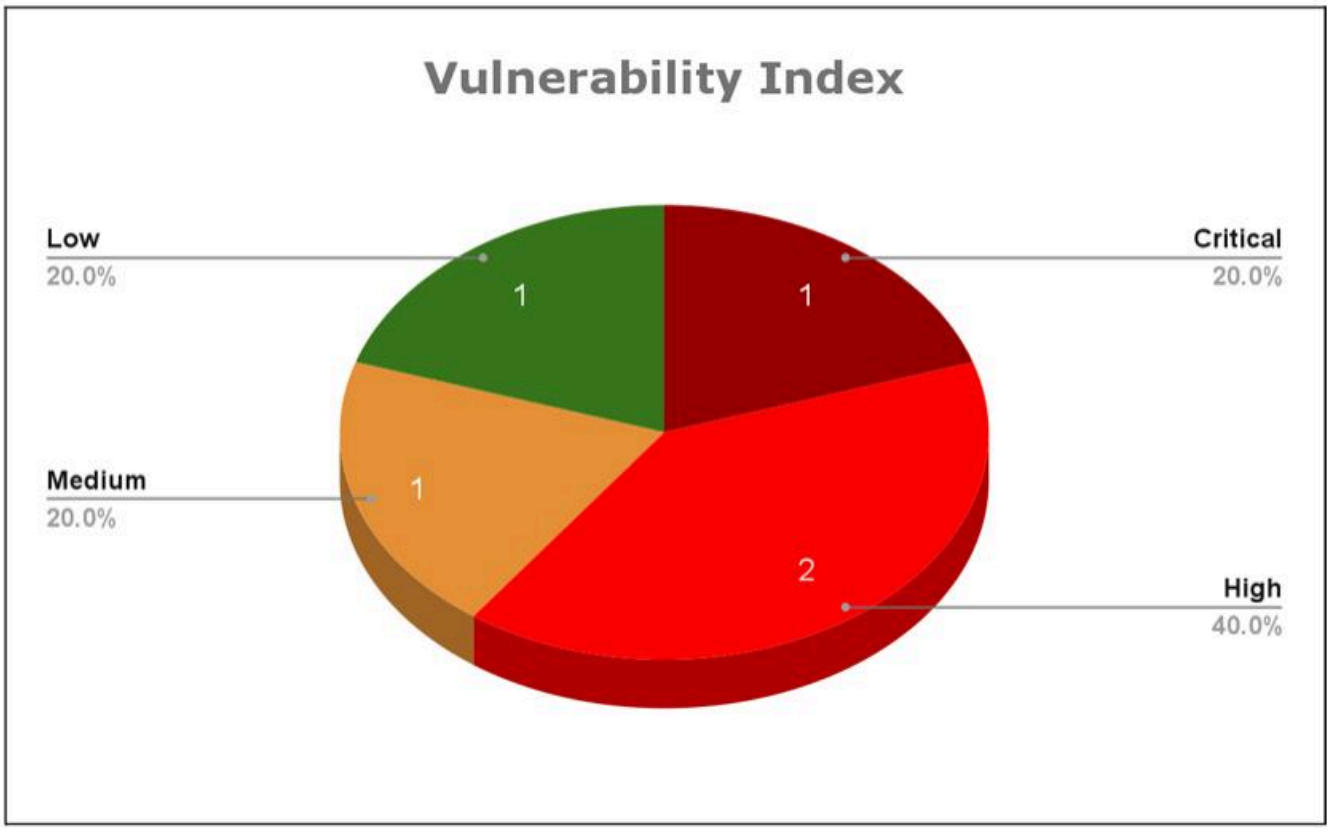
Time limitations were in place for testing. API penetration testing was permitted for seven (7) business days.

Testing Summary

(Overall Summary of the findings)

Scope	Critical	High	Medium	Low	Info	Total
Client AD	1	2	1	1	0	5

Table 1: Findings per asset



Vulnerability Index

Vulnerability Status

Sr. No.	Vulnerability	Severity	Status
1	Access Control Issues - Authentication Bypass	CRITICAL	OPEN
2	SSL Pinning Bypass	CRITICAL	OPEN
3	Access Control Issues - Login Bypass	HIGH	OPEN
4	Insecure Data Storage	MEDIUM	OPEN
5	Clipboard Vulnerability	LOW	OPEN

Recommendation

Based on the results of this assessment, KLEAP Technologies Pvt. Ltd. has the following high-level key recommendations.

Key Recommendation 1

Key Issue	The mobile application has Improper Privilege Management, Weak Authentication, and Insufficient Binary Protections.
Recommendation	Applying anti-hooking techniques.

Scope Summary

In-Scope Assets

The following assets were considered explicitly in-scope for testing:

Assets In-Scope	Hostname / CIDR / IP
Mobile Application	Test APK

Out-of-Scope Assets

(If any)

Assets Out-of-Scope	Hostname / CIDR / IP
N/A	N/A

Methodology

The pentest methodology employed by KLEAP Technologies Pvt. Ltd. follows a systematic approach to assess the security posture of client systems.

Our Penetration Testing Methodology is based on the following guidelines and standards:

- Penetration Testing Execution Standard
- OWASP Top 10 Application Security Risks
- OWASP Testing Guide
- SANS: Conducting a Penetration Test on an Organization
- The Open Source Security Testing Methodology

Engagement Phases

1. Reconnaissance

In this phase, penetration testers gather valuable information about the mobile application. This includes researching the app on various app stores and public sources to gain insights into its functionalities, user interface, and potential vulnerabilities. Testers may also review documentation, user manuals, and publicly available information to understand the app's features and behavior better.

2. Scanning and Enumeration

In this phase, analyzing the mobile app from an attacker's perspective to identify potential threats and attack vectors. Testers consider scenarios such as data breaches, unauthorized access, injection attacks, and other security risks that the app may be susceptible to.

3. Vulnerability Assessment

In this phase, Prioritizing Critical Components and Weak Points Testers prioritize critical components and weak points based on their potential impact on the app's security and business objectives. This step helps allocate testing efforts efficiently and address the most severe vulnerabilities first.

4. Exploitation

In this phase, Exploiting Vulnerabilities Ethically Testers demonstrate ethical exploitation of identified vulnerabilities to understand their potential impact on the app. Understanding the Impact of Successful Attacks The impact of successful attacks is assessed to comprehend the severity of vulnerabilities and potential consequences.

5. Reporting

In this final phase, the pentester compiles all findings, categorizes them based on severity levels, and provides detailed explanations, proof-of-concept demonstrations, and prioritized recommendations for remediation. The report includes a summary of the pentest engagement, an overview of the methodology used, and a comprehensive analysis of the vulnerabilities discovered. It also includes actionable recommendations to mitigate the identified vulnerabilities and improve the overall security posture of the target systems. The report serves as a valuable resource for the client to understand the security risks and take appropriate measures to address them.

Vulnerability Classification & Severity

To categorize vulnerabilities according to a commonly understood vulnerability taxonomy, KLEAP Technologies Pvt. Ltd. uses the industry-standard Common Weakness Enumeration (CWE). CWE is a community-developed taxonomy of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

To rate the severity of vulnerabilities, KLEAP Technologies Pvt. Ltd. uses the industry standard Common Vulnerability Scoring System (CVSS) to calculate severity for each identified security vulnerability. CVSS provides a way to capture the principal characteristics of a vulnerability, and produce a numerical score reflecting its severity, as well as a textual representation of that score.

To help prioritize vulnerabilities and assist vulnerability management processes, KLEAP Technologies Pvt. Ltd. translates the numerical CVSS rating to a qualitative representation (such as low, medium, high and critical):

CVSS Score v3.1	
Severity	**Score**
Critical	9.0 - 10.0
High	7.0 - 8.9
Medium	4.0 - 6.9
Low	0.1 - 3.9
Informational	0.0

Findings Summary

Findings are sorted by their severity and grouped by the asset and CWE classification. Each asset section will contain a summary. Table 1 in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

Findings Overview

During the engagement, 5 unique vulnerabilities were found across 3 different vulnerability categories. The 3 most common vulnerabilities identified were:

- Insecure Communication
- Insufficient Binary Protections
- Insecure Data Storage

Exploring the findings further by their actual vulnerability type as defined by CWE, Table 3 shows the number of individual findings and its distribution of severity.

Vulnerabilities	Critical	High	Medium	Low	Info
Access Control Issues - Authentication Bypass	1	0	0	0	0
SSL Pinning Bypass	0	1	0	0	0
Access Control Issues - Login Bypass	0	1	0	0	0
Insecure Data Storage	0	0	1	0	0
Clipboard Vulnerability	0	0	0	1	0
	1	2	1	1	0

Table 3: severity distribution across vulnerability types

Findings Overview as per OWASP Standards

Vulnerabilities	Results	Findings
M1:2023 Improper Credential Usage	Pass	0
M2:2023 Inadequate Supply Chain Security	Pass	0
M3:2023 Insecure Authentication/Authorization	Fail	2
M4:2023 Insufficient Input/Output Validation	Pass	0
M5:2023 Insecure Communication	Fail	1
M6:2023 Inadequate Privacy Controls	Pass	0
M7:2023 Insufficient Binary Protections	Fail	1
M8:2023 Security Misconfiguration	Pass	0
M9:2023 Insecure Data Storage	Fail	1
M10:2023 Insufficient Cryptography	Pass	0

Technical Findings Details

01: Access Control Issues - Authentication Bypass

Vulnerability Severity	CWE ID
Critical	0
OWASP Category	CVSS Score
Insecure Authentication/Authorization	0

Vulnerability Description

A Broadcast Receiver is found to be shared with other apps on the device, therefore, leaving it accessible to any other application on the device. It is protected by permission which is not defined in the analyzed application. As a result, the protection level of the permission should be checked where it is defined.

Vulnerable Application

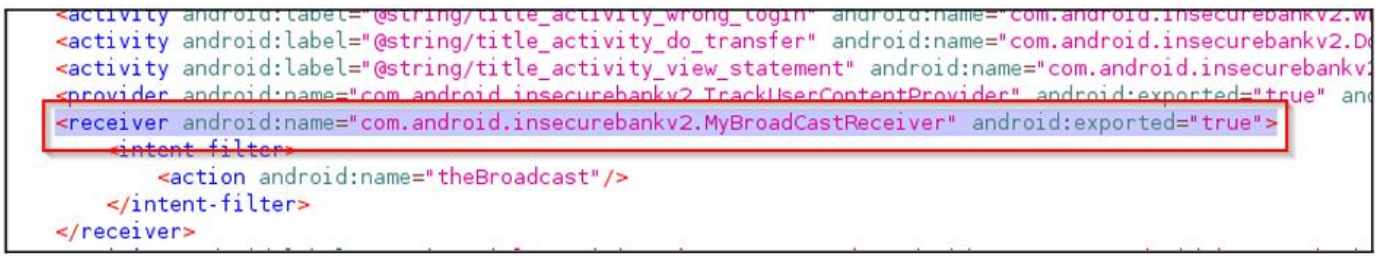
Test Apk

Impact

An SMS message can be sent by an attacker. Malicious actors can profit from forcing consumers to send messages against their will and setting a premium rate SMS number.

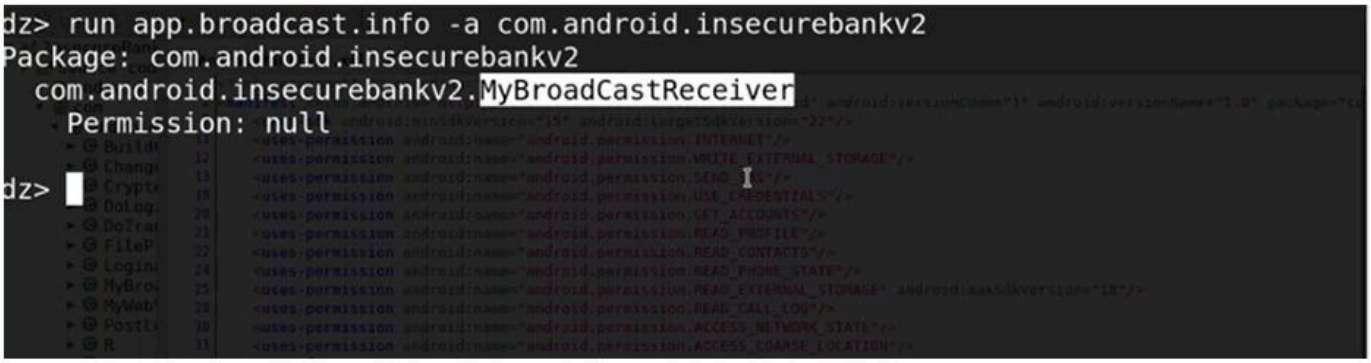
Steps to Reporduce

1. In static source code analysis using the apk source code reading tool jadx-gui, in the AndroidManifest.xml file we found Broadcast Receiver com.android.insecurebankv2.MyBroadCastReceiver is protected by a.permission, but the protection level of the permission should be checked Permission: android.permission [android:exported=true]



Steps to Reporduce

2. Upon examining the code, we discovered that the broadcast receiver is transmitting the new password, or "newpass," to the user's phone number via the smsManager function.



3. Given that the myBroadcastReceiver is exported: true, any phone number can receive an SMS with the updated username and password.

```
try {
    SharedPreferences settings = context.getSharedPreferences("mySharedPreferences", 1);
    this.usernameBase64ByteString = new String(Base64.decode(settings.getString("EncryptedUsername", null), 0), "UTF-8");
    String decryptedPassword = new CryptoClass().aesDeccryptedString(settings.getString("superSecurePassword", null));
    String textPhoneno = phn.toString();
    String textMessage = "Updated Password from: " + decryptedPassword + " to: " + newpass;
    SmsManager smsManager = SmsManager.getDefault();
    System.out.println("For the changepassword - phonenumber: " + textPhoneno + " password is: " + textMessage);
    smsManager.sendTextMessage(textPhoneno, null, textMessage, null, null);
} catch (Exception e) {
    e.printStackTrace();
}
```

Remediation

- Changing the value of the android:exported attribute to false.
- Setting unique permissions for every receiver in order to restrict access. This is useful if the developer wants to grant permission-holding apps just access to the components of their app.

References

https://resources.infosecinstitute.com/topic/android-hacking-security-part-3-exploiting-broadcast-receivers/

02: SSL Pinning Bypass

Vulnerability Severity	CWE ID
High	0
CVE ID	CVSS Score
Insecure Communication	0

Vulnerability Description

With SSL pinning, an application can only rely on a public key or valid, pre-defined certificate. The SSL pinning technique is employed by the application developer to provide an extra security layer for application traffic. As usual, the program permits the application to intercept the traffic since it trusts a custom certificate. However, the application under the SSL Pinning implementation does not accept custom certificates and prevents traffic from being intercepted by proxy tools

Vulnerable Application

Test APK

Impact

Using SSL Pinning, which an attacker can circumvent, could result in the following attacks:

1. A hostile actor who intervenes in the communication between the client (user/app) and the server is referred to as a man-in-the-middle (MITM) attack.
2. Locating and taking advantage of vulnerabilities on the server side Following a successful pinning bypass, the intercepted requests may aid an attacker in deciphering the API request flow and identifying any server-side vulnerabilities that may not have been discovered previously.

Step to Reproduce

1. Static analysis of the source code in the HttpClientBuilder library to find SSL pinning code.

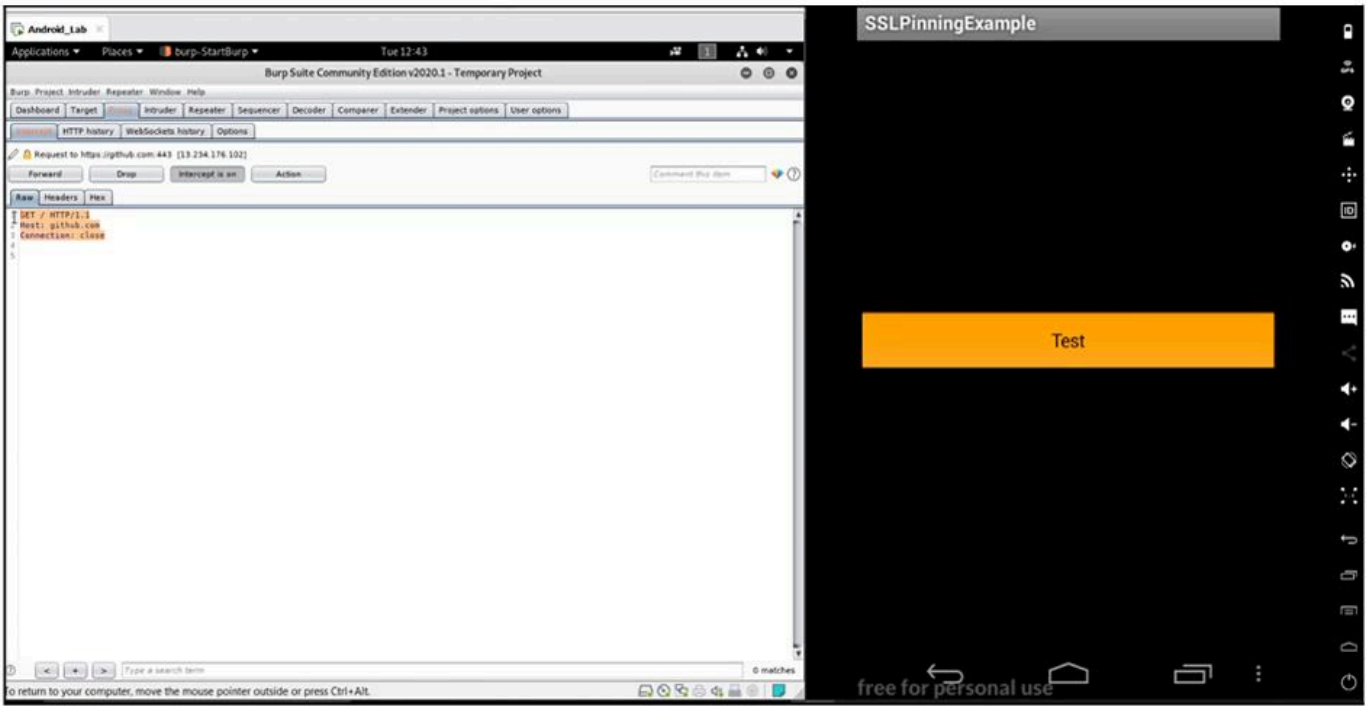
Step to Reproduce

```
public HttpClientBuilder pinCertificates(InputStream resourceStream, char[] password) throws KeyS
    this.keyStore = KeyStore.getInstance(BOUNCY_CASTLE);
    this.keyStore.load(resourceStream, password);
    return this;
}
```

2. Finding the same code location in the samli Code.

```
  .line 92
  const-string v0, "BKS"

  invoke-static {v0}, Ljava/security/KeyStore;->getInstance(Ljava/lang/String;)Ljava/security/KeyStore;

  move-result-object v0

  iput-object v0, p0, Lcom/example/sslpinningexample/HttpClientBuilder/HttpClientBuilder;-
keyStore:Ljava/security/KeyStore;

  .line 93
  iget-object v0, p0, Lcom/example/sslpinningexample/HttpClientBuilder/HttpClientBuilder;-
keyStore:Ljava/security/KeyStore;

  invoke-virtual {v0, p1, p2}, Ljava/security/KeyStore;->load(Ljava/io/InputStream;[C)V

  .line 95
```

3. Following the study of the static code, we moved on to the testing's dynamic phase, where we used the SSLpinningBypass apk to employ a framework to bypass SSL pinning.
4. Burp Suite, a network capture tool, is now able to record the request that is received while logging into the application after the Test Tool apk was implemented and the Test app's modifications were applied.



Remediation

Here are some mitigations to address a proxy bypass vulnerability in a mobile app :

- Applying anti-hooking: Detecting hooking applications like Frida and objection.
- Properly Enforce Proxy Settings: Ensure that the mobile app correctly adheres to the proxy settings configured on the device, preventing attackers from circumventing the proxy.
- Implement Secure Communication Protocols: Use secure communication protocols such as HTTPS to encrypt data transmitted between the app and its servers, preventing interception and tampering by attackers.
- Enforce Proxy Authentication: Require authentication for proxy connections to verify the identity of users or devices accessing the proxy server, reducing the risk of unauthorized access.
- SSL/TLS Pinning: Implement SSL/TLS certificate pinning to validate the authenticity of SSL/TLS certificates presented by servers, protecting against Man-in-the-Middle attacks that could bypass the proxy.

References

- https://obfuscator.re/omvll/passes/anti-hook/
- https://mobile-security.gitbook.io/mobile-security-testing-guide/iostesting-guide/0×06j-testing-resiliency-against-reverse-engineering

03: Access Control Issues - Login Bypass

Vulnerability Severity	CWE ID
High	0
CVE ID	CVSS Score
Insecure Authentication/Authorization	0

Vulnerability Description

It is discovered that an Activity is shared with other apps on the device, making it available to all other programs as well.

Vulnerable Application

Test APK

Impact

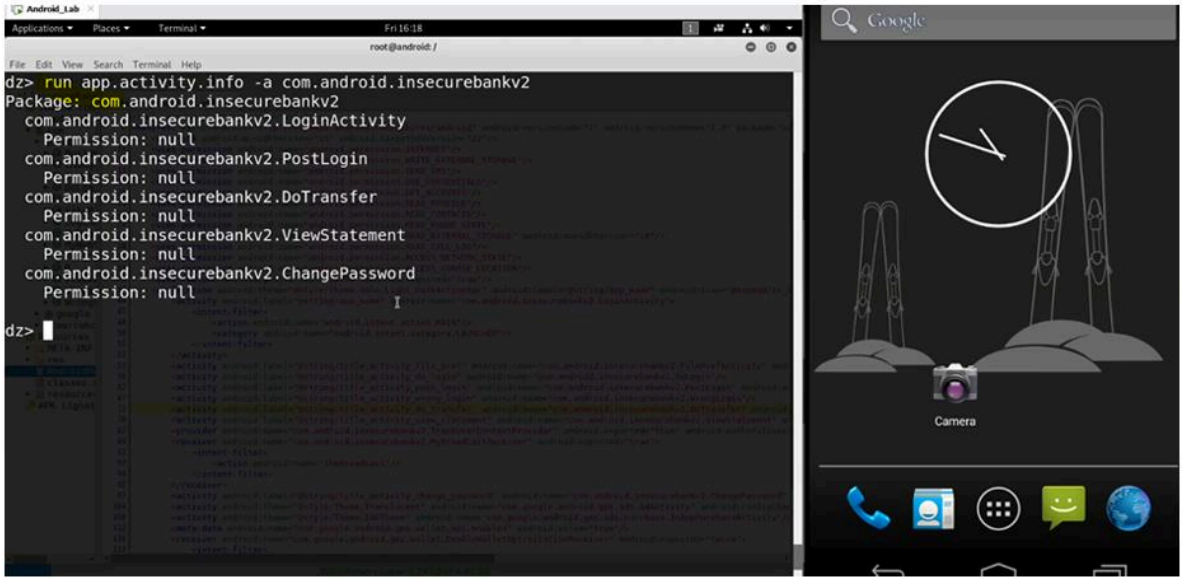
This category includes abusing a platform feature or not utilizing security protections on the platform. It could involve the exploitation of TouchID, the Keychain, Android intents, platform permissions, or another built-in security feature of the mobile operating system.

Step to Reproduce

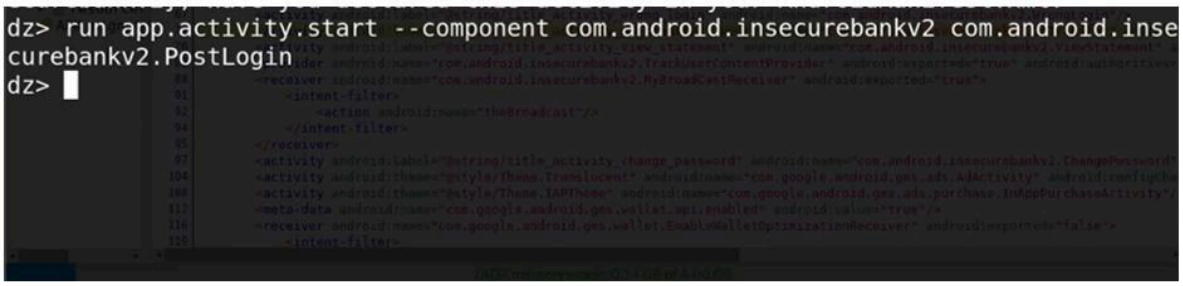
1. We discovered com.android.insecurebankv2.PostLogin, which is exported=true, in the AndroidManifest.xml file during static source code analysis utilizing the apk source coding reading tool jadx-gui.

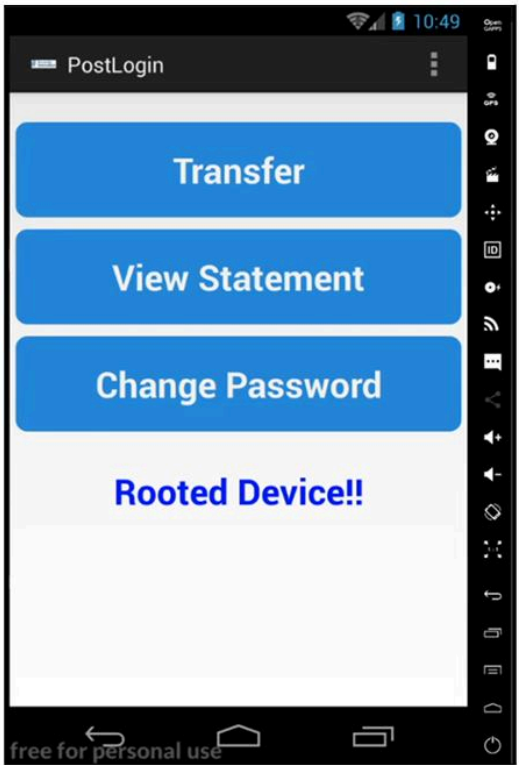
2. We also employed a run-time/dynamic analysis tool in the testing, which is installed on the mobile device we are utilizing for testing as well as the sandbox. It allowed us to identify the revealed actions. One of the actions that we observe is com.android.insecurebankv2. Post Login.

Step to Reproduce



3. Next, we attempted to launch the com.android.insecurebankv2 using the Drozer. After logging in, observe what transpires. The phone prompt should open for authentication, as predicted. The application launches and allows you to log in after you insert the pin. (App wasn't previously opened).





Remediation

- Changing the value of the android:exported attribute to false.
- Setting unique permissions for every receiver in order to restrict access. This is useful if the developer wants to grant permission-holding apps just access to the components of their app.

References

- https://developer.android.com/training/articles/security-tips.html#IPC

04: Insecure Data Storage

Vulnerability Severity	CWE ID
Medium	0
CVE ID	CVSS Score
Insecure Data Storage	0

Vulnerability Description

Insecure data storage can occur when apps store sensitive user information such as passwords, personal identification information, financial data, or other confidential information on the device's file system or database without proper encryption or protection.

Vulnerable Application

Test APK

Impact

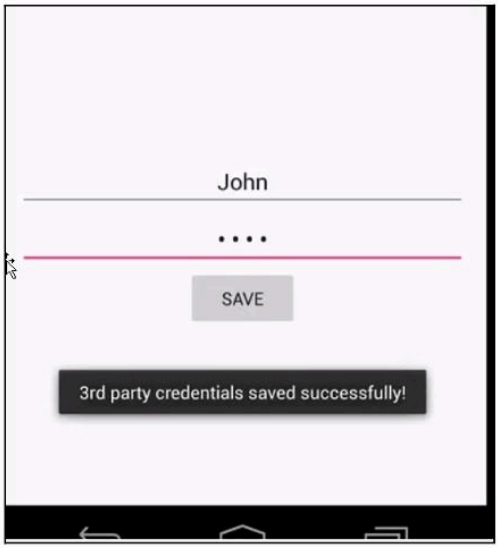
Insecure data storage in Android apps can lead to several severe impacts:

- Data Theft: Sensitive information, such as personal data, financial details, and login credentials, can be stolen by malicious actors.
- Privacy Breaches: Users' private information can be exposed, leading to identity theft or personal harm.
- Application Exploitation: Attackers can manipulate stored data to exploit the app, gaining unauthorized access or escalating privileges.

Step to Reproduce

1. From the application we are saving 3rd party credentials in the phone.

Step to Reproduce



2. During the analysis of the source code, we found the SQLiteDatabase has been imported, while checking it on the source code reading tool. In that, we found that the /data/data/jakhar.aseem.diva/databases/ids2 database was created.

```
@Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.su
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try {
        this.mDB = openOrCreateDatabase("ids2", 0, null);
        this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
    } catch (Exception e) {
        Log.d("Diva", "Error occurred while creating database: " + e.getMessage());
    }
    setContentView(R.layout.activity_insecure_data_storage2);
}

public void saveCredentials(View view) {
    try {
        this.mDB.execSQL("INSERT INTO myuser VALUES ('" + ((EditText) findViewById(R.id.ids2Usr)).getText(
        this.mDB.close();
    } catch (Exception e) {
        Log.d("Diva", "Error occurred while inserting into database: " + e.getMessage());
    }
    Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
}
```

3. In the hunt of finding the database we used an ADB (Android Debug Bridge is a versatile command-line tool that lets you communicate with a device) with its help, we scrolled through the internal storage of the phone and at location /data/data/jakhar.aseem.diva/databases/ids2 and we found id2 database, and while reading it we found Username, and Password in SQL database format.

```
root@android:/data/data/jakhar.aseem.diva/databases # sqlite3 ids2
SQLite version 3.7.11 [redacted]
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables;
Error: unknown command or invalid arguments:  "tables;". Enter ".help" for help
sqlite> .tables
android_metadata  myuser
sqlite> select * from myuser;
John|Cena
sqlite>
```

Remediation

Here are some remediations to prevent this vulnerability:

- Data Encryption: Encrypt sensitive data both in transit and at rest using strong encryption standards such as AES (Advanced Encryption Standard).
- Minimize Local Storage: Avoid storing sensitive information on the device whenever possible. Instead, use secure, remote servers with robust encryption and authentication mechanisms.
- Use Secure Containers: Store sensitive data in secure containers like SQLCipher for encrypted databases or use third-party libraries that provide secure storage solutions.
- Secure Storage APIs: Use secure storage options provided by Android, such as the Android Keystore for storing cryptographic keys and encrypted SharedPreferences for sensitive data.

References

- https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage

05: Clipboard Vulnerability

Vulnerability Severity	CWE ID
Low	0
CVE ID	CVSS Score
Insufficient Binary Protections	0

Vulnerability Description

The clipboard vulnerability poses a security risk on mobile devices due to the way mobile operating systems handle clipboard functionality. When users copy text or data on their devices, it's temporarily stored in the clipboard enabling them to paste it elsewhere. However, if not properly secured, unauthorized apps or websites can access the clipboard which potentially compromises sensitive information.

Vulnerable Application

Test APK

Impact

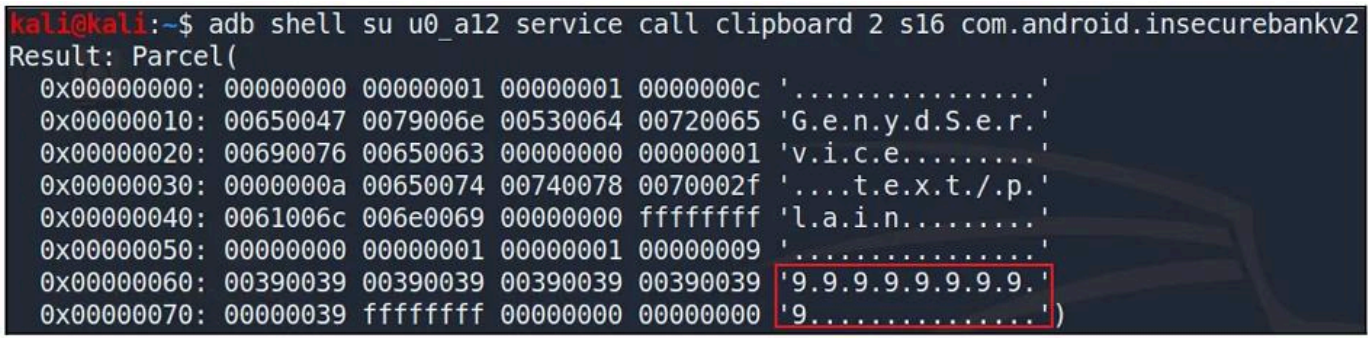
The impact of the clipboard vulnerability lies in its potential to compromise sensitive information stored on mobile devices. Unauthorized access to the clipboard can lead to the exposure of various types of data including passwords, credit card numbers, personal messages, and more. This exposure puts users at risk of identity theft, financial fraud, and invasion of privacy.

Step to Reproduce

1. Log into the account and copy any sensitive data to the clipboard as shown in the POC.

Step to Reproduce

2. It was observed that the copied clipboard data is being disclosed in the clipboard monitor through the adb too.



Remediation

Mitigating the clipboard vulnerability requires implementing several measures to enhance security on mobile devices:

- Applying anti-hooking: Detecting hooking application like frida and objection.
- Clipboard Access Controls: Mobile operating systems should implement stricter controls on clipboard access, allowing only authorized apps to read from and write to the clipboard.
- Encryption: Sensitive data stored in the clipboard should be encrypted to prevent unauthorized access even if the clipboard is compromised.
- Permission Management: Users should be informed about which apps have access to the clipboard and be given the option to revoke access for suspicious or unnecessary applications.

References

- https://redfoxsec.com/blog/protecting-android-clipboard-content/#:~:text=Despite%20its%20convenience%2C%20the%20Android,Pasteboard%20and%20Android%20Keyboard%20Cache.
- https://obfuscator.re/omvll/passes/anti-hook/
- https://mobile-security.gitbook.io/mobile-security-testing-guide/iostesting-guide/0x06j-testing-resiliency-against-reverse-engineering

KLEAP

CYBERSECURITY

https://kleapcybersecurity.com/

info@kleapcybersecurity.com

4111, Briargrove Circle, Raleigh,
North Carolina, 27607, USA