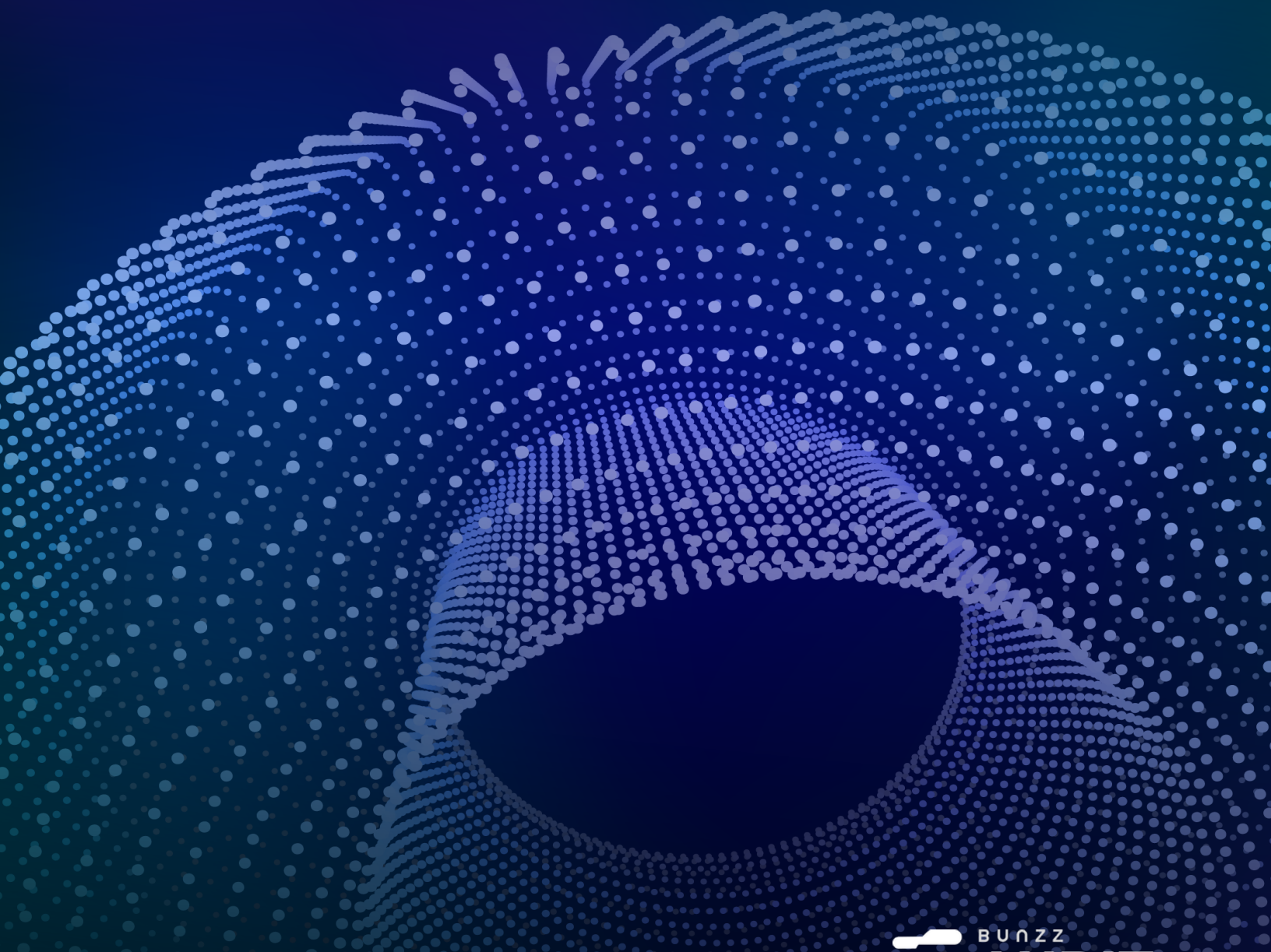
Security Assessment

Xenea Wallet

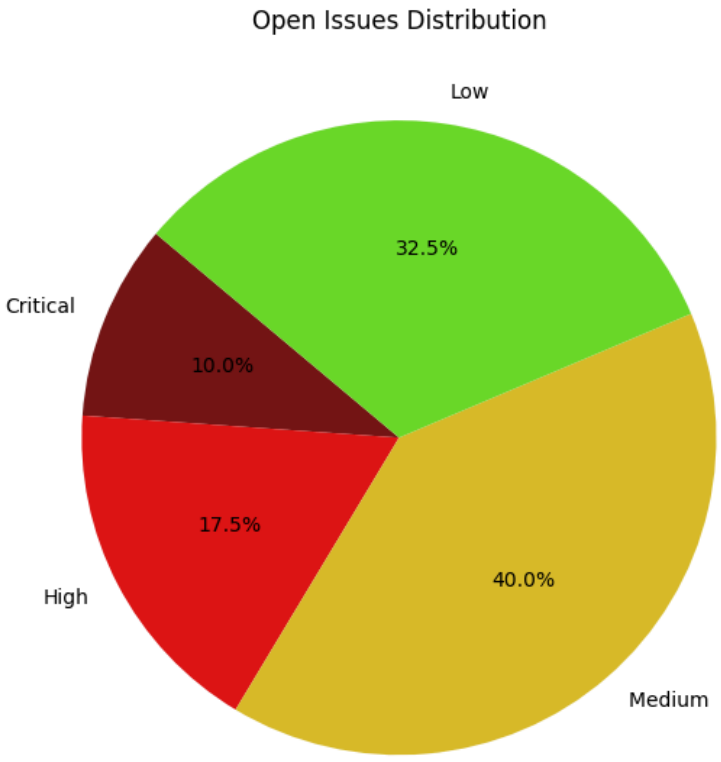
2025-April-2nd



BUNZZ AUDIT

Table of Contents

	Critical	High	Medium	Low
Open Issues	0	0	0	0
Acknowledged Issues	1	1	4	7
Resolved Issues	**3**	**6**	**12**	**6**
Total Issues	4	7	16	13



1 Executive Summary

A comprehensive Grey Box penetration test was conducted on the Xenea Wallet application and its admin endpoint to assess its security posture and identify potential vulnerabilities that could impact user data, financial transactions, and overall system integrity. Xenea Wallet is a blockchain-based digital wallet designed for seamless financial transactions and rewards, offering an intuitive interface and social login support. However, multiple security vulnerabilities were discovered during the assessment, which could expose sensitive user information and make the platform susceptible to various cyber threats.

The evaluation uncovered critical security flaws, including SSL pinning bypass via root-level certificate installation and ReFlutter, allowing attackers to intercept encrypted communications. Improper implementation of root and emulator detection mechanisms made the application susceptible to unauthorized modifications. IDOR vulnerabilities led to account ID and NFT ticket data exposure, as well as unauthorized access to the admin API, increasing the risk of privilege escalation. The lack of anti-hooking mechanisms left the application vulnerable to runtime manipulation, while the Android export flag was enabled, increasing the risk of unauthorized component access. Additionally, the application could be installed on an unpatched Android version, making it susceptible to known exploits. Further issues included insecure storage of sensitive data in local databases, Android memory, and application logs. Weak encryption algorithms were used, making sensitive information easier to decrypt. Improper input validation was observed, leading to potential SQL injection and JWT injection vulnerabilities. The exposure of public JWKS keys increased the risk of cryptographic attacks. The application also suffered from missing authentication on sensitive endpoints, insecure deep link configurations, and permission overload, granting unnecessary and dangerous access to user data. Unprotected SQLite databases made stored data vulnerable to tampering, and insecure random number generation posed a risk for cryptographic weaknesses. Additionally, session management vulnerabilities were present, including concurrent login allowance, session tokens exposed in URLs, and active sessions after logout, which could lead to unauthorized access. Cross-Origin Resource Sharing (CORS) misconfigurations made the application prone to cross-site request forgery (CSRF) and data leakage. The application logged sensitive information, created temporary files, and lacked proper security headers and cookie attributes, exposing it to session hijacking and cross-site scripting (XSS) attacks. The debugging bypass vulnerability allowed unauthorized access to internal application functions, increasing the attack surface.

To address these issues, stronger authentication and authorization mechanisms must be enforced, including proper session handling, rate-limiting, and secure API communication. The use of secure encryption standards like AES-GCM and SHA-256 should replace weak cryptographic implementations. Input validation, output encoding, and strict access control mechanisms should be implemented to mitigate injection attacks. The enforcement of security headers, improved logging practices, and application integrity verification will further strengthen the security posture of Xenea Wallet.

Overall, while Xenea Wallet offers a user-friendly digital payment solution, the assessment reveals significant security risks that need immediate remediation to prevent data breaches, financial fraud, and unauthorized system access. Addressing these vulnerabilities will enhance the platform's security and ensure user trust.

2 Scope

The section defines the scope and boundaries of the project.

2.1 Target Scope

Identify weaknesses that might be exploited by adversaries who have authorized or unauthorized access to Xenea host’s Technical Skill Set and underlying infrastructure:

- Test access credentials were provided. It was a Grey-Box Testing.
- The objective is to mimic an adversary and identify the threats and vulnerabilities.

Following application was in the scope of the penetration test. Automated as well as manual security testing was conducted.

Sr. No	Application Name	Test Type
1	Xenea Wallet	Grey-Box / Security Audit

2.2 Application and Environment Details

Application Name	URL
Xenea Wallet	xenea-wallet.apk
	https://mobile-api.staging.xenea.app/wallet/v1/
Admin endpoint	http://api-stg.xenea.network/api/v1/

2.3 Test Duration

Start Date	26th February 2025
End Date	21th March 2025

3 Confidentiality Statement

This document is the exclusive property of Xena wallet. This document contains proprietary and confidential information.

4 Contact Information

Name	Title	Contact Information
Piyush shukla	Linkedin Profile	Email: piyushshuka599@gmail.com

5 Detailed Findings

5.1 SSL Pinning Bypass via Root-Level Certificate Installation and ReFlutter

Vulnerability Severity	**OWASP Category**
High [redacted]	Insecure Communication
Tools Used	**Date of reporting**
Burp Suite Certificate, ReFlutter, and Burp Suite Proxy.	11th March 2025
Vulnerability Observation	
During testing, it was observed that the application implements SSL pinning; however, it is not properly enforced. As a result, SSL pinning can be bypassed by installing a custom certificate at the system level.	

Additionally, ReFlutter can be utilized to repackage and modify the APK file, allowing an attacker to intercept and analyze network traffic. This process involves extracting the application, configuring Burp Suite as a proxy, and enabling invisible proxying to facilitate traffic interception and analysis.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk, affecting the application's network communication security.

Technical Impact

The improper enforcement of SSL pinning allows attackers to bypass encryption mechanisms, intercept sensitive data in transit, modify application requests and responses, and perform man-in-the-middle (MITM) attacks, compromising the application's security and user data integrity.

Business impact

An attacker can intercept and manipulate sensitive network traffic, potentially leading to data theft, session hijacking, or man-in-the-middle (MITM) attacks.

Remediation

Implement strict SSL pinning by validating the server certificate within the application code, restricting user-installed CA certificates, and using frameworks like Android's Network Security Configuration or TrustKit to prevent bypass techniques.

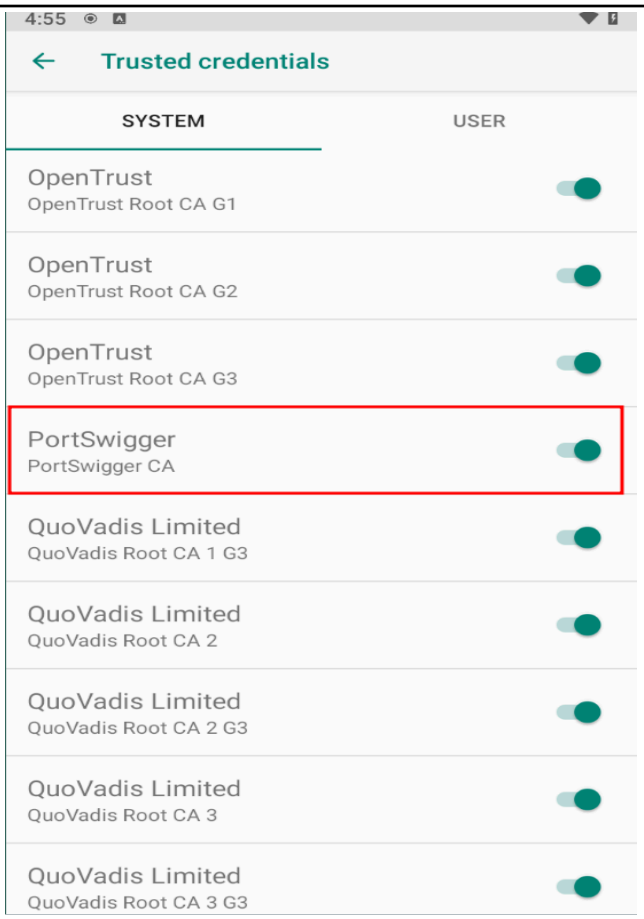
Status - Fixed

References

- Android Developers - Network Security Configuration
- OWASP Mobile Security Testing Guide (MSTG) - SSL Pinning
- TrustKit - SSL Pinning for Mobile Apps
- Google Security Best Practices

Steps to Reproduce

1. Install the Burp Suite certificate at the Android system level using the following method.



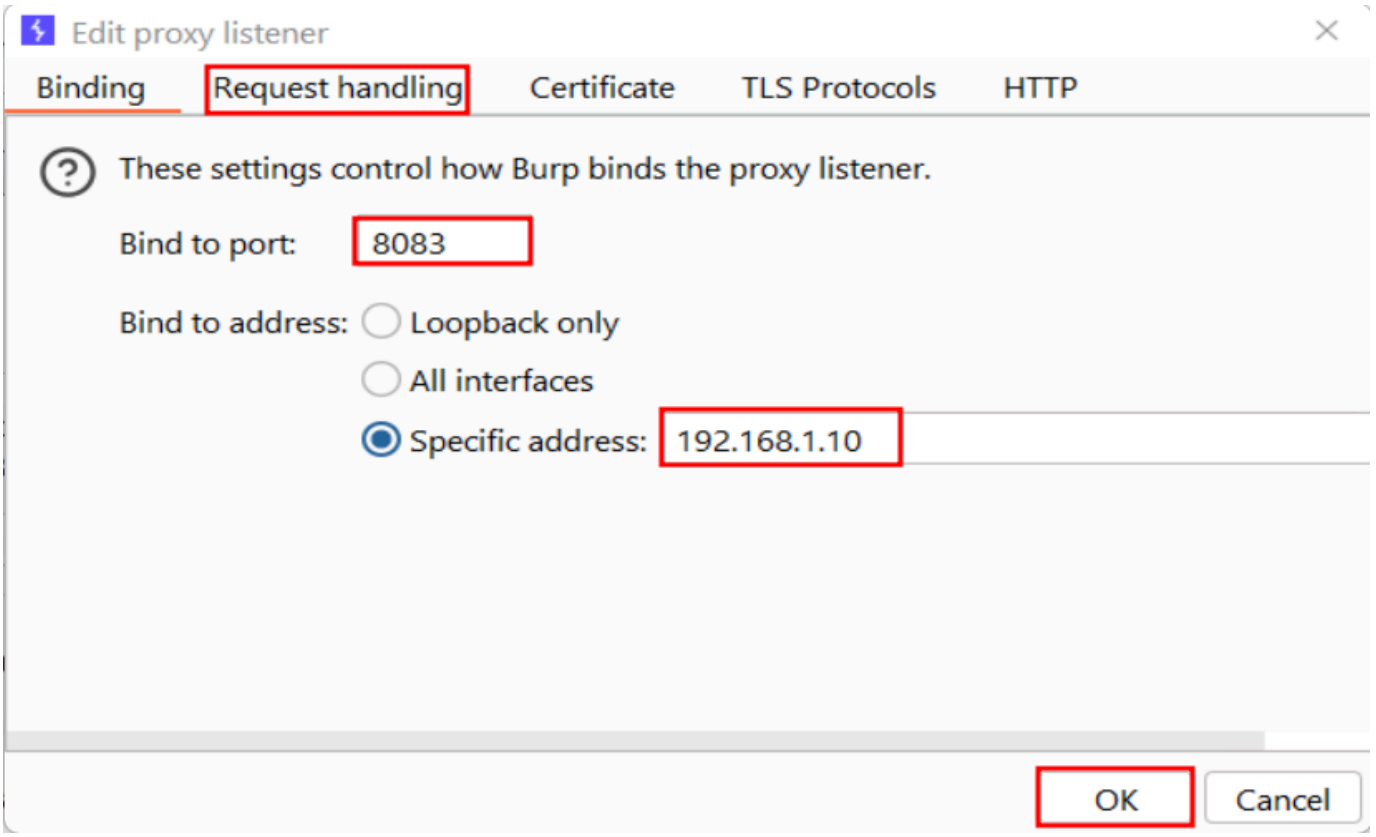
2. Now, run ReFlutter against xenera-wallet.apk, enter your Burp Suite/system IP, and press Enter. This will generate a new APK named release.RE.apk.

```
>reflutter.exe xenea-wallet.apk
[*] Processing...

Example: (192.168.1.154) etc.
Please enter your BurpSuite IP: 192.168.1.10

SnapshotHash: 80a49c7111088100a233b2ae788e1f48
The resulting apk file: ./release.RE.apk
Please sign, align & install the apk file

Configure Burp Suite proxy server to listen on *:8083
Proxy Tab -> Options -> Proxy Listeners -> Edit -> Binding Tab

Then enable invisible proxying in Request Handling Tab
Support Invisible Proxying -> true
```

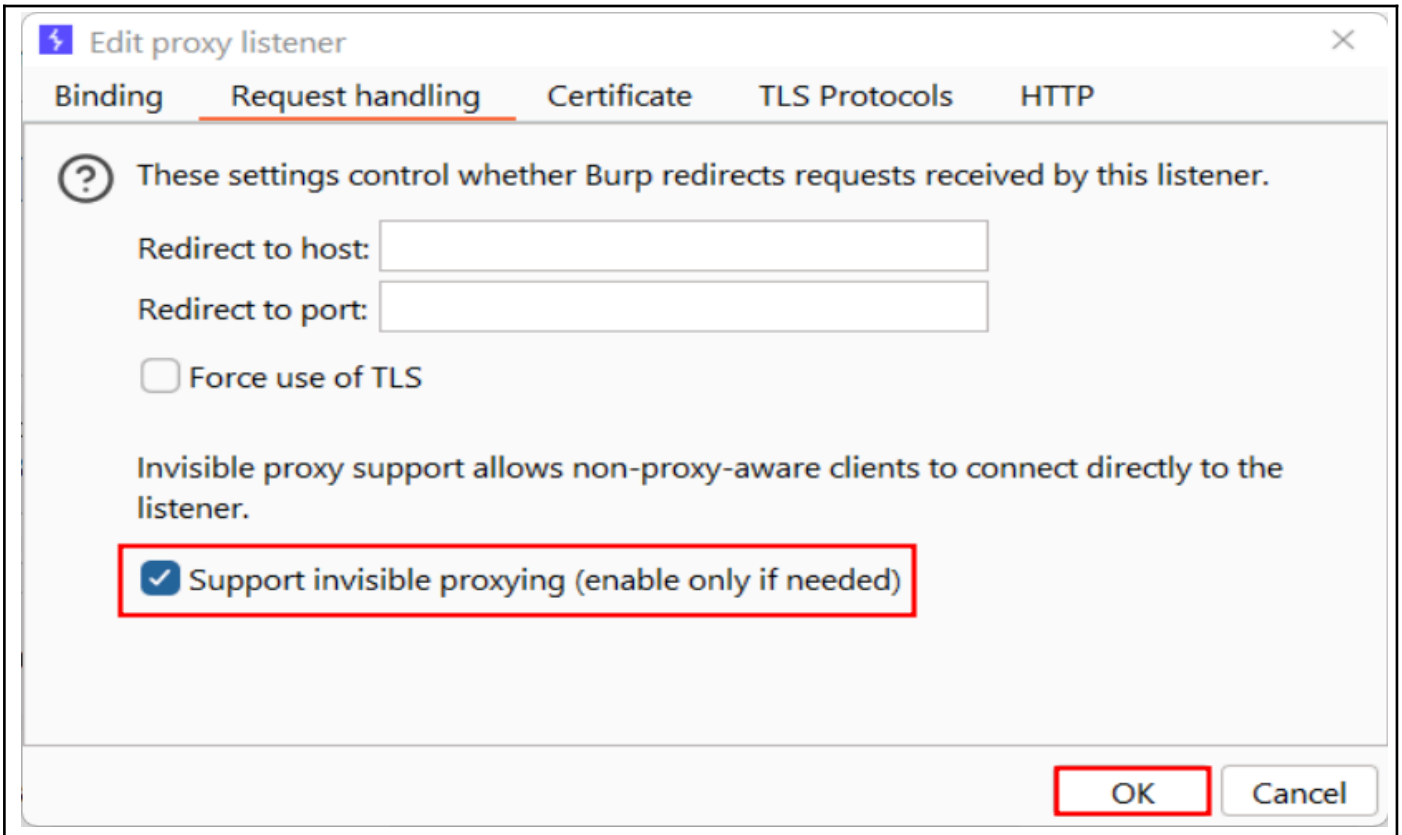
3. Then, use uber APK Signer on release.RE.apk to sign it. Once the process is complete, it will generate a new APK named release.RE-aligned-debugSigned.apk and install it in the android device.

```
PS C:\Users\sampr\Desktop\eMAPT\Tools> java -jar .\uber-apk-signer-1.3.0.jar -allowResign -a release.RE.apk
source:
        C:\Users\sampr\Desktop\eMAPT\Tools
binary-lib/windows-33_0_2/libwinpthread-1.dll
C:\Users\sampr\AppData\Local\Temp\uapksigner-12178213796176269485
zipalign location: BUILT_IN
        C:\Users\sampr\AppData\Local\Temp\uapksigner-12178213796176269485\win-zipalign_33_0_2.exe18108851466027304954.tmp
keystore:
        [0] 161a0018 C:\Users\sampr\AppData\Local\Temp\temp_11323517342105618210_debug.keystore (DEBUG_EMBEDDED)

01. release.RE.apk

        SIGN
        file: C:\Users\sampr\Desktop\eMAPT\Tools\release.RE.apk (110.58 MiB)
        checksum: 2e008da4d2e855101014bf78c4d6b6c2fec5e7fab3d9c25e9a47a66bf1708ecd (sha256)
        - zipalign success
        - sign success

        VERIFY
        file: C:\Users\sampr\Desktop\eMAPT\Tools\release.RE-aligned-debugSigned.apk (110.69 MiB)
        checksum: b2065b9361f2557916098341a4a7ae712a0256fc4111dca082a064ea2c13c540 (sha256)
        - zipalign verified
        - signature verified [v2, v3]
                Subject: CN=Android Debug, OU=Android, O=US, L=US, ST=US, C=US
                SHA256: 1e08a903aef9c3a721510b64ec764d01d3d094eb954161b62544ea8f187b5953 / SHA256withRSA
                Expires: Fri Mar 11 01:40:05 IST 2044

[Wed Feb 26 20:34:41 IST 2025][v1.3.0]
Successfully processed 1 APKs and 0 errors in 2.02 seconds.
```

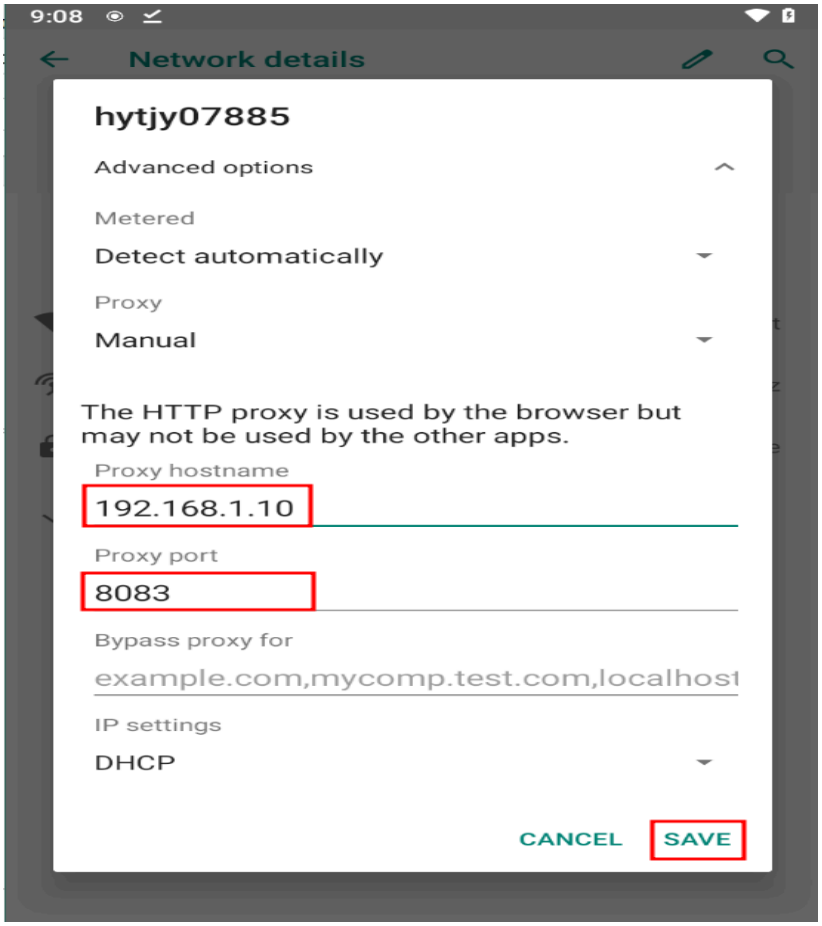
4. Now, set up the Burp Suite proxy with port number 8083 and select your system's IP address, then click on the Request handling tab.



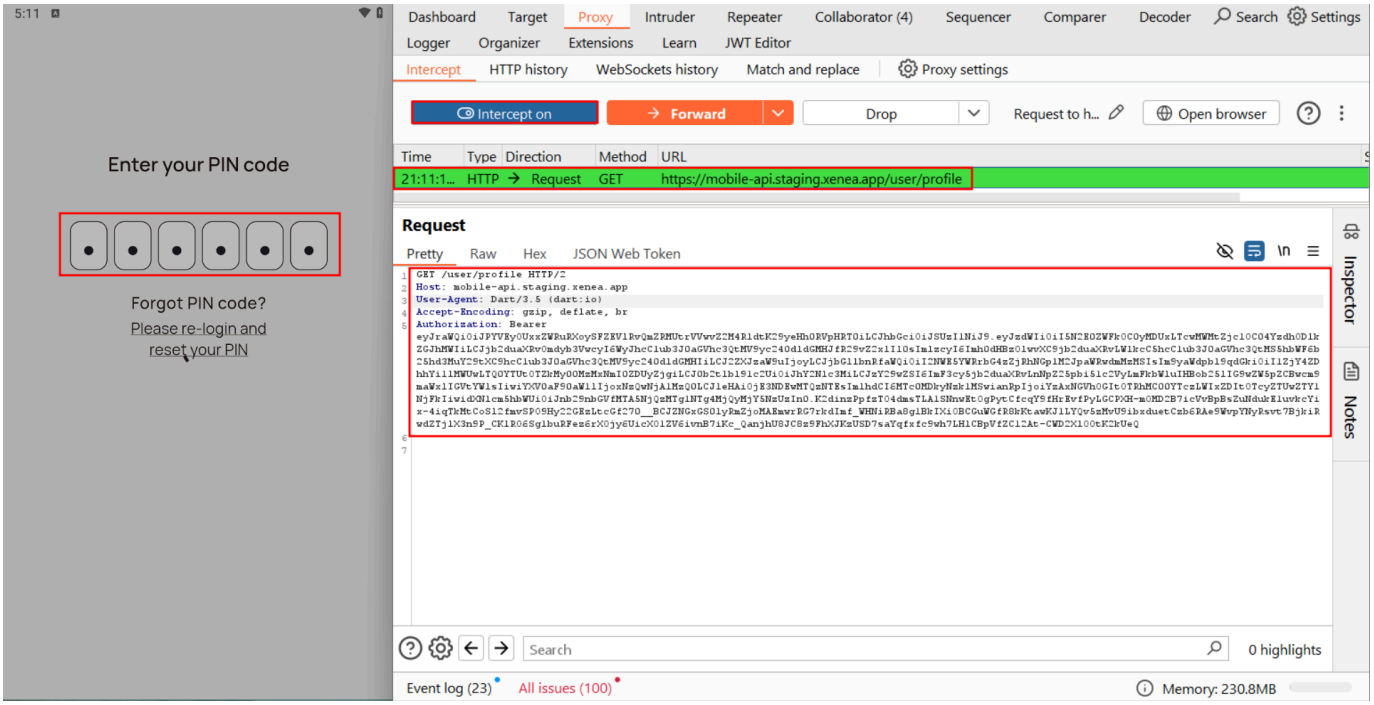
5. Select Support invisible proxying (enable only if needed) and click OK.



6. Also, configure the proxy on the Android device with the system IP and port **8083** and click on SAVE.



7. Launch the app, enter the PIN code, and intercept its request in the Burp Suite.



5.2 Improper Implementation of Root and Emulator Detection Mechanisms

Vulnerability Severity	**OWASP Category**
High [redacted]	Reverse Engineering
Tools Used	**Date of reporting**
Jadx-gui & adb	11th March 2025

Vulnerability Observation

During the decompilation of the APK, it was observed that the application implements a root detection mechanism; however, it is not properly enforced. Despite the presence of the su file in the /xbin/ directory, the application fails to detect that it is running on a rooted device or an emulator. A rooted device is highly vulnerable to malware and remote attacks, significantly increasing the risk of data theft and unauthorized access.

Vulnerable location

Throughout the application: xenea-wallet.apk

Technical Impact

A rooted device allows a user to install any applications which are not downloaded from the Google Play Store. This increases the risk of malicious applications getting installed in the device which can hijack the user credentials and can perform action on behalf of the user.

Business impact

Failure to enforce proper root detection increases the risk of data breaches, fraud, and reputational damage. Attackers can exploit this weakness to tamper with the application, extract confidential data, and bypass security controls, potentially leading to financial and compliance-related consequences.

Remediation

- Employ root and emulator detection techniques in the application. This can be done by checking for the presence of superuser.apk or running the "su" command. The presence of which indicates that the device is rooted and the application should terminate or should notify the user about the risk associated with using the application on a rooted device.
- Root detection can be implemented by adding the following Java checks:-

  CheckRootManagementApps

  CheckPotentially Dangerous Apps

  CheckRootCloakingApps

  CheckTestKeys

  checkForDangerousProps

  checkForBusyBoxBinary

  checkForSuBinary

  checkSuExists

  checkForRWSystem

  Native checks

- More information about different root detection techniques:

  https://github.com/scottyab/rootbee

Status - Fixed

References

1. OWASP Mobile Security Testing Guide (MSTG)
2. Android Developer Guide – SafetyNet Attestation API
3. Google Play Security Recommendations
4. Android Security Best Practices – Detecting Rooted Devices

Steps to Reproduce

1. Open the xenea-wallet.apk in JADX-GUI and analyze the decompiled code.

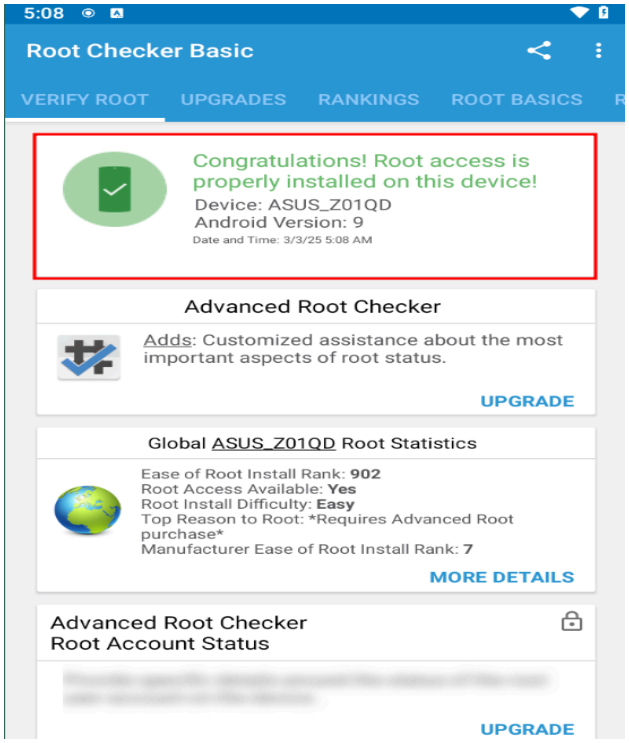
AbstractC2032i

```
            return false;
        }
        return true;
    }

    public static boolean w() {
        if (!Build.PRODUCT.contains("sdk")) {
            String str = Build.HARDWARE;
            if (!str.contains("goldfish") && !str.contains("ranchu")) {
                return false;
            }
        }
        return true;
    }

    public static boolean x() {
        boolean w5 = w();
        String str = Build.TAGS;
        if ((!w5 && str != null && str.contains("test-keys")) || new File("/system/app/Superuser.apk").exists()) {
            return true;
        }
        File file = new File("/system/xbin/su");
        if (!w5 && file.exists()) {
            return true;
        }
        return false;
    }

    public static boolean y(String str, String str2) {
        if (str == null) {
            if (str2 == null) {
                return true;
            }
            return false;
        }
        return str.equals(str2);
    }

    public static String z(String str) {
        return r(str, "SHA-1");
    }
}
```

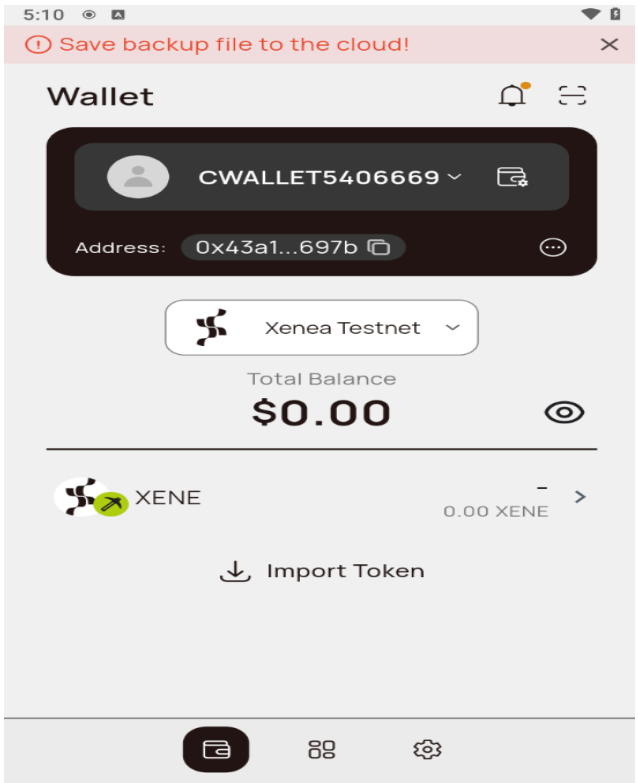
2. Navigate to the specified file path and check that "su" is present in the directory.

```
C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools>adb -s 127.0.0.1:21533 shell
ASUS_Z01QD:/ # ls /system/app/ && ls /system/xbin/
BluetoothMidiService CarrierDefaultApp      ExtShared                GoogleTTS            NfcNci                    SimAppDialog
BookmarkProvider     CertInstaller          Gallery2                 HTMLViewer           PacProcessor              Traceur
BuiltInPrintService  Chrome                 GoogleCalendarSyncAdapter IME                 PlayGames.apk             WAPPushManager
CMFileManager        CompanionDeviceManager GoogleContactsSyncAdapter KeyChain           PrintRecommendationService WallpaperBackup
CaptivePortalLogin   CtsShimPrebuilt        GoogleExtShared          LiveWallpapersPicker SecureElement
su tcpdump
ASUS_Z01QD:/ #
```

3. Use Root Checker to verify if the device is rooted.



4. Launch the application and observe that it runs successfully on a rooted and enumerated device without any restrictions.



5.3 IDOR Leading to Account ID and NFT Ticket data Exposure.

Vulnerability Severity	OWASP Category
High [redacted]	IDOR Leading to Account ID Disclosure
Tools Used	**Date of reporting**
Burp suite	11th March 2025

Vulnerability Observation

During testing, we identified a vulnerability that allows unauthorized access to sensitive user information. An attacker can exploit this issue to retrieve account IDs, NFT ticket details, and NFT data of any user by using their wallet address.

Vulnerable location

Resources Affected:

https://mobile-api.staging.xenea.app/wallet/v1/transactions

https://mobile-api.staging.xenea.app/wallet/v1/nfts/tickets

https://mobile-api.staging.xenea.app/wallet/v1/nfts

Parameters Affected: walletAddress & address

Technical Impact

This vulnerability exposes sensitive user information due to improper access control, allowing an attacker to enumerate and retrieve account-related details. Exploiting this issue could lead to unauthorized access to user data, identity theft, and further exploitation of associated NFT assets.

Business impact

Unauthorized access to user account details and NFT-related data can result in financial losses, reputational damage, and legal compliance violations. If exploited, this issue could undermine user trust, impact the platform's credibility, and lead to regulatory penalties or security breaches.

Remediation

As we are able to access the wallet details of any user, which is intended for this application, but the app should not disclose the account ID in the response. Therefore, we recommend removing it to enhance security.

Note: Since our account does not contain NFT-related data, we were unable to verify the exposure fully; however, we recommend checking with the developer to confirm the issue in NFTs.

Status - Fixed

References

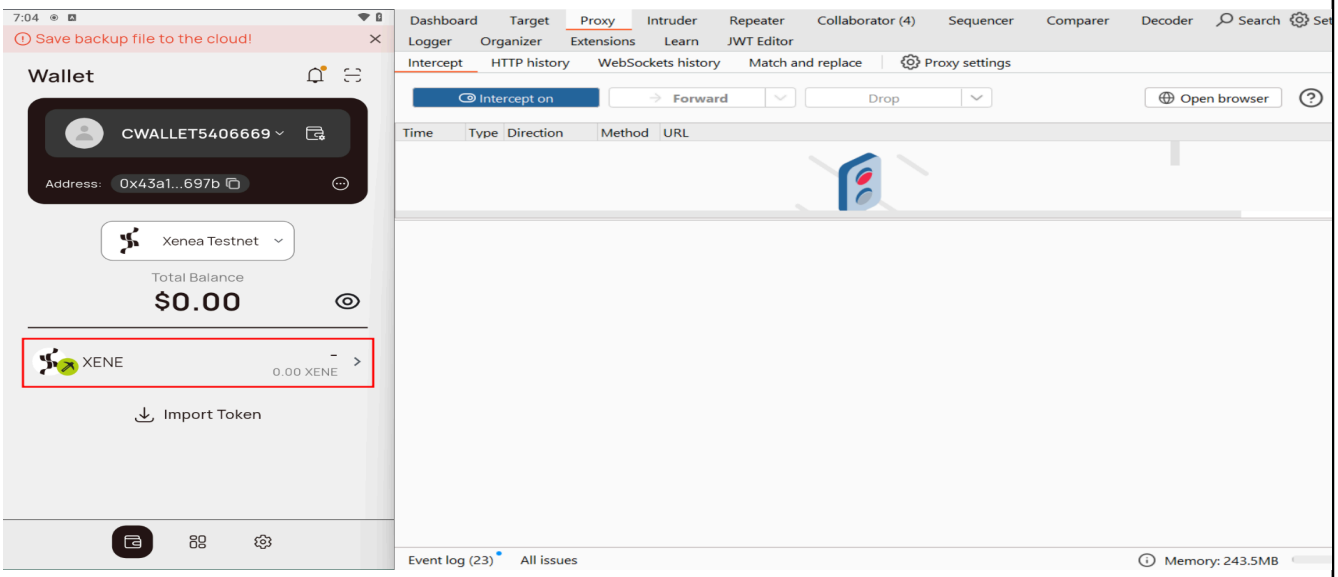
1. https://owasp.org/www-project-api-security/
2. https://owasp.org/www-community/Improper_Access_Control

3. https://portswigger.net/web-security/access-control
4. https://owasp.org/www-project-top-ten/

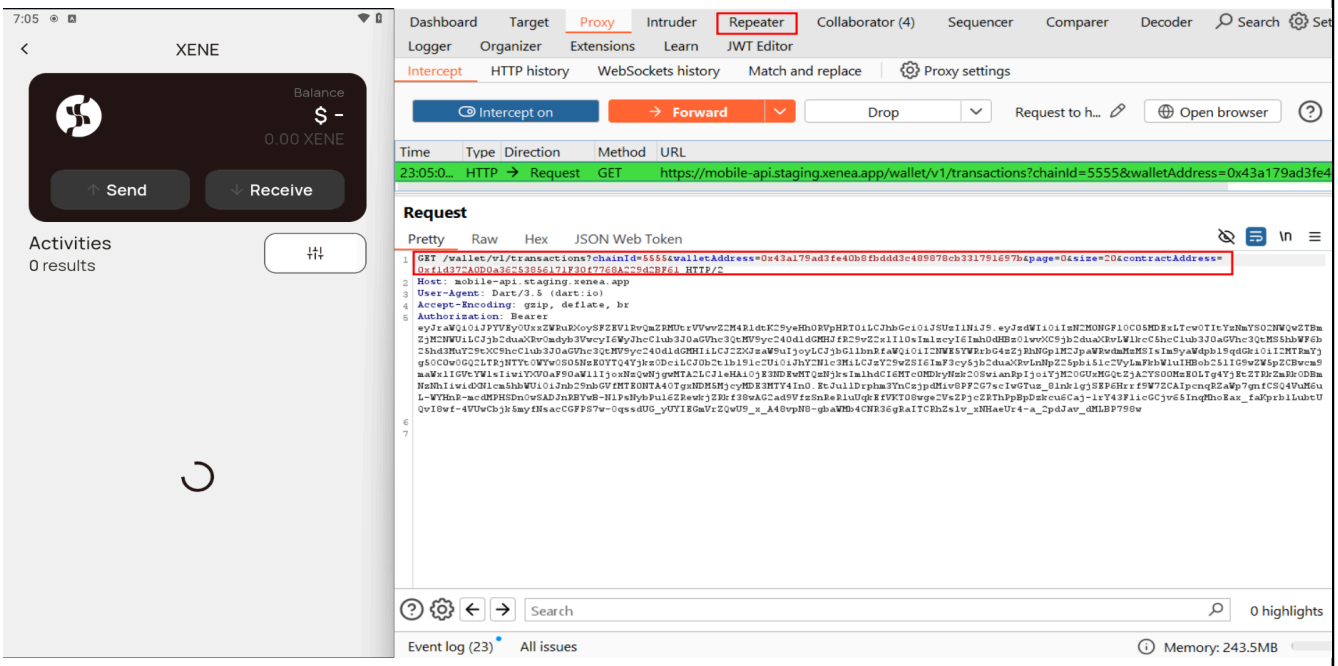
Steps to Reproduce

Case 1:

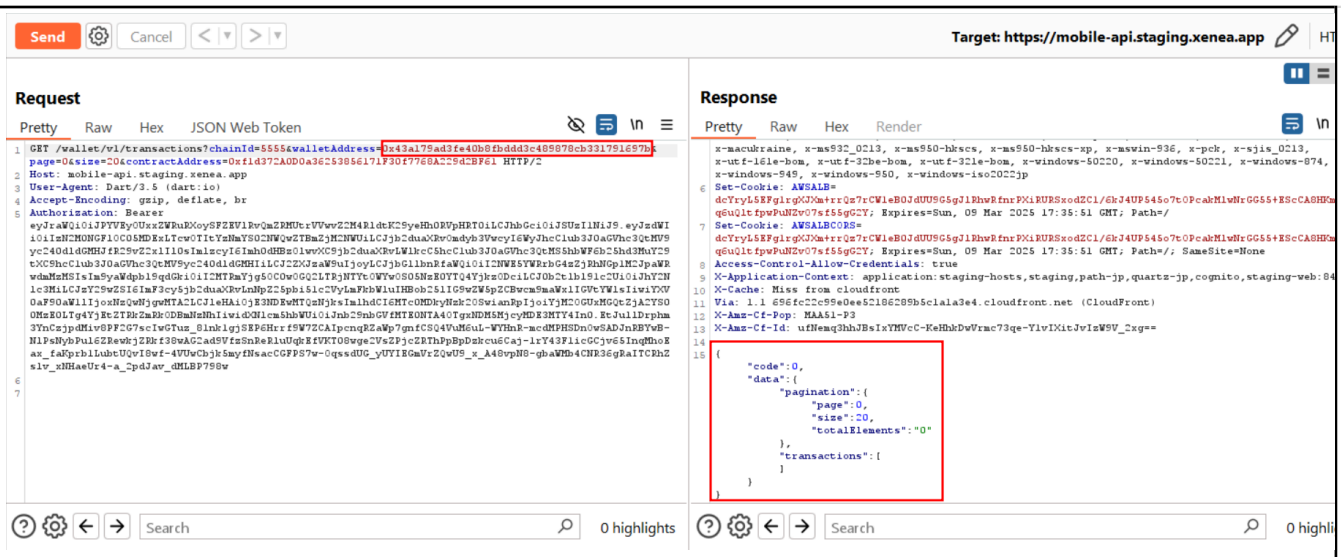
1. Open the Android application and navigate to the wallet transaction history section.



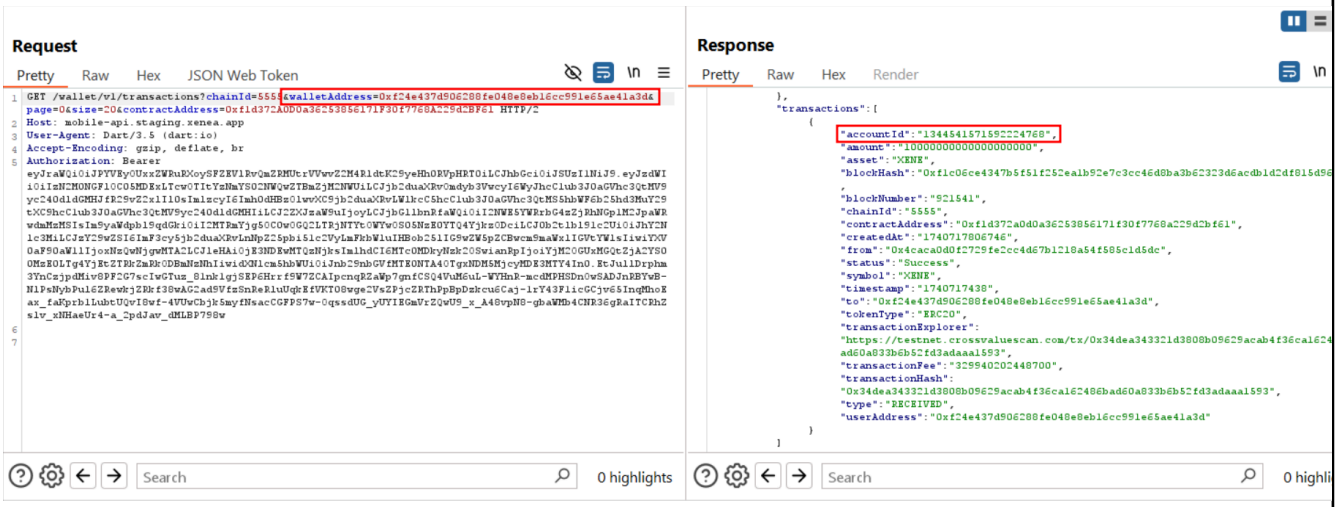
2. Capture the API request that retrieves transaction details using Burp Suite and send it to the Repeater.



3. Send the request and observe the response, noting that no transaction data is returned as this account ID has no transaction history.



4. Modify the walletAddress parameter to another user's wallet address: 0xb54211e3f84cc186f2f87ca06587910787cf841e and resend the request. Observe that the response now reveals sensitive details such as the user's accountID, userAddress, and other information.



Case 2:

1. Modify the `Address` parameter to User 1's wallet address and resend the request. Observe that the application accepts the modified `Address` and returns the response.

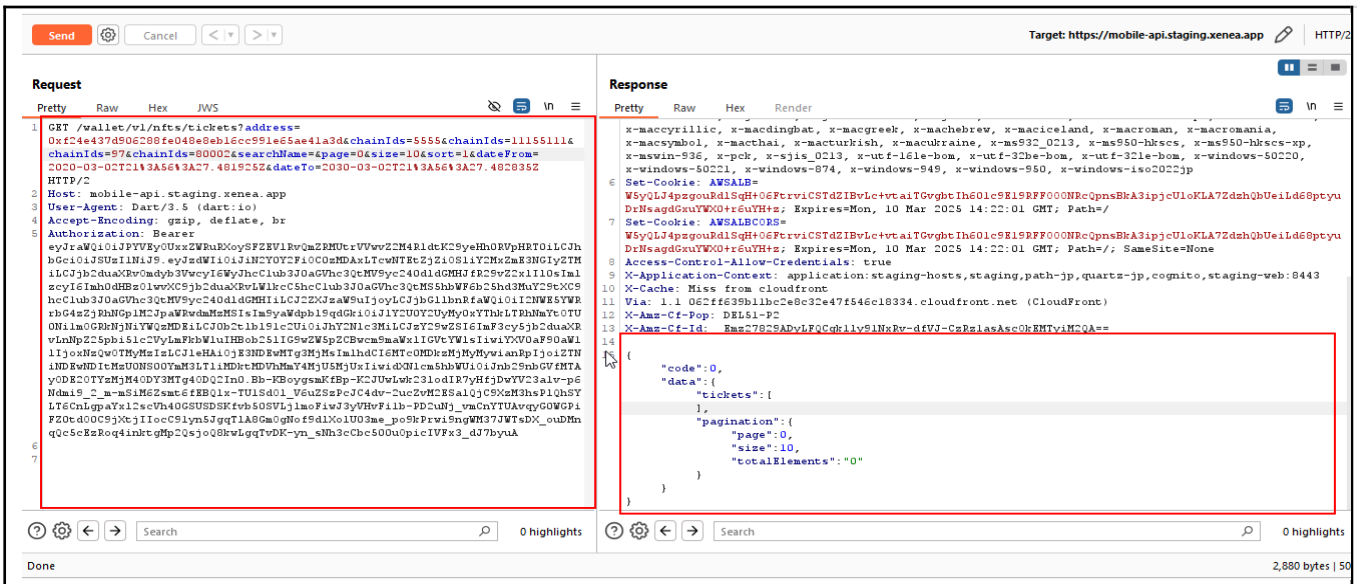
   Note:

   user 1 Details

   wallet id ⇒ 0xf24e437d906288fe048e8eb16cc991e65ae41a3d

   user 2 Details

   wallet id ⇒ 0xb54211e3f84cc186f2f87ca06587910787cf841e



Case 3:

1. Modify the `Address` parameter to User 1's wallet address and resend the request. Observe that the application accepts the modified `Address` and returns the response.

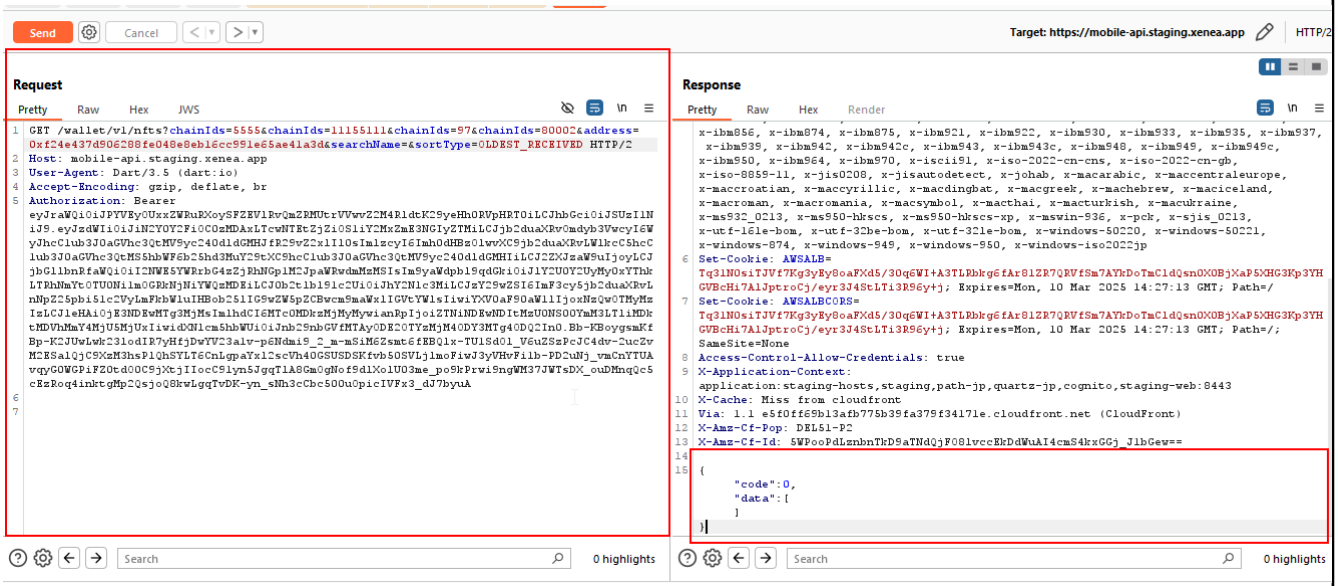
   Note:

   user 1 Details

   wallet id ⇒ 0xf24e437d906288fe048e8eb16cc991e65ae41a3d

   user 2 Details

   wallet id ⇒ 0xb54211e3f84cc186f2f87ca06587910787cf841e



5.4 Unauthorized Access to Admin Api

Vulnerability Severity	OWASP Category
Critical [redacted]	Broken Access Control
Tools Used	**Date of reporting**
Burp Suite	11th March 2025

Vulnerability Observation

During testing, it was observed that the application allows unauthorized access to admin APIs, enabling non-admin users or unauthenticated attackers to perform administrative actions. This occurs due to improper authentication and authorization checks on admin endpoints. Non-admin User able to access userId, walletId, nodes of all users.

Vulnerable location

http://api-stg.xenea.network/api/v1/admin/wallets?page=1&size=100
http://api-stg.xenea.network/api/v1/admin/nodes?page=1&size=10

Technical Impact

Improper authentication and authorization checks on admin endpoints allow non-admin users or unauthenticated attackers to access sensitive administrative functionalities. This could lead to unauthorized access to critical user data, including user IDs, wallet IDs, and node details. Attackers could exploit this vulnerability to enumerate users, manipulate account information, or perform administrative actions without proper privileges.

Business impact

Unauthorized access to admin APIs can result in a severe data breach, exposing sensitive user information and compromising the integrity of the platform. This could lead to regulatory non-compliance, legal repercussions, financial losses, and reputational damage. Additionally, if attackers exploit this flaw, they could gain control over user accounts, leading to potential fraud, identity theft, or further security compromises.

Remediation

- Implement Proper Access Control to restrict access to user and wallet data.
- Use Indirect Identifiers instead of sequential or predictable values.
- Validate and Enforce Authorization Checks on API endpoints.
- Mask or Encrypt Sensitive Identifiers before exposing them in responses.
- Limit Data Exposure in API Responses to only what is necessary .

Status - Fixed

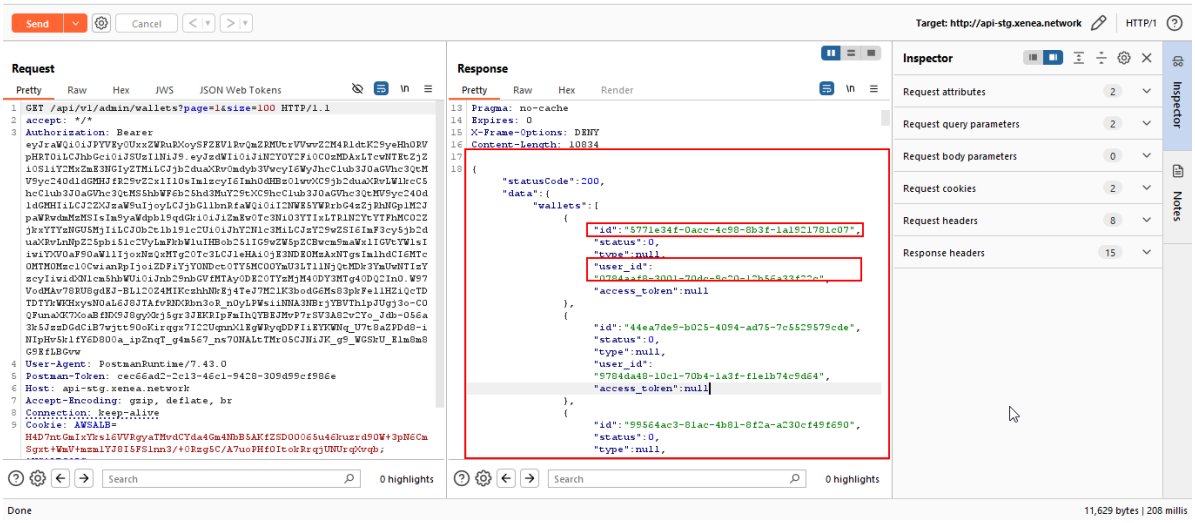
References

- Insecure Authentication/Authorization
- Identification and Authentication Failures
- OWASP Authorization

Steps to Reproduce

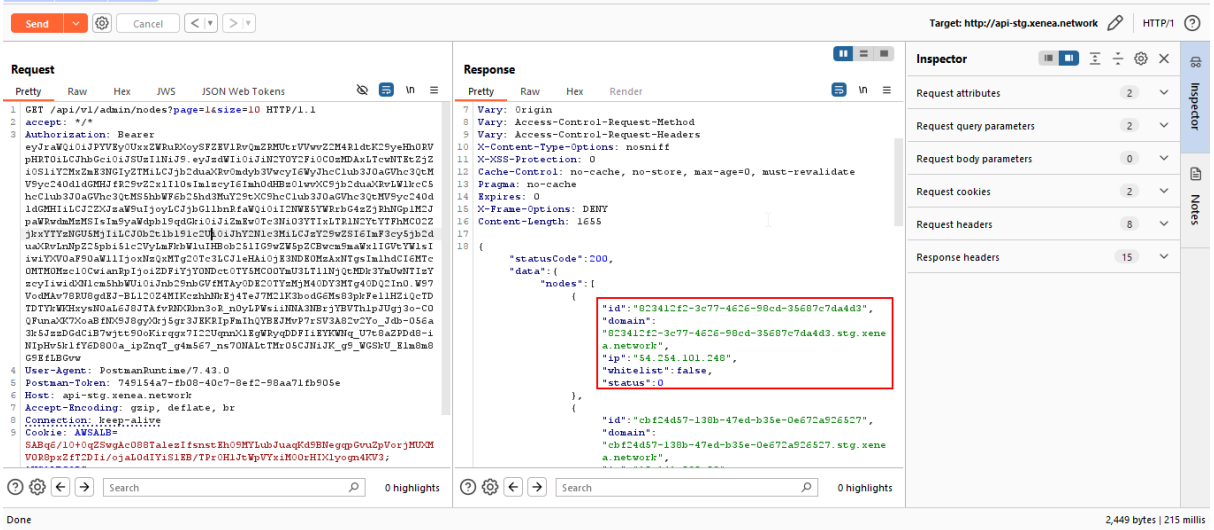
Case 1:

1. Using a normal user's cookies from the mobile wallet, we were able to access the admin /wallets API, which can lead to the exposure of wallet IDs and user IDs.



Case 2:

1. Again using a normal user's cookies from the mobile wallet, we were able to access the admin /nodes API, which can lead to the exposure of wallet IDs and the user ip's.



5.5 Unauthorized IDOR leads to Information Disclosure

Vulnerability Severity	**OWASP Category**
Critical	Broken Access Control
Tools Used	**Date of reporting**
Burp Suite	11th March 2025
Vulnerability Observation	

During testing, it was observed that the application lacks proper authorization checks on admin APIs, allowing non-admin users to retrieve sensitive details such as user ID, wallet ID, nodes, IP address, access token, and username. Additionally, non-admin users can modify the whitelist settings for any user's node, enabling unauthorized changes to security controls like IP addresses and devices. This vulnerability leads to Insecure Direct Object References (IDOR), allowing attackers to bypass access restrictions, modify security settings, and potentially escalate privileges or disrupt services.

Vulnerable location

http://api-stg.xenea.network/api/v1/admin/wallet?walletId=5771e34f-0acc-4c98-8b3f-1a1921781c07
http://api-stg.xenea.network/api/v1/admin/node?nodeId=823412f2-3c77-4626-98cd-35687c7da4d3
http://api-stg.xenea.network/api/v1/wallet/pre-recovery
http://api-stg.xenea.network/api/v1/nodes/f7177a68-a0ac-4302-935b-95177a1ff005

Parameter: WalletId, checksum, nodeId & whitelist

Technical Impact

User Privacy Violation: Unauthorized users can retrieve or manipulate personal data.
ID Enumeration Attack: Sequential or predictable user IDs enable mass data scraping.
Privilege Escalation: Attackers can access admin or other restricted accounts by modifying requests.

Business impact

Regulatory Violations (e.g., GDPR, HIPAA) due to improper data protection.
Loss of User Trust if sensitive data is exposed or misused.
Potential Financial and Legal Consequences if attackers exploit user data for fraud.

Remediation

- Enforce Proper Authorization Checks on all user data endpoints.
- Use Role-Based Access Control (RBAC) to restrict user permissions.
- Implement Object-Level Security by verifying ownership before returning or modifying data.
- Avoid Exposing User Identifiers in URLs or Responses unless necessary.
- Use UUIDs Instead of Sequential IDs to prevent enumeration attacks.
- Validate Requests Server-Side instead of relying on client-side controls.
- Monitor Logs and Apply Rate Limiting to detect and prevent mass data access attempts.

Status - Fixed

References

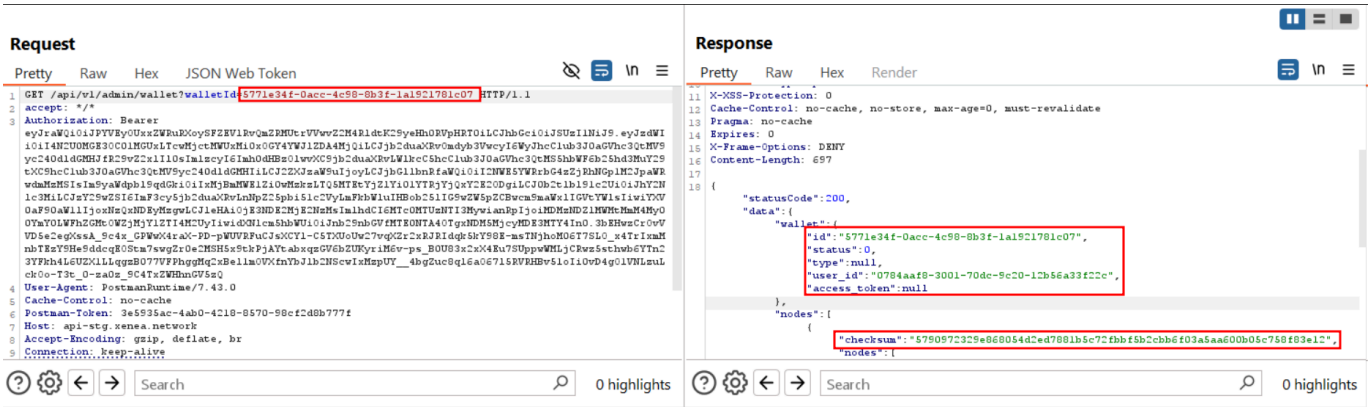
- c7-enforce-access-controls
- www-project-application-security-verification-standard
- final
- access-control

Steps to Reproduce

Case 1:

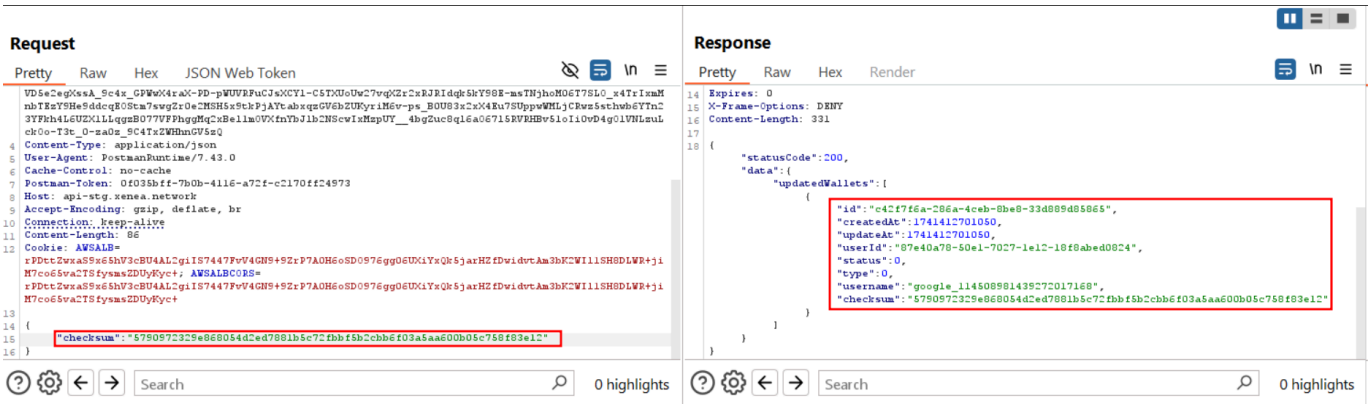
1. Using the obtained wallet IDs of all users, modify the walletId parameter to another user's wallet ID. Send the request and observe the response, specifically checking the checksum value

returned.



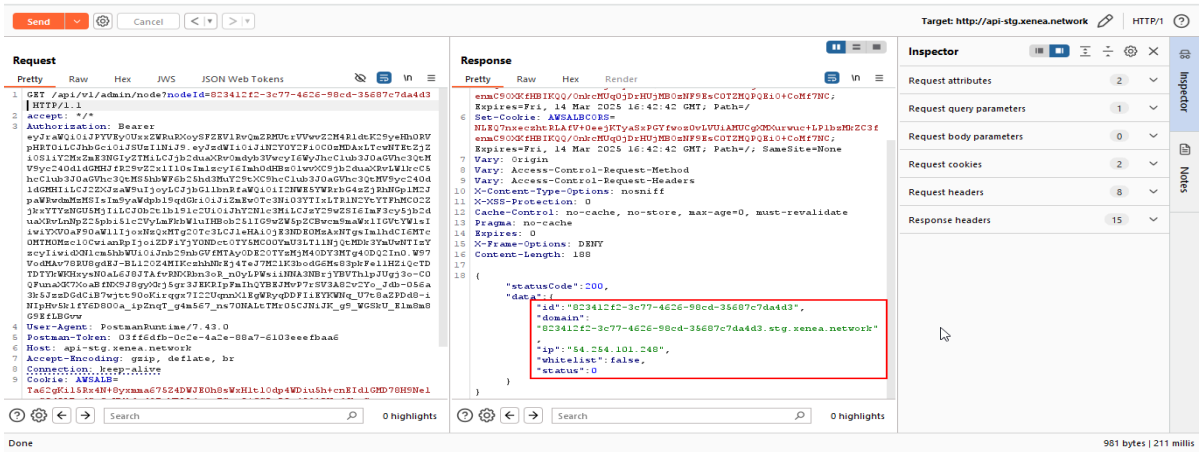
Case 2:

1. Chaining the above case, we can now use any `checksum` value obtained from the response. Modify the `checksum` parameter in the API request by replacing it with another user's `checksum` value and send the request. Observe the response to verify unauthorized access.



Case 3:

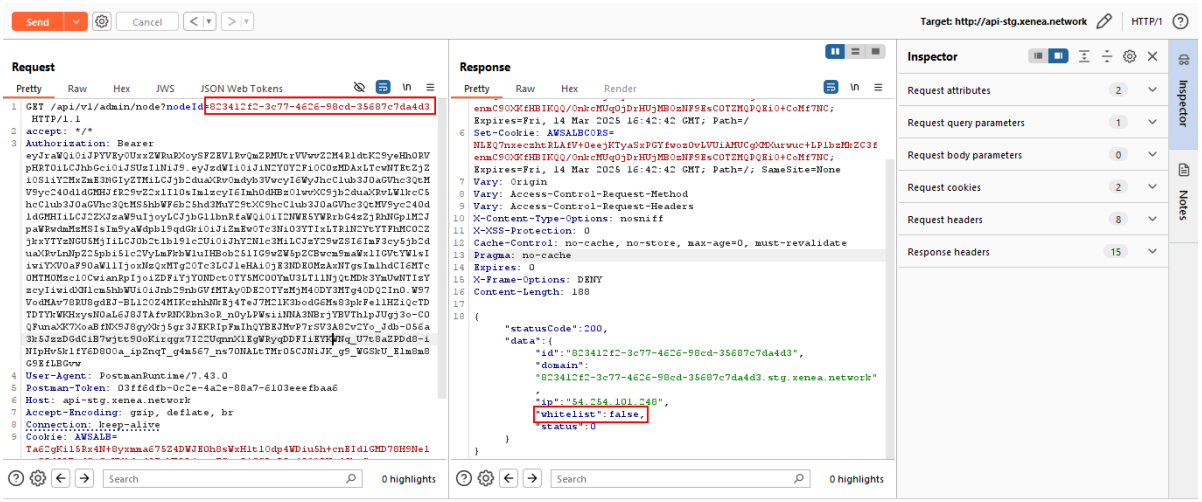
1. By changing nodeId to another user nodeId, sending requests and observing the response body.



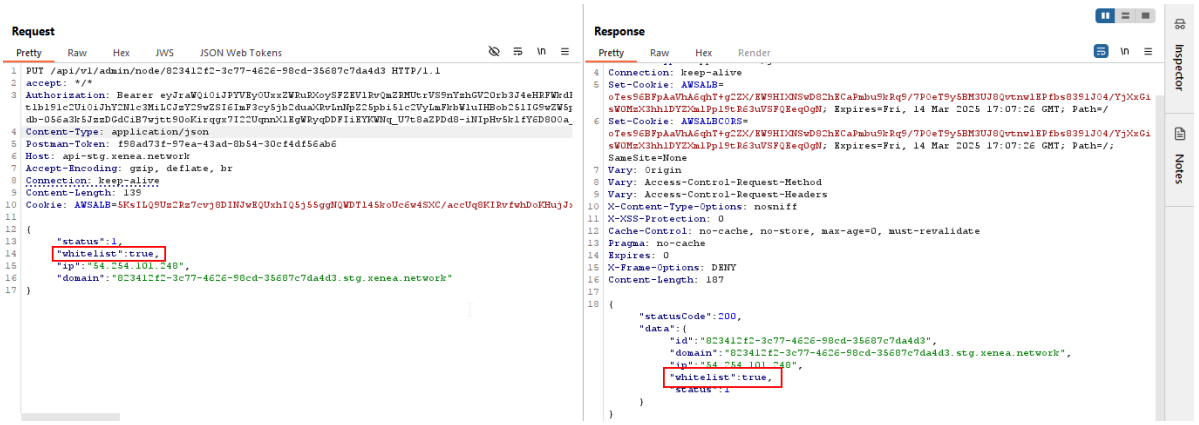
Case 4:

1. Modify the `nodeId` in the URL after `/nodes/`, forward the request, and observe the response

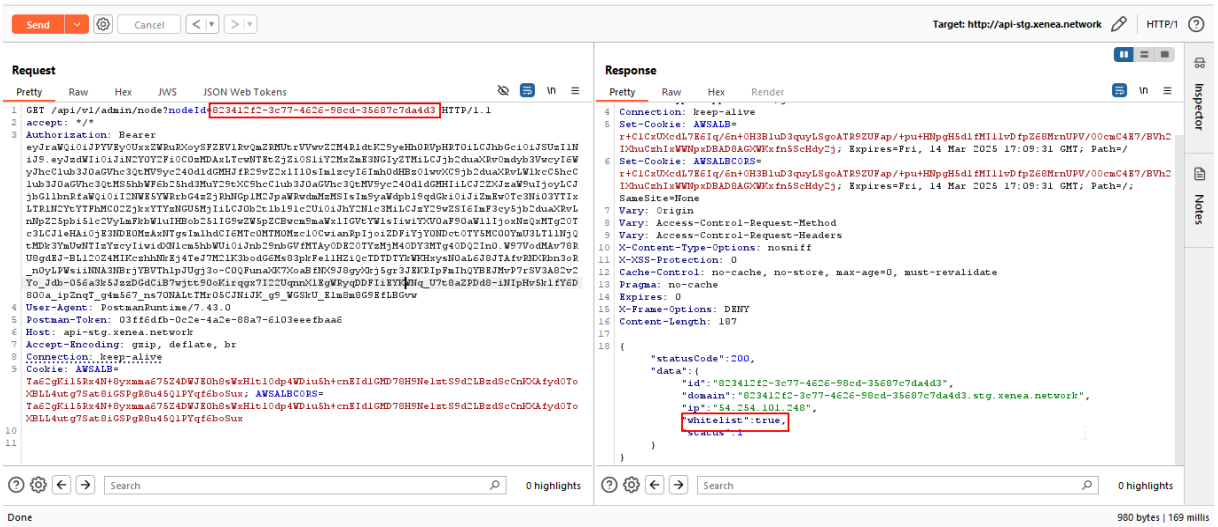
body. Verify that we can access any user's ID and check that the `whitelist` parameter is set to `false`.



2. Modify the `domain` and `IP` parameters to a related node `823412f2-3c77-4626-98cd-35687c7da4d3` in this request. Additionally, change the `whitelist` parameter to `true`.



3. Verify this by sending the first request in the Repeater tab and observe that the whitelisted status can be updated.



5.6 Anti hooking not implemented

Vulnerability Severity	OWASP Category
Medium	Insecure Communication
Tools Used	**Date of reporting**
frida	11th March 2025

Vulnerability Observation

It is observed that applications are not detecting widely used reverse engineering tools, such as code injection tools, hooking frameworks and debugging servers.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

If there is no anti-hooking or anti-debugging detection present in the application, that will make it easier for an attacker to debug the application on run time or hook any function on run time.

Business Impact

The absence of anti-hooking mechanisms allows attackers to intercept and modify the app's runtime behavior, potentially leading to unauthorized access, data manipulation, or bypassing security controls. This increases the risk of reverse engineering, credential theft, and exploitation of sensitive functionalities.

Remediation

- Detecting using default configuration of Hooking-framework: An obvious way to detect anti hooking frameworks is to check the environment for related artifacts, such as package files, binaries, libraries, processes, and temporary files. As an example, I'll hone in on hooking-server, the daemon responsible for exposing hooking over TCP. You can use a Java method that iterates through the list of running processes to determine whether a hooking-server is running. - Also check for ports with which hooking-server's are bind to TCP.
- Some Hooking-server uses the D-Bus protocol to communicate i.e. frida-server, so developers can send a D-Bus AUTH message to every open port and check for an answer, hoping that frida-server will reveal itself.
- The common theme for all Frida's modes is code injection, so we can expect to have Frida libraries mapped into memory whenever Frida is used. The straightforward way to detect these libraries is to walk through the list of loaded libraries and check for suspicious ones.

Status - Fixed

References

1. Testing-Resiliency-Against-Reverse-Engineering
2. Anti-reverse-engineering-protection-techniques-to-use-before-releasing-software

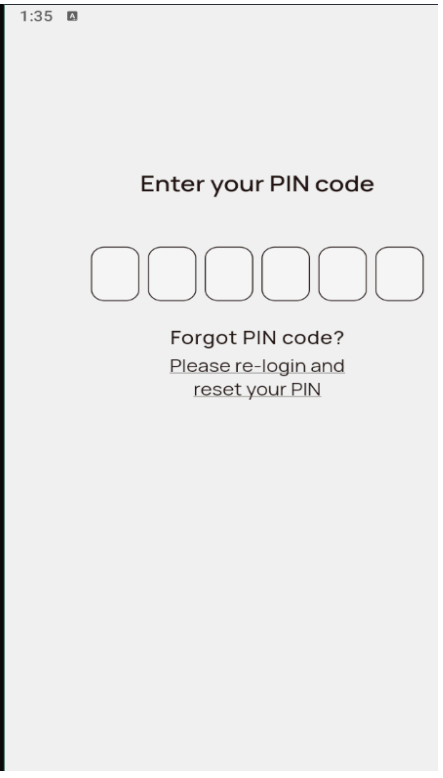
Steps to Reproduce

1. Connect to the rooted android device using USB and start the Frida server. Run the command: frida-ps -U and observe the application PID.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\sampr> frida-ps -U
  PID  Name
----  ------------------------------------------------
1302  Chrome
1846  Google Play Games
1313  Google Play Store
3489  MEmu Launcher2
3526  XENEA Wallet Staging
 178  adbd
 776  android.ext.services
 159  android.hardware.audio@2.0-service
 160  android.hardware.camera.provider@2.4-service
 161  android.hardware.cas@1.0-service
 162  android.hardware.configstore@1.1-service
 163  android.hardware.dumpstate@1.0-service
 164  android.hardware.gnss@1.0-service
```

2. Run command: frida-trace -U -f com.xenea.wallet.stg -i "open" and observe the output. Also, keep an eye on the android device, the application started automatically by Frida.

```
\Users\sampr> frida-trace -U -f com.xenea.wallet.stg -i "open"
umenting...
Loaded handler at "C:\\Users\\sampr\\__handlers__\\libc.so\\open.js"
ed tracing 1 function. Press Ctrl+C to stop.
           /* TID 0x1ec3 */
  ms  open()
  ms  open()
  ms  open()
  ms  open()
  ms  open()
  ms  open()
  ms  open()
  ms  open()
           /* TID 0x1ee1 */
  ms  open()
           /* TID 0x1ee5 */
  ms  open()
           /* TID 0x1ee4 */
  ms  open()
  ms  open()
  ms  open()
  ms  open()
           /* TID 0x1ec3 */
  ms  open()
           /* TID 0x1ee8 */
  ms  open()
           /* TID 0x1eeb */
  ms  open()
  ms  open()
           /* TID 0x1ee7 */
  ms  open()
           /* TID 0x1eeb */
  ms  open()
           /* TID 0x1ee7 */
  ms  open()
           /* TID 0x1eee */
  ms  open()
           /* TID 0x1ee4 */
  ms  open()
```



5.7 Android Export flag is enabled

Vulnerability Severity	OWASP Category

Low	Improper Platform Usage
Tools Used	**Date of reporting**
Sublime text	11th March 2025

Vulnerability Observation

During the assessment, it was observed that the application utilizes activities, services, and broadcast receivers that are not adequately protected. Additionally, the protection level needs to be verified, especially when an intent filter is present, as it may allow unauthorized access or interaction.

Vulnerable location

AndroidManifest.xml

Technical Impact

An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.

Business impact

When the Android exported flag is enabled for an activity, service, or receiver, it can be accessed by other applications, potentially leading to unauthorized data exposure or component manipulation. This increases the risk of security threats like intent hijacking, privilege escalation, and unauthorized execution of app functions.

Note: Most of the activities, services, and broadcasts mentioned in this report are not exploitable; however, we recommend verifying them on your end and ensuring proper intent filters are applied to all mentioned components. Exploitable components have been listed separately in this report.

Remediation

Implement protection level to signature so that only applications signed with the same certificate can access the activity.

Status - Acknowledge

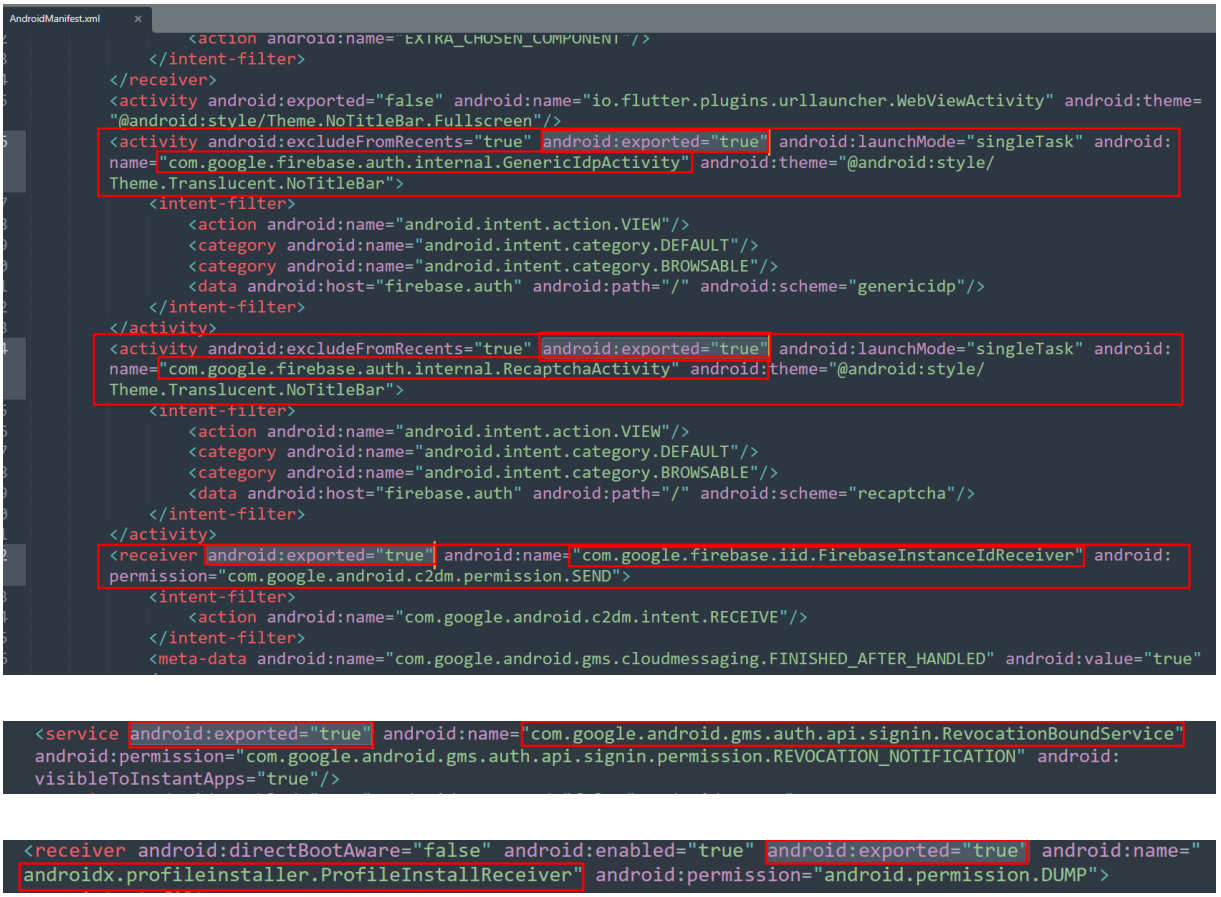
References

- OWASP Mobile Security Testing Guide (MSTG)
- Android Developers Documentation
- Google Security Best Practices
- OWASP Mobile Top 10

Steps to Reproduce

1. Decompile the apk and open AndroidManifest.xml file with sublime text and analyze the android:exported=”true”.

```
<receiver android:exported="true" android:name="
io.flutter.plugins.firebase.messaging.FlutterFirebaseMessagingReceiver" android:permission="
com.google.android.c2dm.permission.SEND">
```



5.8 App can be installed on a vulnerable upatched Android version

Vulnerability Severity	**OWASP Category**
Low	Improper Platform Usage
Tools Used	**Date of reporting**
Jadx-gui & Memu	11th March 2025

Vulnerability Observation

The app can be installed on a vulnerable, unpatched Android version (Android 7.0, [minSdk=24]). This application supports installation on older Android versions with multiple known vulnerabilities that remain unpatched. These devices no longer receive security updates from Google, posing a significant security risk.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk.

Technical Impact

Installing an Android app on an outdated version can expose devices to critical security risks, such as data breaches, malware infections, and privilege escalation. Older OS versions lack modern protections and patches, allowing attackers to exploit known vulnerabilities, gain unauthorized access, and steal sensitive

information. This can lead to app instability, compromised personal data, and potential long-term damage, making it crucial to use updated Android versions for security and performance.

Business impact

Using an unpatched Android version with known vulnerabilities can be exploited through the installation of the [xenea-wallet] app. Attackers can leverage security flaws to execute arbitrary code, escalate privileges, or compromise sensitive data, leading to potential financial and privacy risks.

Remediation

Supporting older Android versions with multiple known security vulnerabilities increases the risk of exploitation, as these devices no longer receive security updates from Google. It is recommended to enforce a minimum supported Android version of 10 (API 29) or higher to ensure a more secure environment.

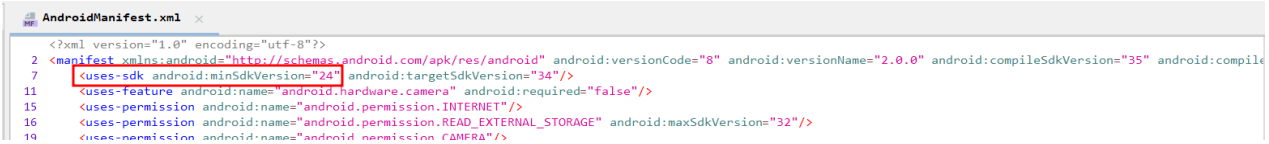
Status - Fixed

References

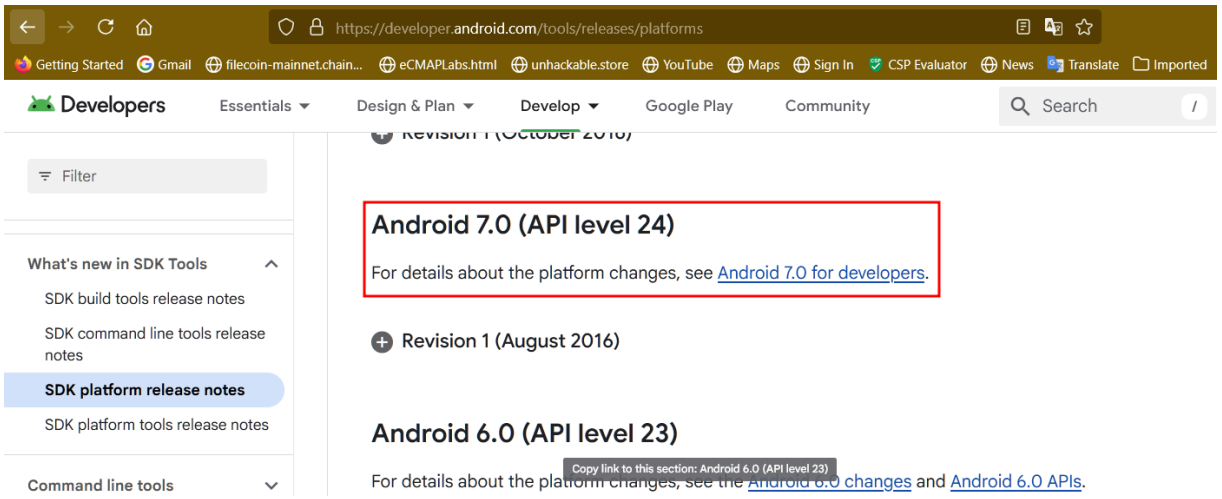
- Android Security Bulletin
- Google Play's Target API Level Policy
- Android Version Distribution & Support

Steps to Reproduce

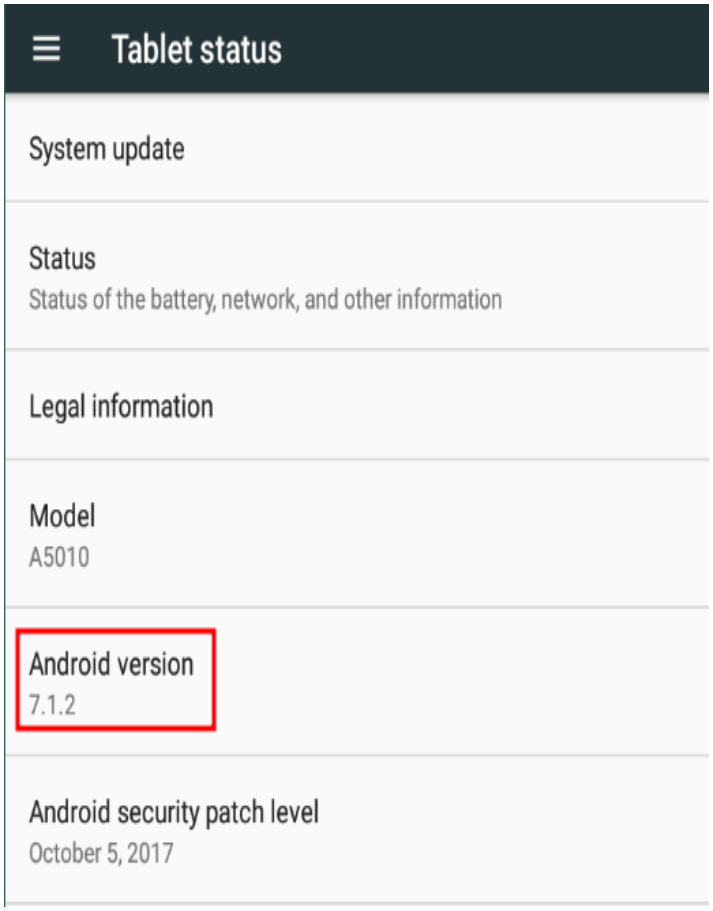
1. using jadx-gui tools we decompile the application and analysis it's android-manifest.xml file

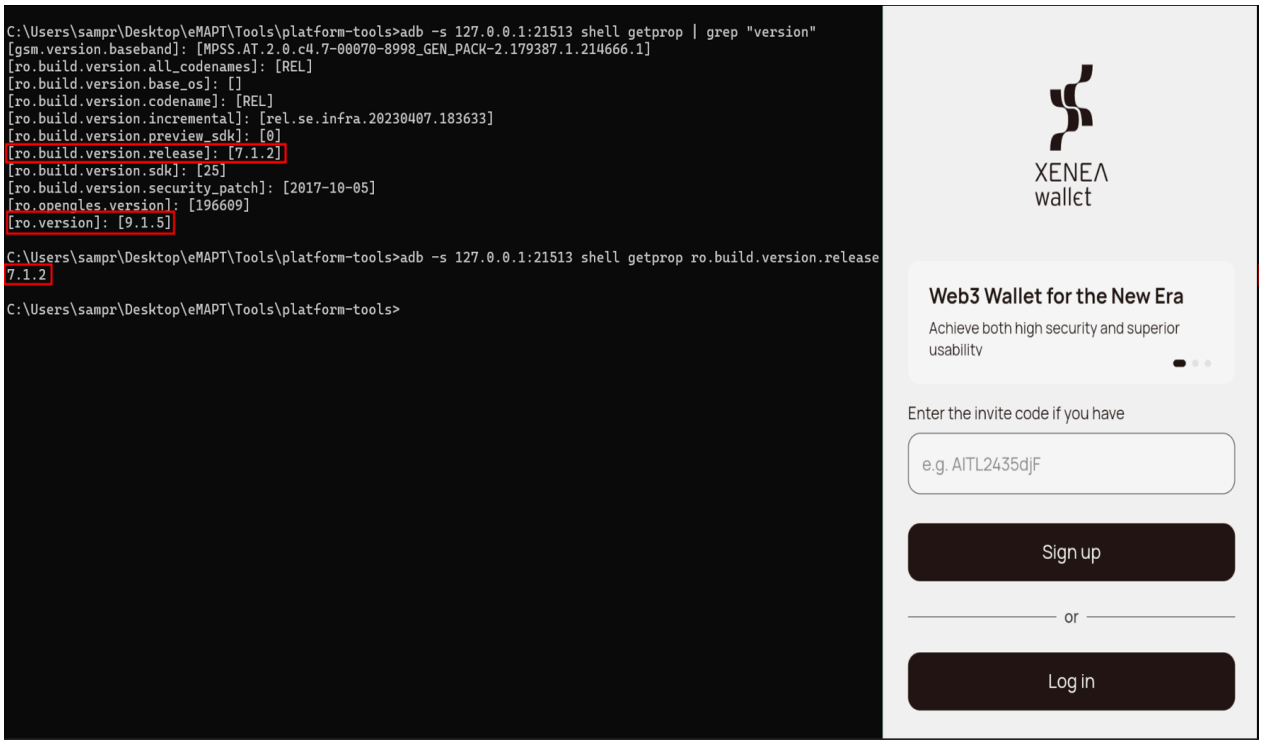


2. while reading android-manifest.xml file we paid attention to it's sdk-version and we see that it's sdk-version is 24.



3. Which can be installed on older android version.





5.9 App creates temp file

Vulnerability Severity	OWASP Category

Low	Insecure Data Storage
Tools Used	**Date of reporting**
JADX-GUI	11th March 2025

Vulnerability Observation

During the assessment, it was observed that the application creates a temporary file, which may contain sensitive information. Sensitive data should never be written into a temporary file without proper security measures.

Vulnerable location

Throughout the xenea-wallet.apk

Technical Impact

An attacker with access to the temporary file could retrieve sensitive data, potentially using it for future attacks.

Business impact

Exposure of sensitive information could lead to data leakage, regulatory compliance violations, and reputational damage to the organization.

Remediation

- Avoid creating temporary files if not necessary.
- If creating a temporary file is essential, ensure that sensitive information is encrypted before being stored.
- Securely delete temporary files after use to prevent unauthorized access.

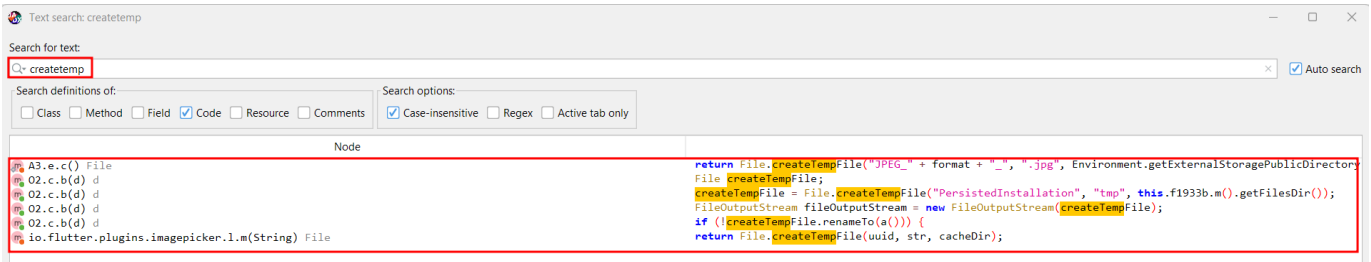
Status - Acknowledge

References

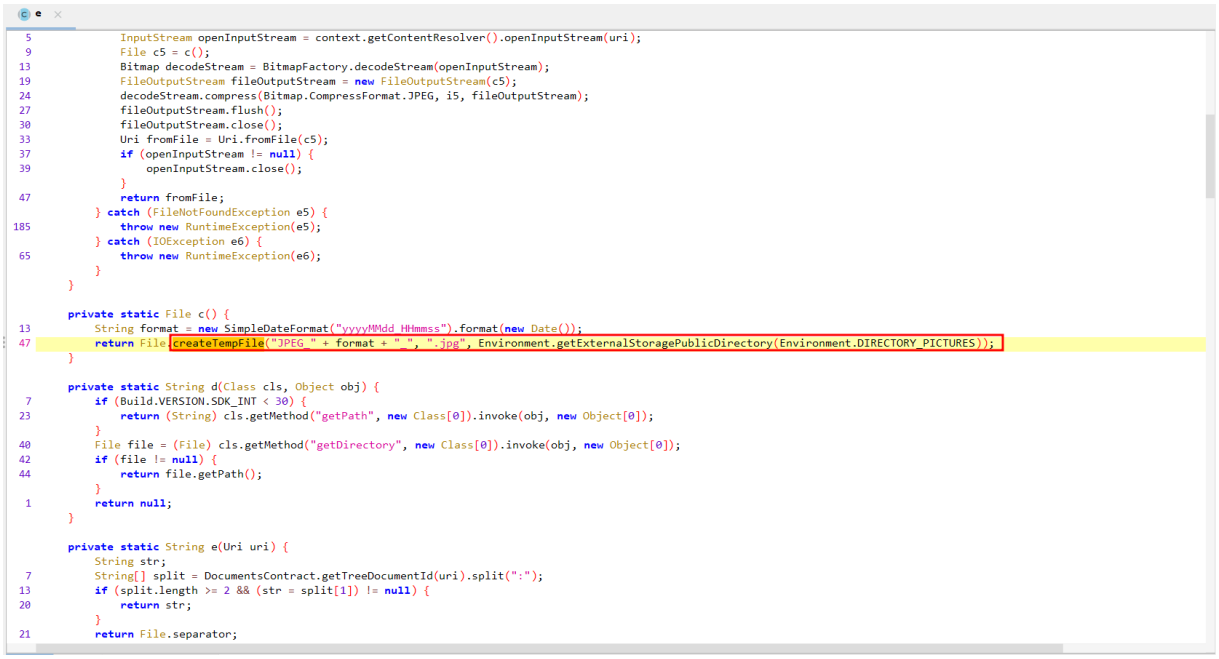
- OWASP Mobile Security Testing Guide (MSTG) - Storage Security
- Android Developers - Internal Storage Best Practices
- Android Security Best Practices - Temporary Files and Data Leakage
- OWASP Mobile Top 10 - Insecure Data Storage (M2)

Steps to Reproduce

1. Open the application in JADX-GUI search, createtemp and observe the file's.



2. Open any of the files and observe that application using the createTemp function.



```
5           InputStream openInputStream = context.getContentResolver().openInputStream(uri);
9           File c5 = c();
13          Bitmap decodeStream = BitmapFactory.decodeStream(openInputStream);
19          FileOutputStream fileOutputStream = new FileOutputStream(c5);
24          decodeStream.compress(Bitmap.CompressFormat.JPEG, i5, fileOutputStream);
27          fileOutputStream.flush();
30          fileOutputStream.close();
33          Uri fromFile = Uri.fromFile(c5);
37          if (openInputStream != null) {
39              openInputStream.close();
            }
47          return fromFile;
        } catch (FileNotFoundException e5) {
185         throw new RuntimeException(e5);
        } catch (IOException e6) {
65          throw new RuntimeException(e6);
        }
    }

    private static File c() {
13      String format = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
47      return File.createTempFile("JPEG_" + format + "_", ".jpg", Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES));
    }

    private static String d(Class cls, Object obj) {
7       if (Build.VERSION.SDK_INT < 30) {
23          return (String) cls.getMethod("getPath", new Class[0]).invoke(obj, new Object[0]);
        }
40      File file = (File) cls.getMethod("getDirectory", new Class[0]).invoke(obj, new Object[0]);
42      if (file != null) {
44          return file.getPath();
        }
1       return null;
    }

    private static String e(Uri uri) {
        String str;
7       String[] split = DocumentsContract.getTreeDocumentId(uri).split(":");
13      if (split.length >= 2 && (str = split[1]) != null) {
20          return str;
        }
21      return File.separator;
```

5.10 Improper Input Validation

Vulnerability Severity	**OWASP Category**
Medium	Injection (Cross-Site Scripting - XSS)
Tools Used	**Date of reporting**
Burp Suite	11th March 2025

Vulnerability Observation

During testing, it was observed that user input from multiple insertion points is reflected in the application's response without proper sanitization or encoding. This improper input validation allows the injection of special characters and script payloads, leading to potential security risks such as Cross-Site Scripting (XSS), data leakage, and unauthorized data exposure. Additionally, certain API endpoints store user input and later return it in responses, increasing the risk of Stored XSS and sensitive data disclosure. The lack of proper validation and content-type enforcement makes the application susceptible to manipulation and exploitation by an attacker.

Vulnerable location

http://api-stg.xenea.network/api/v1/admin/wallets?page=1&size=100

http://api-stg.xenea.network/api/v1/wallet/pre-recovery

https://mobile-api.staging.xenea.app/wallet/v1/create

Parameter: size, checksum & address

Technical Impact

Improper input validation can lead to various security vulnerabilities, including Cross-Site Scripting (XSS), Injection Attacks, and Data Manipulation. An attacker can inject malicious scripts that execute in a victim’s browser, potentially stealing session tokens, credentials, or other sensitive information. If

exploited in an API request, this vulnerability can allow unauthorized data modification, bypassing security mechanisms and leading to unintended application behavior.

Business Impact

The vulnerability allows attackers to inject malicious JavaScript, leading to session hijacking, credential theft, phishing attacks, and regulatory non-compliance (OWASP ASVS, GDPR, PCI DSS). Exploitation can compromise user accounts, harm reputation, and deceive users into providing sensitive information.

Remediation

To mitigate this, implement strict input validation, proper output encoding, secure content-type headers, and use a web application firewall (WAF).

Status - Fixed

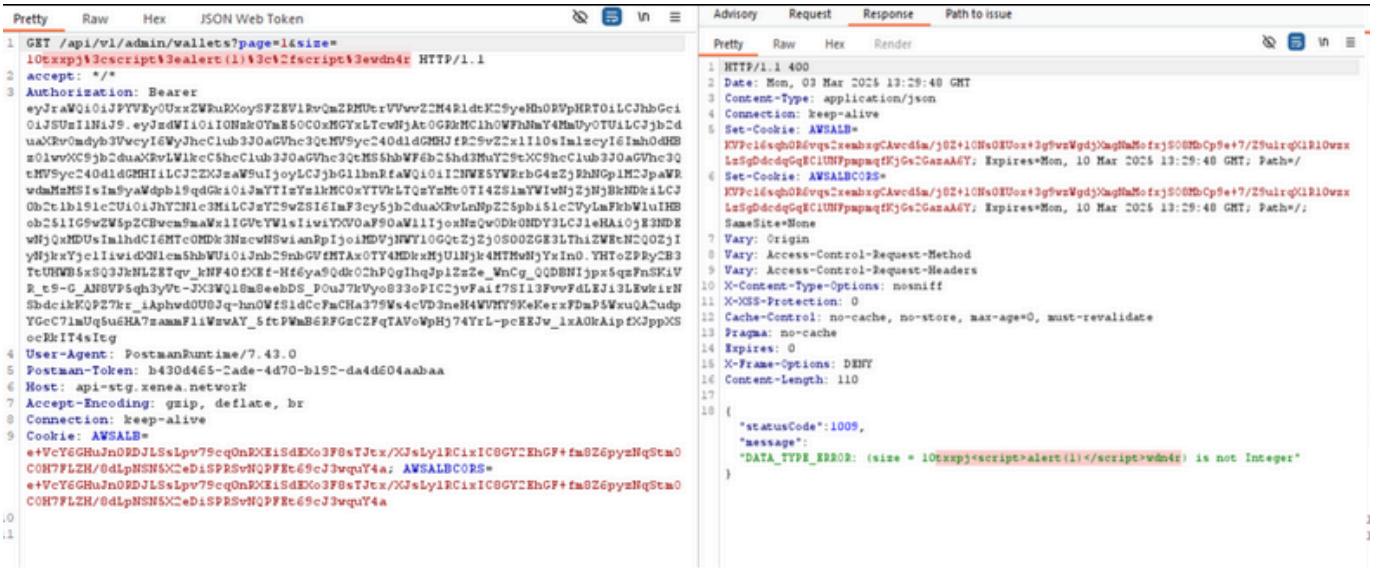
References

1. Web Security Academy - Cross-Site Scripting
2. OWASP Cross-Site Scripting (XSS) Prevention Cheat Sheet

Steps to Reproduce

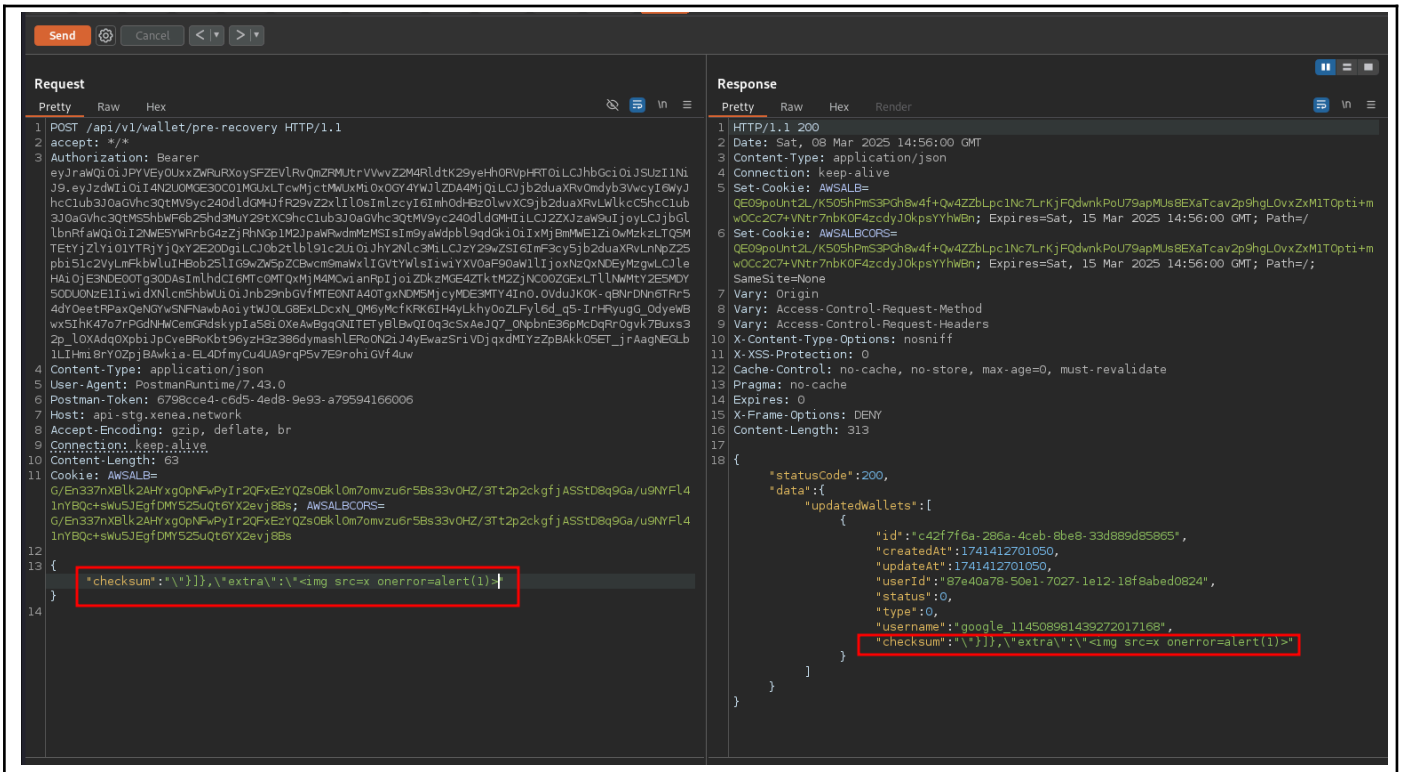
Case 1:

1. Capture the request by burp suite proxy service. Inject the payload txxpj<script>alert(1)</script>wdn4r into the SIZE param Send the request and observe the application's response the input is reflected without sanitization



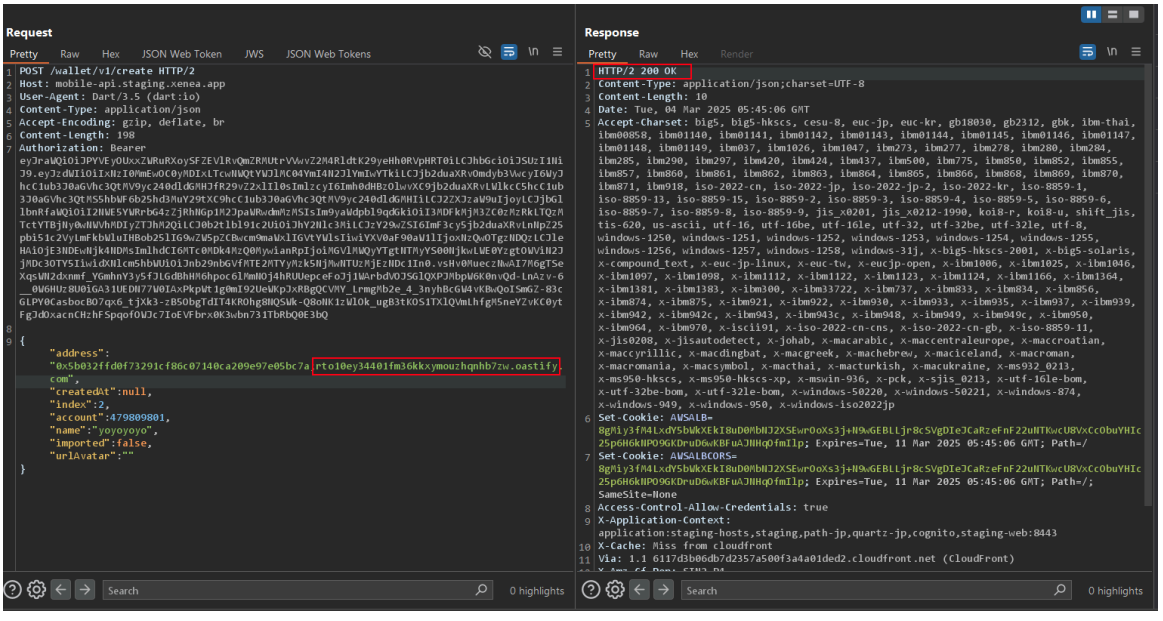
Case 2:

1. Capture the request by burp suite proxy service. Add the payload: \"}]},\"extra\":\"<img src=x onerror=alert(1)> in the checksum.

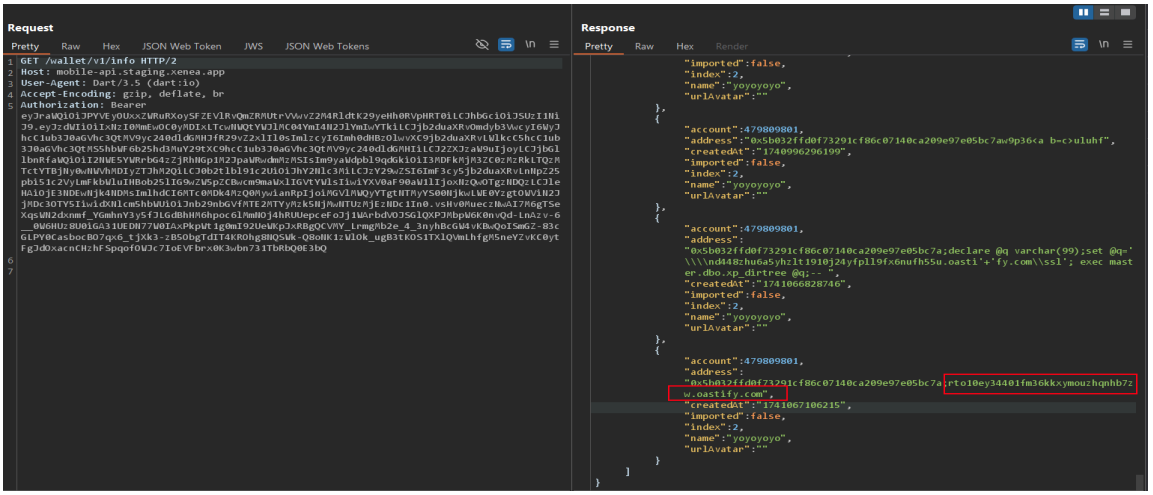


Case 3:

1. Capture the POST request to /wallet/v1/create using Burp Suite, inject a malicious payload into the request body, and send the request.



2. The response from the corresponding GET request (e.g., /wallet/v1/info) and verify that the injected input is reflected in the API response, confirming stored input disclosure.



5.11 Application Data can be Backed up

Vulnerability Severity	**OWASP Category**
Medium	Insecure Data Storage
Tools Used	**Date of reporting**
Sublime text	11th March 2025

Vulnerability Observation

The flag [android:allowBackup] should be set to false. By default, it is set to true, allowing anyone to back up the application's data via ADB. This could enable users with USB debugging enabled to extract application data from the device, posing a security risk.

Vulnerable location

AndroidManifest.xml

Technical Impact

If an Android app sets android:allowBackup="true", it allows sensitive data, like user preferences, databases, and files, to be backed up and potentially accessed by unauthorized parties. This creates a significant security risk, as attackers could exploit backup files to retrieve private information or inject malicious data. It bypasses Android's security model, making data vulnerable during device backups, risking privacy breaches and unauthorized access.

Business impact

If an Android app's backup feature is insecure, sensitive user data such as credentials, session tokens, and personal information may be stored unprotected. This can lead to unauthorized access, data breaches, and compliance violations, posing a significant risk to user privacy and the organization's reputation.

Remediation

To remediate the risk of android:allowBackup="true", disable the backup feature by setting android:allowBackup="false" in the app's manifest. This prevents sensitive data from being included in device backups and reduces the chance of unauthorized access. Additionally, ensure proper encryption

of sensitive data, implement secure storage methods, and educate users on the importance of securing their device to prevent data leakage.

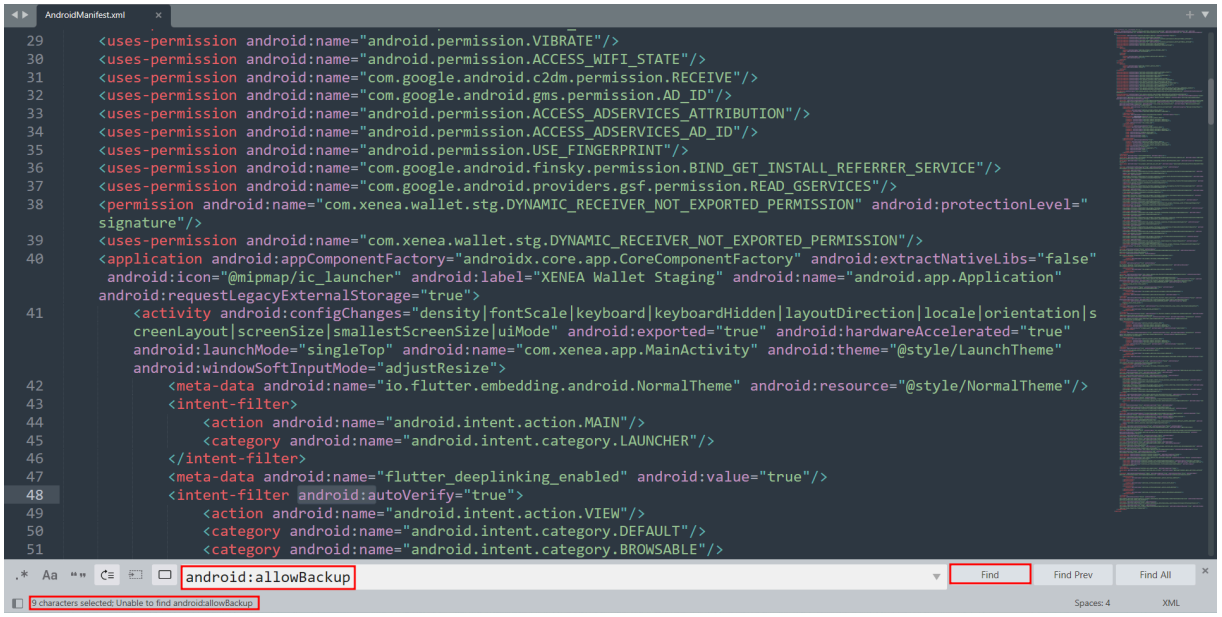
Status - Fixed

References

- OWASP Mobile Security Testing Guide - Backups
- Android allowBackup Attribute
- Android Security Best Practices

Steps to Reproduce

1. First use jadx-gui tool to decompile the android application and analysis it's android-manifest.xml file reading the android-manifest.xml file we find android:allow-backup=true



5.12 Application logs sensitive information

Vulnerability Severity	OWASP Category
Medium	Logging of Sensitive Data
Tools Used	**Date of reporting**
adb & Memu	11th March 2025

Vulnerability Observation

During testing, it was observed that the application logs Base64-encoded data, which, when decoded, reveals sensitive information such as the account ID, master private key, and wallet ID. Logging such sensitive data poses a significant security risk as it can be accessed by unauthorized users or malicious applications, leading to potential data exposure and compromise.

Vulnerable location

Throughout the xenea-wallet.apk while the enter pin is implemented.

Technical Impact

Sensitive information such as the account ID, master private key, and wallet ID being logged in plaintext (even if Base64-encoded) increases the risk of exposure. Attackers with access to log files can easily decode and retrieve critical data, potentially leading to unauthorized access, account takeover, and compromise of user funds or private keys.

Business impact

The leakage of sensitive user data can result in financial loss, reputational damage, regulatory non-compliance, and legal consequences. If an attacker gains access to private keys or wallet credentials, it could lead to unauthorized transactions, identity theft, and loss of customer trust, affecting business continuity and user confidence.

Remediation

Ensure that sensitive data such as account ID, master private key, and wallet ID is never logged, even in encoded form, by reviewing and removing all logging statements that store such information in application logs.

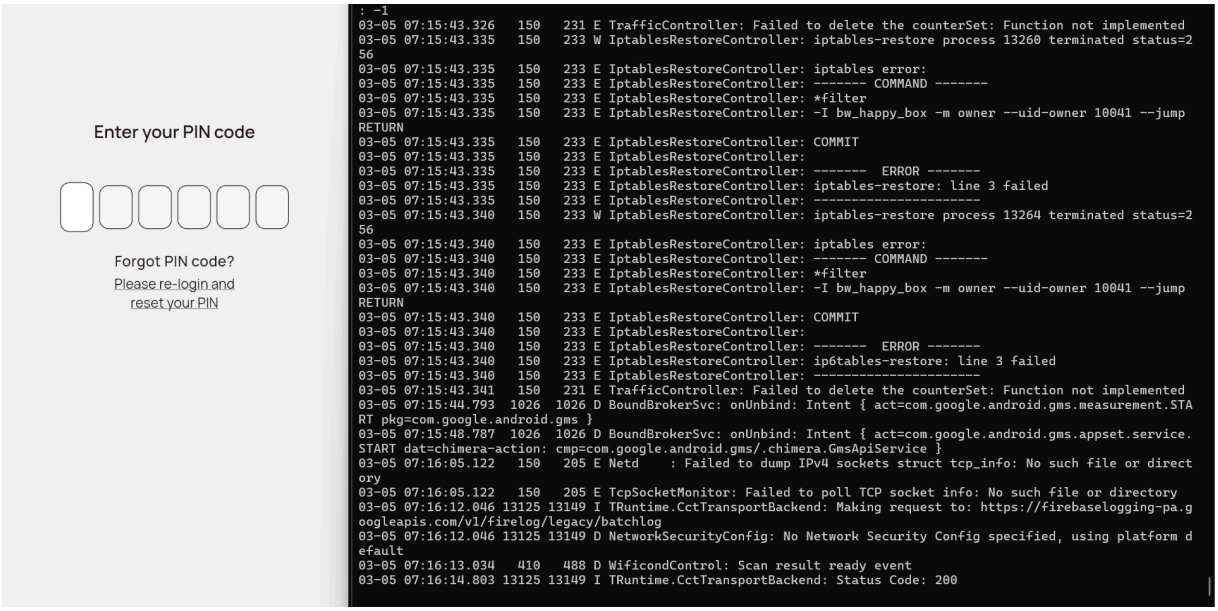
Status - Fixed

References

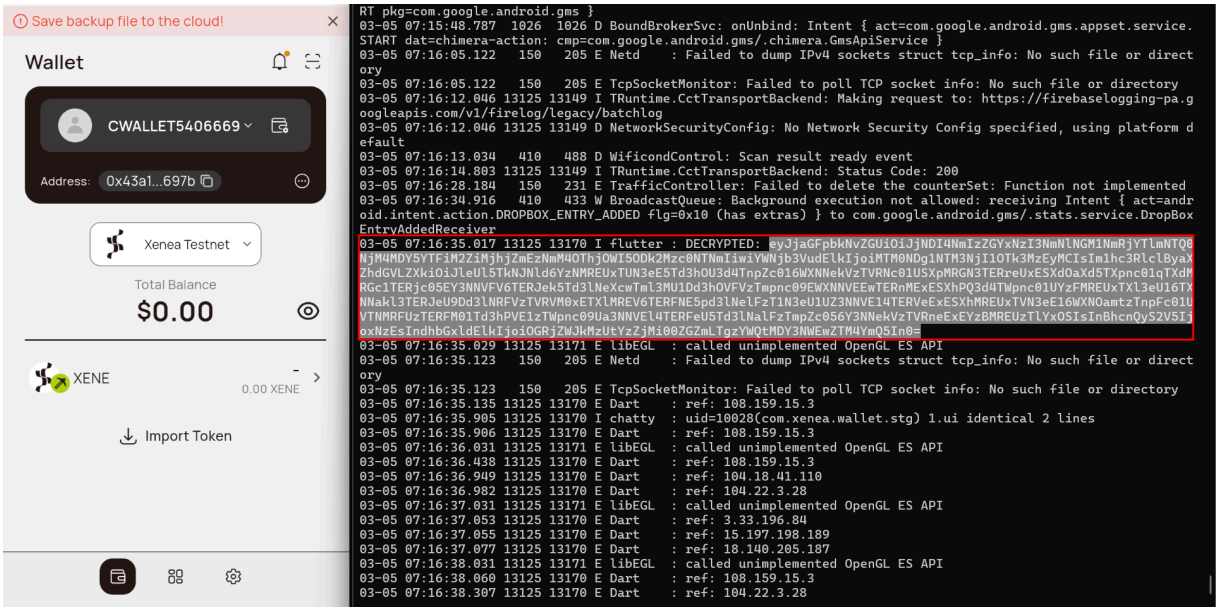
- OWASP Mobile Security Testing Guide (MSTG) - Logging
- Insufficient Security Logging and Monitoring
- Network security configuration

Steps to Reproduce

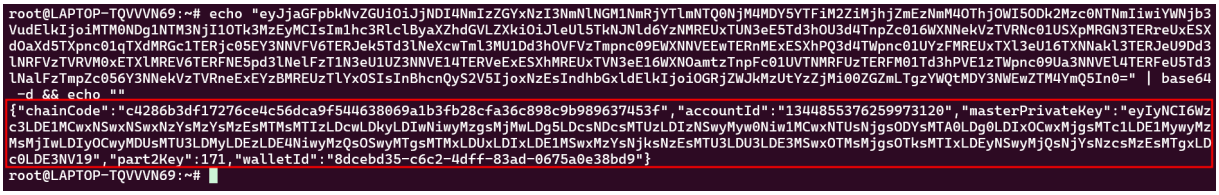
1. Enter the PIN and run adb logcat.



2. Now, once the app will open we will see base64 encoded text in the logs.



3. Now decode the base64 encoding text and observe the output.



5.13 CBC mode leads to oracle padding attack

Vulnerability Severity	**OWASP Category**
Medium	Insufficient Cryptography
Tools Used	**Date of reporting**
Jadx-gui	11th March 2025

Vulnerability Observation

During testing, we observe that the application uses "AES/CBC/PKCS7Padding" for an encryption operation which is vulnerable to padding oracle attacks.

Vulnerable location

Throughout the xenea-wallet.apk Application

Technical Impact

The existence of a padding oracle allows an attacker to decrypt encrypted data and encrypt arbitrary data without knowledge of the key used for these cryptographic operations. This can lead to leakage of sensible data or to privilege escalation vulnerabilities, if integrity of the encrypted data is assumed by the application.

Business impact

If an attacker exploits the padding oracle vulnerability in the application's encryption mechanism, they may decrypt sensitive information, such as user credentials, financial data, or personal information. This

can lead to data breaches, identity theft, regulatory non-compliance, and reputational damage to the organization.

Remediation

Use "AES/GCM/NoPadding" instead of "AES/CBC/PKCS5Padding" & "AES/CBC/PKCS7Padding"

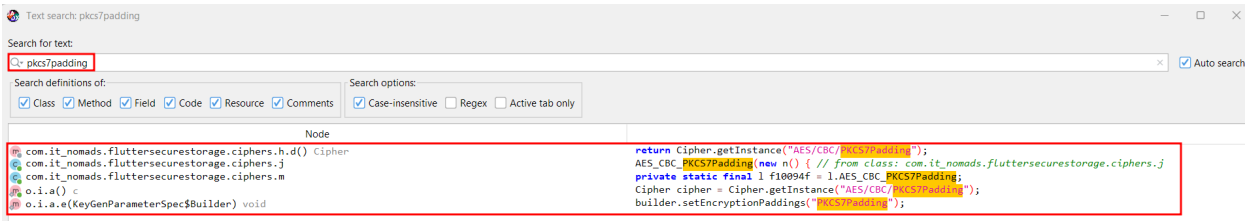
Status - Fixed

References

- Testing for Padding Oracle
- Padding Oracle Crypto Attack

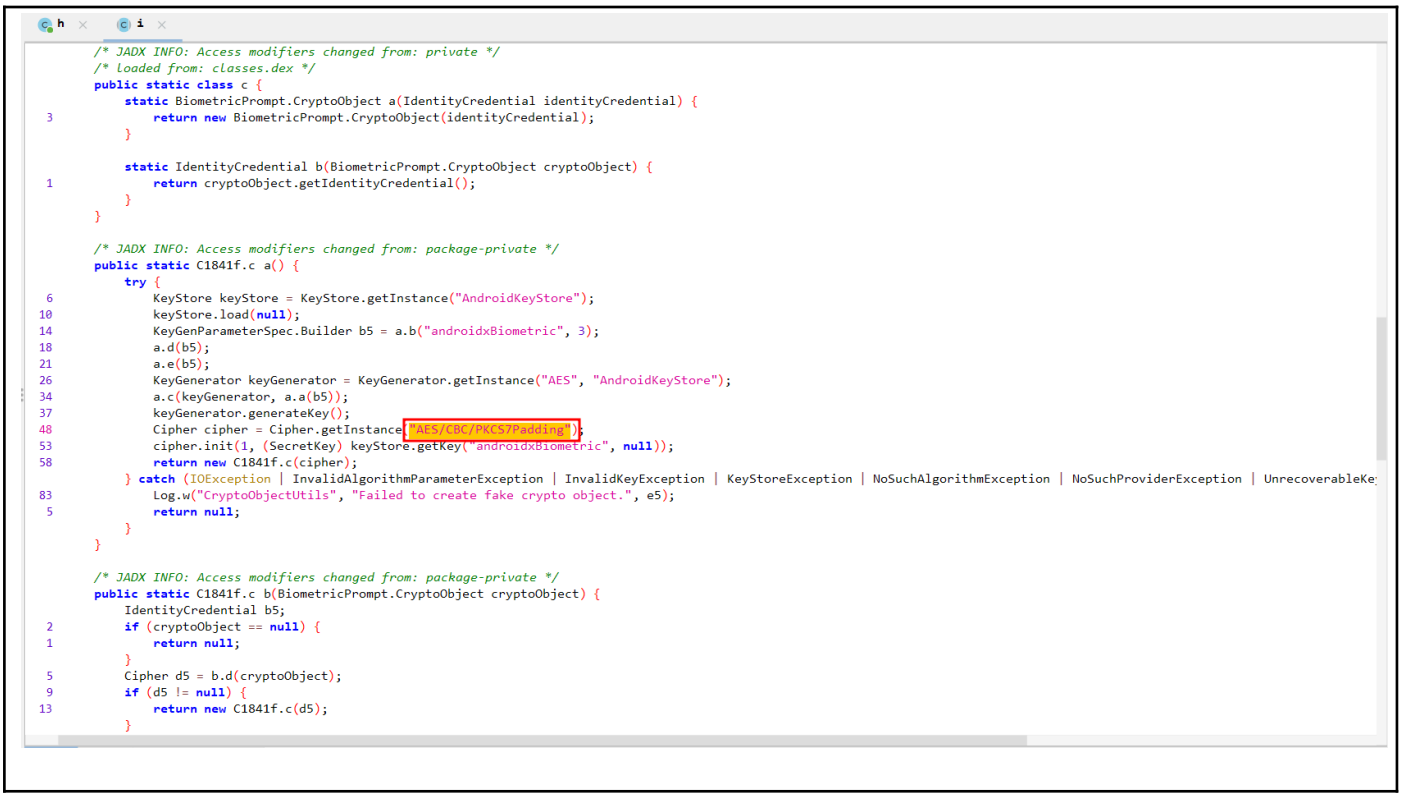
Steps to Reproduce

1. Open the application in JADX-GUI and search “PKCS7Padding” and observe that the application uses "AES/CBC/PKCS7Padding" which is vulnerable to padding oracle attacks.



2. Open any one of the files and observe the "AES/CBC/PKCS7Padding"

```
h  i
48      @Override // com.it_nomads.fluttersecurestorage.ciphers.i
49      public byte[] a(byte[] bArr) {
50          int e5 = e();
51          byte[] bArr2 = new byte[e5];
52          this.f10091b.nextBytes(bArr2);
53          this.f10090a.init(1, this.f10092c, f(bArr2));
54          byte[] doFinal = this.f10090a.doFinal(bArr);
55          byte[] bArr3 = new byte[doFinal.length + e5];
56          System.arraycopy(bArr2, 0, bArr3, 0, e5);
57          System.arraycopy(doFinal, 0, bArr3, e5, doFinal.length);
58          return bArr3;
59      }
60
61      @Override // com.it_nomads.fluttersecurestorage.ciphers.i
62      public byte[] b(byte[] bArr) {
63          int e5 = e();
64          byte[] bArr2 = new byte[e5];
65          System.arraycopy(bArr, 0, bArr2, 0, e5);
66          AlgorithmParameterSpec f5 = f(bArr2);
67          int length = bArr.length - e();
68          byte[] bArr3 = new byte[length];
69          System.arraycopy(bArr, e5, bArr3, 0, length);
70          this.f10090a.init(2, this.f10092c, f5);
71          return this.f10090a.doFinal(bArr3);
72      }
73
74      protected String c() {
75          return "VGhpcyBpcyB0aGUga2V5IGZvciBhIHNlY3VyZSBzdG9yYWdlIEFFUyBLZXkK";
76      }
77
78      protected Cipher d() {
79          return Cipher.getInstance("AES/CBC/PKCS7Padding");
80      }
81
82      protected int e() {
83          return 16;
84      }
85
86      protected AlgorithmParameterSpec f(byte[] bArr) {
87          return new IvParameterSpec(bArr);
88      }
89  }
```



```
h  i
/* JADX INFO: Access modifiers changed from: private */
/* loaded from: classes.dex */
public static class c {
    static BiometricPrompt.CryptoObject a(IdentityCredential identityCredential) {
        return new BiometricPrompt.CryptoObject(identityCredential);
    }

    static IdentityCredential b(BiometricPrompt.CryptoObject cryptoObject) {
        return cryptoObject.getIdentityCredential();
    }
}

/* JADX INFO: Access modifiers changed from: package-private */
public static C1841f.c a() {
    try {
        KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
        keyStore.load(null);
        KeyGenParameterSpec.Builder b5 = a.b("androidxBiometric", 3);
        a.d(b5);
        a.e(b5);
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore");
        a.c(keyGenerator, a.a(b5));
        keyGenerator.generateKey();
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        cipher.init(1, (SecretKey) keyStore.getKey("androidxBiometric", null));
        return new C1841f.c(cipher);
    } catch (IOException | InvalidAlgorithmParameterException | InvalidKeyException | KeyStoreException | NoSuchAlgorithmException | NoSuchProviderException | UnrecoverableKe
        Log.w("CryptoObjectUtils", "Failed to create fake crypto object.", e5);
        return null;
    }
}

/* JADX INFO: Access modifiers changed from: package-private */
public static C1841f.c b(BiometricPrompt.CryptoObject cryptoObject) {
    IdentityCredential b5;
    if (cryptoObject == null) {
        return null;
    }
    Cipher d5 = b.d(cryptoObject);
    if (d5 != null) {
        return new C1841f.c(d5);
    }
```

5.14 Exposed Debugging Symbols in Library

Vulnerability Severity	**OWASP Category**
Medium	Reverse Engineering
Tools Used	**Date of reporting**
objdump	11th March 2025

Vulnerability Observation

The application contains unremoved debugging symbols in its libraries, making it vulnerable to reverse engineering. These symbols expose critical details about the app’s internals, such as function names, memory addresses, and logic flow. During testing, it was observed that the application's shared library contains debugging symbols, which could aid an attacker in reverse-engineering the library and identifying potential security weaknesses.

Vulnerable location

lib/arm64-v8a
lib/armeabi-v7a
lib/x86
lib/x86_64

Technical Impact

Debugging symbol leaks the information such as debugging information, line numbers, and descriptive function or method names, make the binary or bytecode easier for the reverse engineer to understand, but these aren't needed in a release build and can therefore be safely omitted without impacting the app's functionality.

Business impact

The presence of unremoved debugging symbols increases the risk of reverse engineering, allowing attackers to analyze the application's internal logic, identify vulnerabilities, and develop exploits. This could lead to intellectual property theft, unauthorized modifications, and potential security breaches, ultimately affecting the application's integrity and user trust.

Remediation

To remediate the vulnerability of debugging symbols not being removed from the library, ensure that all debugging information is stripped during the build process. Use tools like ProGuard or R8 to obfuscate and optimize the code, making it harder for attackers to reverse-engineer. Additionally, configure the build system to disable debugging features in production and perform regular security audits to ensure that no unnecessary symbols are exposed in released app versions.

Status - Fixed

References

- Reverse Engineering and Tampering
- Android Developer Documentation - Debugging and Profiling
- NIST - Guide to Software Security Assurance
- Google Security Best Practices - Secure Your Code

Steps to Reproduce

1. Decompile the APK and navigate to the lib folder. Observe the highlighted directories corresponding to different architectures. Enter any of these directories and examine the .so files present.

```
C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is C422-85F9

 Directory of C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib

26-02-2025  21:33    <DIR>          .
26-02-2025  21:33    <DIR>          ..
26-02-2025  21:33    <DIR>          arm64-v8a
26-02-2025  21:33    <DIR>          armeabi-v7a
26-02-2025  21:33    <DIR>          x86
26-02-2025  21:33    <DIR>          x86_64
               0 File(s)              0 bytes
               6 Dir(s)  33,910,857,728 bytes free

C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib>cd x86_64

C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is C422-85F9

 Directory of C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64

26-02-2025  21:33    <DIR>          .
26-02-2025  21:33    <DIR>          ..
26-02-2025  21:33        17,302,432 libapp.so
26-02-2025  21:33         5,521,032 libbarhopper_v3.so
26-02-2025  21:33        11,768,080 libflutter.so
26-02-2025  21:33            46,200 libimage_processing_util_jni.so
26-02-2025  21:33         1,375,416 libsqlite3.so
               5 File(s)     36,013,160 bytes
               2 Dir(s)  33,915,043,840 bytes free

C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64>
```

2. Now, run objdump.exe against any .so file and examine the extracted symbols.

```
C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools\winlibs-x86_64-posix-seh-gcc-14.2.0-llvm-19.1.7-mingw-w64ucrt-12.0.0-r3\mingw64\bin>objdump.exe -T C:\User
s\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64\libapp.so

C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64\libapp.so:     file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
0000000000650000 g    DO .text  0000000000015f40 _kDartVmSnapshotInstructions
0000000000665f40 g    DO .text  000000000a0cdf0 _kDartIsolateSnapshotInstructions
0000000000000200 g    DO .rodata        0000000000003ea0 _kDartVmSnapshotData
00000000000040c0 g    DO .rodata        0000000000640af0 _kDartIsolateSnapshotData
00000000000001c8 g    DO .note.gnu.build-id     0000000000000020 _kDartSnapshotBuildId



C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools\winlibs-x86_64-posix-seh-gcc-14.2.0-llvm-19.1.7-mingw-w64ucrt-12.0.0-r3\mingw64\bin>objdump.exe -T C:\User
s\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64\libbarhopper_v3.so

C:\Users\sampr\Desktop\eMAPT\Tools\xenea-wallet\lib\x86_64\libbarhopper_v3.so:     file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
0000000000000000      DF *UND*  0000000000000000 (LIBC)       __cxa_finalize
0000000000000000      DF *UND*  0000000000000000 (LIBC)       __cxa_atexit
0000000000000000      DF *UND*  0000000000000000  Base        AndroidBitmap_getInfo
0000000000000000      DF *UND*  0000000000000000  Base        AndroidBitmap_lockPixels
0000000000000000      DF *UND*  0000000000000000  Base        AndroidBitmap_unlockPixels
0000000000000000      DF *UND*  0000000000000000 (LIBC)       strlen
0000000000000000      DF *UND*  0000000000000000 (LIBC)       memset
0000000000000000      DF *UND*  0000000000000000 (LIBC)       memcpy
0000000000000000      DF *UND*  0000000000000000 (LIBC)       abort
0000000000000000      DF *UND*  0000000000000000 (LIBC)       memmove
0000000000000000      DF *UND*  0000000000000000 (LIBC)       expf
0000000000000000      DF *UND*  0000000000000000 (LIBC)       logf
0000000000000000      DF *UND*  0000000000000000 (LIBC)       __stack_chk_fail
0000000000000000      DF *UND*  0000000000000000 (LIBC)       exp
0000000000000000      DF *UND*  0000000000000000 (LIBC)       atan2f
0000000000000000      DF *UND*  0000000000000000 (LIBC)       sincosf
0000000000000000      DF *UND*  0000000000000000 (LIBC)       memcmp
0000000000000000      DF *UND*  0000000000000000 (LIBC)       memchr
0000000000000000      DF *UND*  0000000000000000 (LIBC)       strtod
0000000000000000      DF *UND*  0000000000000000 (LIBC)       strtol
```

5.15 Deeplink Misconfiguration Enables Unauthorized Navigation

Vulnerability Severity	**OWASP Category**
Low	Insecure Communication
Tools Used	**Date of reporting**
jadx-gui & adb	11th March 2025

Vulnerability Observation

We observed that the application is vulnerable to deep link misconfigurations, allowing untrusted or improperly validated deep links to trigger unintended actions. This could enable unauthorized navigation within the app, potentially exposing sensitive functionality or data.

Vulnerable location

AndroidManifest.yml

Technical Impact

An attacker can exploit deep link misconfigurations to bypass authentication, access restricted areas of the application, or manipulate app behavior, potentially leading to data leakage, privilege escalation, or unauthorized transactions.

Business Impact

Unauthorized access to sensitive app features or data can lead to reputational damage, financial losses, and compliance violations. Attackers may exploit this flaw to gain access to user accounts, perform fraudulent actions, or disrupt business operations.

Remediation

Validate and restrict deep links by implementing intent filters with proper authentication checks and allowlisting trusted sources to prevent unauthorized access.

Status - Acknowledge

References

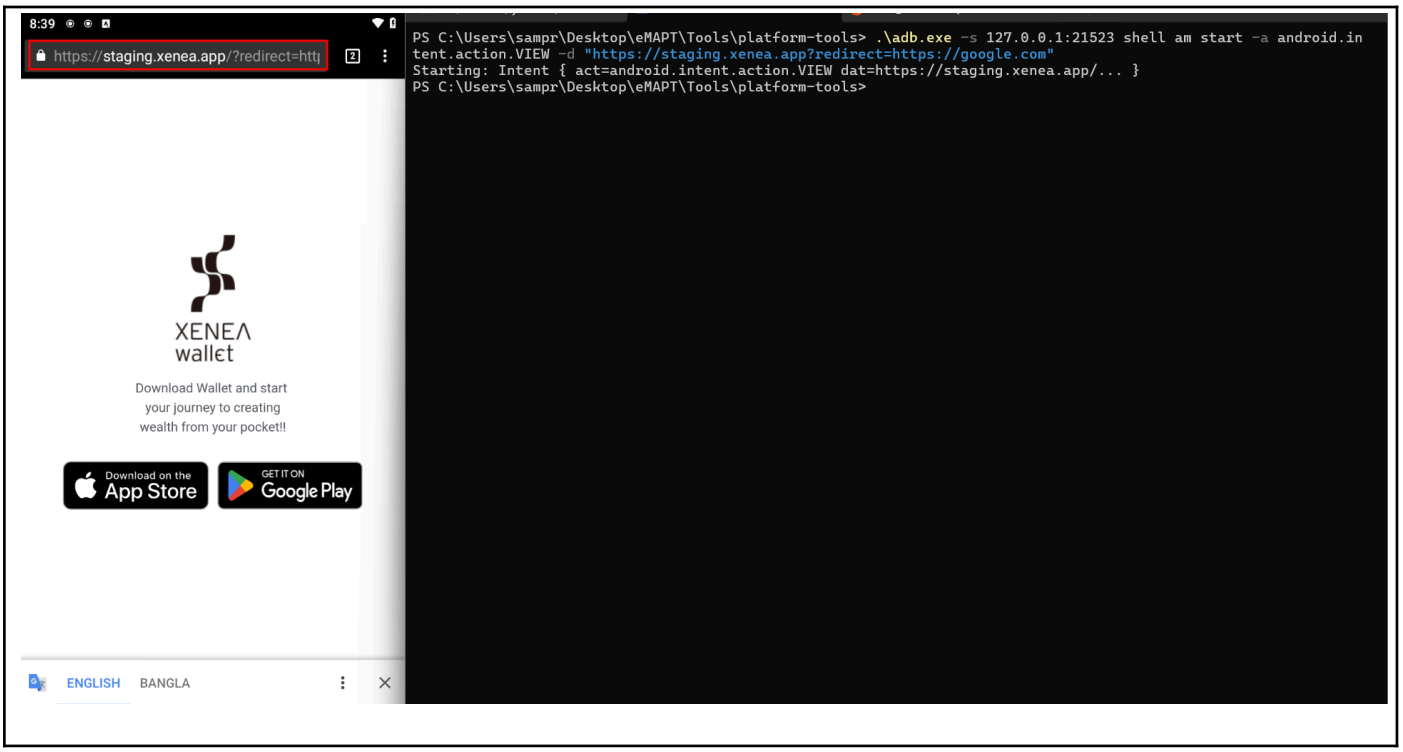
- OWASP Mobile Security Testing Guide (MSTG)
- Android Developers - Deep Links
- Google Security Best Practices for Deep Links

Steps to Reproduce

1. First open the apk with jadx-gui and observe the AndroidManifest.yml file.

```
<meta-data android:name="flutter_deeplinking_enabled" android:value="true"/>
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="com.xenea.app"/>
</intent-filter>
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:host="staging.xenea.app"/>
    <data android:pathPattern="/register/.*"/>
    <data android:scheme="http"/>
    <data android:scheme="https"/>
</intent-filter>
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:host="xenea.app"/>
    <data android:pathPattern="/register/.*"/>
    <data android:scheme="http"/>
    <data android:scheme="https"/>
```

2. When we analyze android manifest.xml file we find that there is an activity tag in this we see one tag in which is responsible for deep link means using deep link when we click on an application it's redirected to a website and we can exploit this with adb am.



5.16 Sensitive Google API Key Leaked in Android App Source Code

Vulnerability Severity	**OWASP Category**
Low	Improper Security Configuration
Tools Used	**Date of reporting**
jadx-gui & sublime text	11th March 2025

Vulnerability Observation

During testing, it was observed that the application stores an unobfuscated API key in the strings.xml and AndroidManifest.xml files. Storing sensitive keys in plaintext within these files makes them easily accessible through reverse engineering, potentially leading to unauthorized access and API abuse.

Vulnerable location

/res/vaues/string.xml
AndroidManifest.yml

Technical Impact

If the device is compromised, the malicious user will be able to get these sensitive data via the device and can then perform further exploitation using these API_KEY.

Business impact

Exposing API keys in plaintext within the application files can lead to unauthorized access, API abuse, and data breaches. Attackers can extract these keys through reverse engineering and use them to interact with backend services, impersonate legitimate requests, or perform malicious activities such as data exfiltration, fraud, or denial-of-service (DoS) attacks. This can result in financial losses, reputational damage, and compliance violations.

Remediation

Store API keys securely using environment variables, cloud-based secret management services, or encrypted storage. Avoid hardcoding sensitive keys in application files like strings.xml or AndroidManifest.xml. Implement API key restrictions (e.g., IP-based, domain-based) and rotate keys periodically to minimize the risk of exposure.

Note: We attempted to exploit this API key, but it was not found to be exploitable. However, we still recommend removing it if it is not in use to minimize security risks.

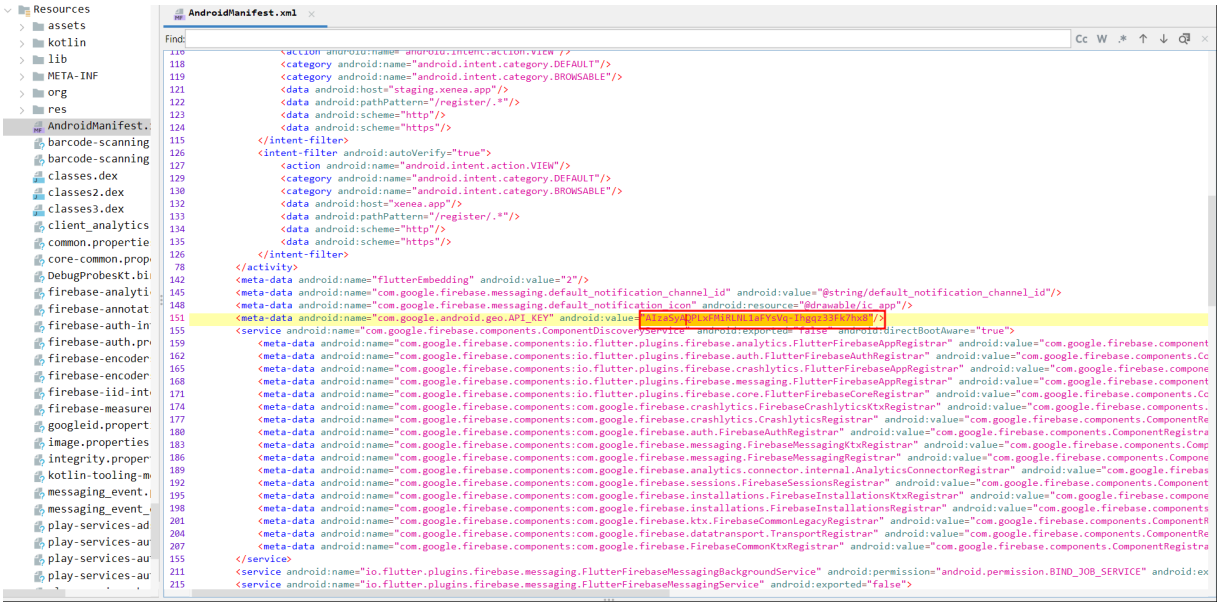
Status - Fixed

References

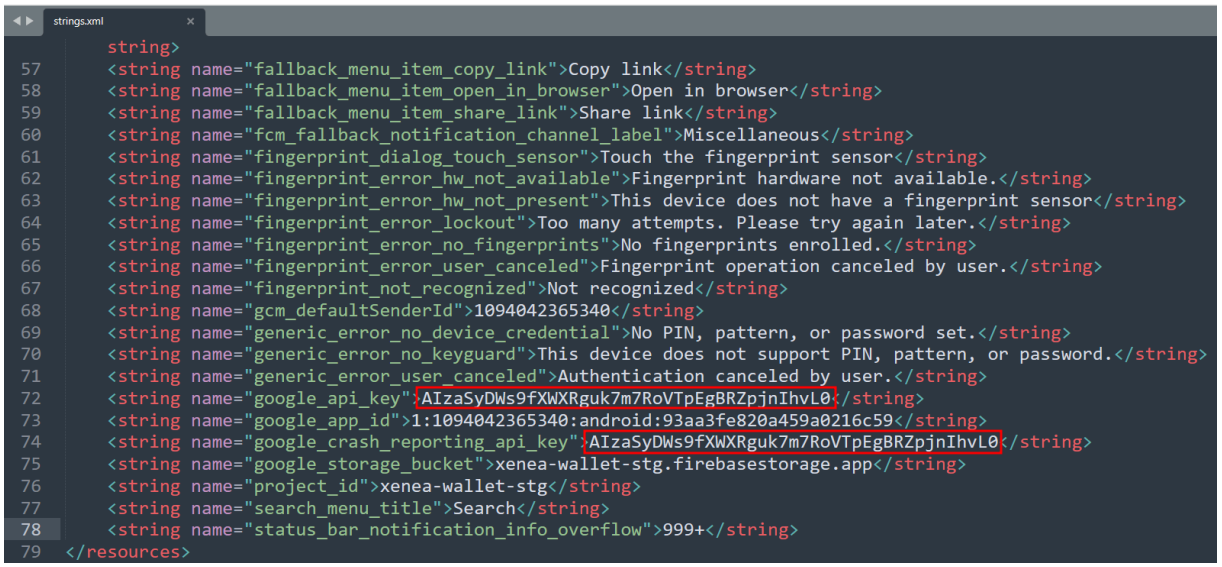
- OWASP Mobile Security Testing Guide - Static Analysis for API Keys
- Google Developer Guide - Keeping your API keys secure
- Android Security Best Practices - Avoid Hardcoding Secrets

Steps to Reproduce

Step 1. Open the apk with jadx-gui and observe the AndroidManifest.yml



Step 2. Open the file string.xml and observe that it contains an API_KEY.



5.17 Insecure Broadcast Receiver Leading to Arbitrary File Write

Vulnerability Severity	**OWASP Category**
Critical [redacted]	Improper Authorization
Tools Used	**Date of reporting**
adb am	11th March 2025

Vulnerability Observation

During testing, it was observed that the application registers an exported broadcast receiver that allows arbitrary file write operations due to improper access control. The broadcast receiver listens for the event com.xenea.wallet.stg.MY_CUSTOM_EVENT, and an attacker can abuse this by sending a crafted ADB command to execute arbitrary commands and write files inside the application's data directory (/data/data/com.xenea.wallet.stg/files/).

Vulnerable location

AndroidMainifest.xml

Technical Impact

Unauthorized apps can exploit the insecure broadcast receiver to perform arbitrary file writes, potentially leading to data tampering, privilege escalation, or remote code execution.

Business impact

This vulnerability can result in data corruption, unauthorized modifications, regulatory non-compliance, reputational damage, and financial losses due to exploitation.

Remediation

If com.xenea.wallet.stg.MY_CUSTOM_EVENT does not exist in AndroidManifest.xml, but can still be triggered, it means the Broadcast Receiver is dynamically registered in the code. To fix this, modify the registration by restricting it to internal use:

- Unregister the dynamically registered receiver if not needed:

context.unregisterReceiver(myBroadcastReceiver);

- If required, register it securely within the app using LocalBroadcastManager:

  LocalBroadcastManager.getInstance(context).registerReceiver(myBroadcastReceiver, new IntentFilter("com.xenea.wallet.stg.MY_CUSTOM_EVENT"));

Status - Acknowledge

References

- Understanding Android Broadcast Receivers
- Local Broadcasts
- Dynamically Registering a Broadcast Receiver
- Sending Alarm Broadcast to a Dynamically Registered Receiver

Steps to Reproduce

1. Run the `adb am` tool to create a file in the Android path `/data/data/com.xenea.wallet.stg/files`. Once the command is executed, navigate to the specified file path to verify that the file has been successfully created with the injected content via the broadcast intent, confirming the vulnerability to arbitrary file write.

```
PS C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools> .\adb.exe -s 127.0.0.1:21533 shell am broadcast -a com.xenea.wallet.stg/com.xenea.wall
et.stg.MY_CUSTOM_EVENT --es "cmd" "echo 'exploit' > /data/data/com.xenea.wallet.stg/files/Write.txt"
PS C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools> .\adb.exe -s 127.0.0.1:21533 shell
ASUS_Z01QD:/ # ls /data/data/com.xenea.wallet.stg/files/
PersistedInstallation.W0RFRkFVTFRd+MToxMDk0MDQyMzY1MzQwOmFuZHJvaWQ6OTNhYTNmZTgyMGE0NTlhMDIxNmM1OQ.json hydrated_box.lock
Write.txt                                                                                                   libCachedImageData.db
datastore                                                                                                   libCachedImageData.db-shm
flutter_callback_cache.json                                                                                 libCachedImageData.db-wal
generatefid.lock                                                                                            profileInstalled
hydrated_box.hive
ASUS_Z01QD:/ # cat /data/data/com.xenea.wallet.stg/files/Write.txt
Broadcasting: Intent { act=com.xenea.wallet.stg/com.xenea.wallet.stg.MY_CUSTOM_EVENT flg=0x400000 pkg=exploit (has extras) }
Broadcast completed: result=0
ASUS_Z01QD:/ #
```

5.18 Sensitive Data Stored in Android Memory.

Vulnerability Severity	OWASP Category
High [redacted]	Insecure Data Storage
Tools Used	**Date of reporting**
fridump3 & sublime text	11th March 2025

Vulnerability Observation

During testing, it was observed that sensitive user information, including the private key, wallet app PIN, email ID, authorization token, and account ID, was stored in the application's memory without proper protection. By dumping the application's memory, it was possible to extract and analyze the data, revealing these sensitive details.

Vulnerable location

Throughout the xenea-wallet.apk application

Technical Impact

An attacker can extract sensitive information, including the account ID, app PIN, authentication token, email ID, and private keys from memory, leading to unauthorized access, account takeover, and data theft.

Business impact

An attacker can extract sensitive user credentials and private keys from memory, leading to unauthorized access, account takeover, and data theft.

Remediation

- The application should clear the memory part after logging out.
- The application should store the credentials in encrypted format.
- Clear the memory area that contains critical data after the operation. This means the application should clear the sensitive data from the memory after lead or FI data is saved. This will ensure that later on a malicious user cannot retrieve any sensitive data from the memory.

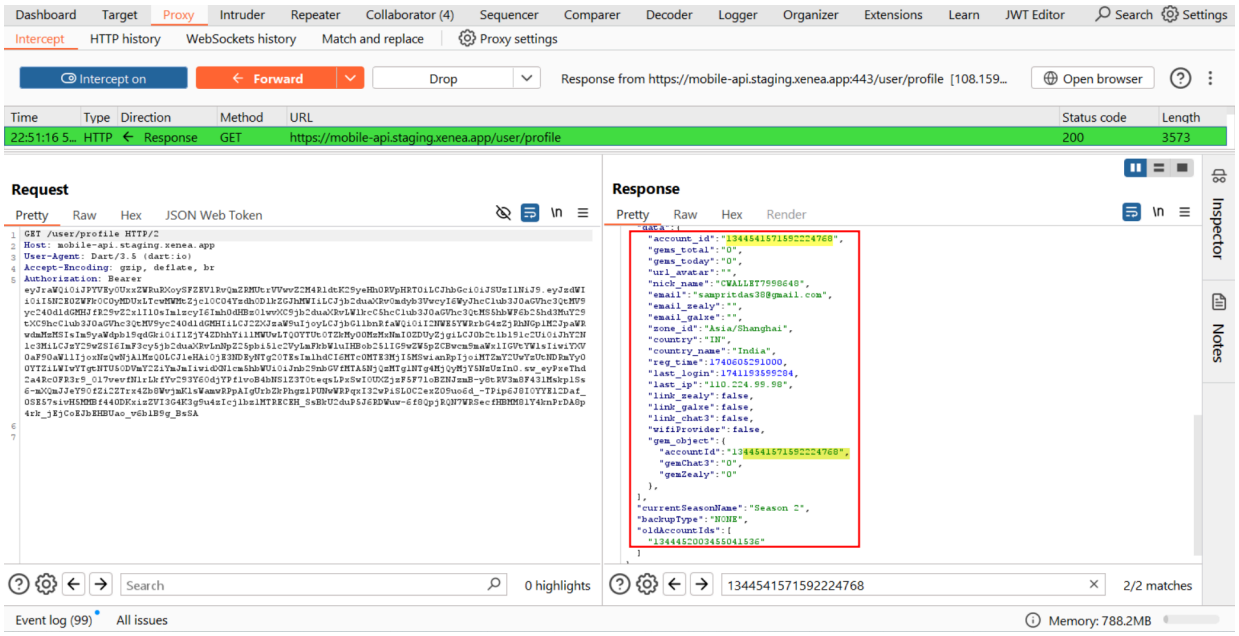
Status - Fixed

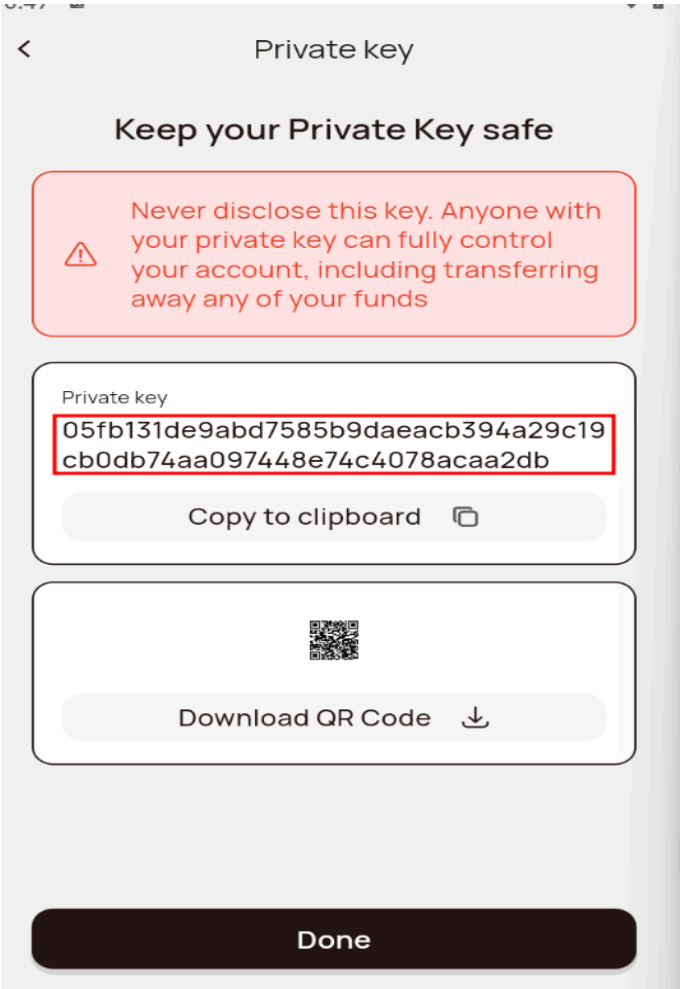
References

- Testing-Data-Storage
- capture-heap-dump
- Dumping Android Application Memory

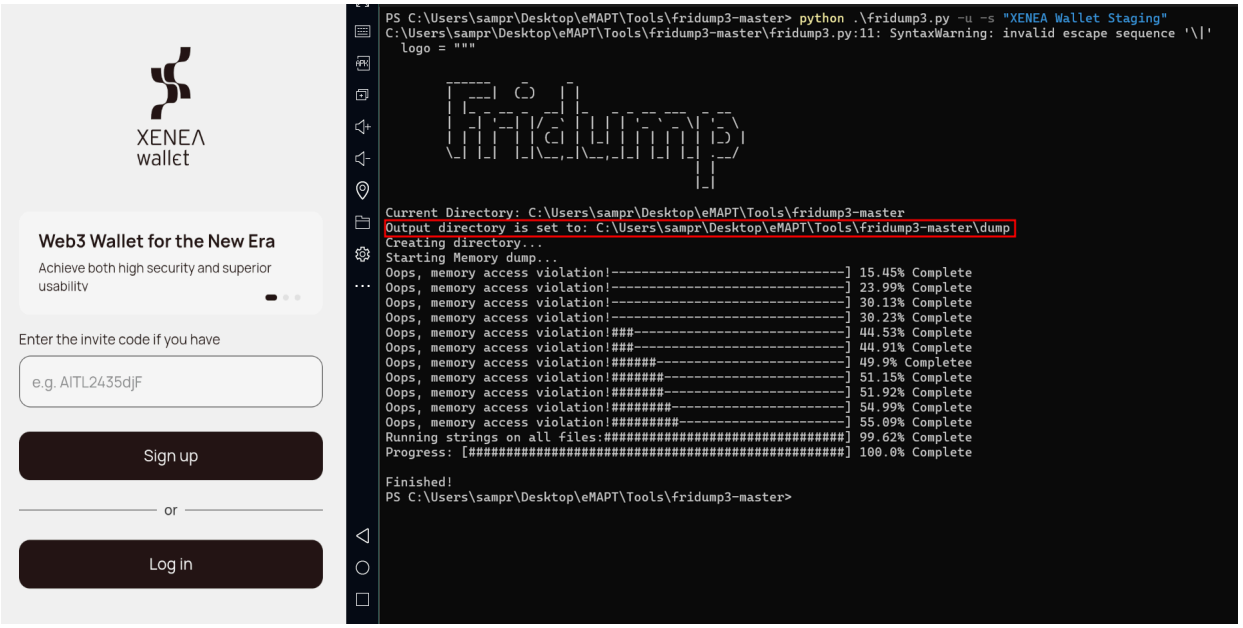
Steps to Reproduce

1. Note the account ID and private key of the wallet account from the screenshots below.

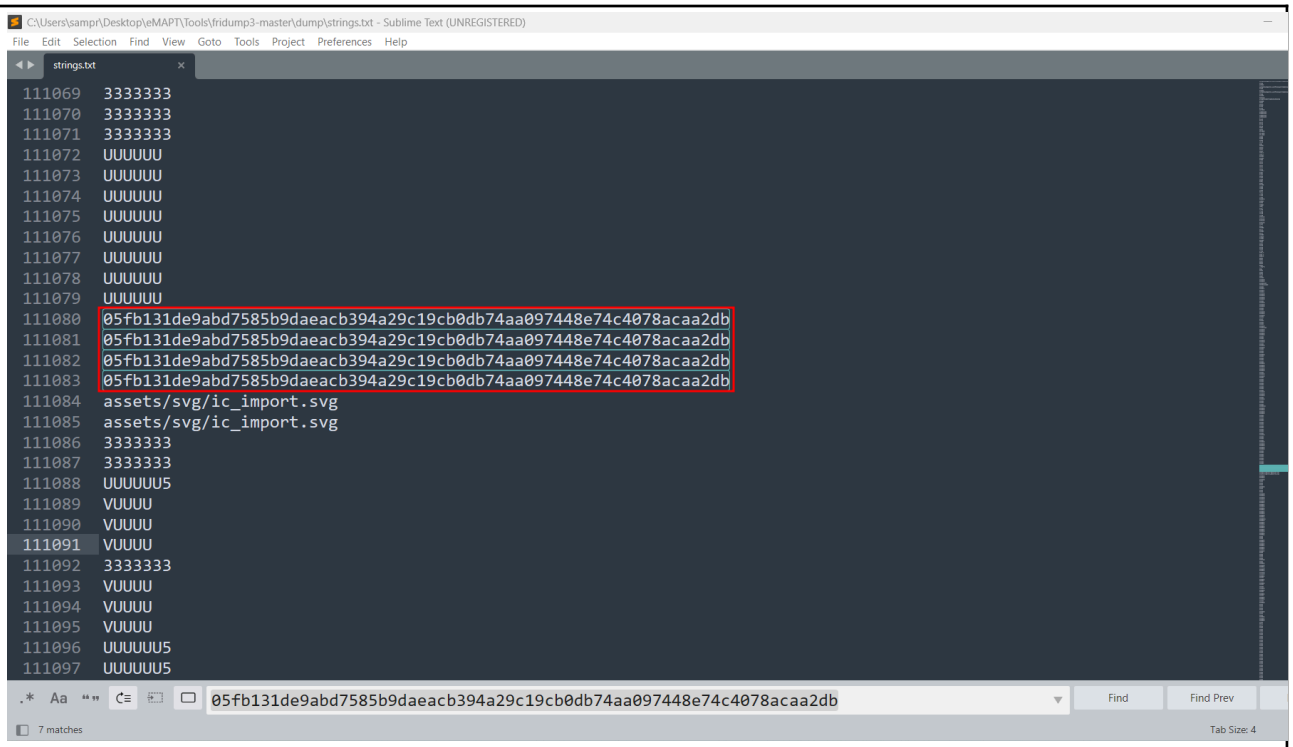




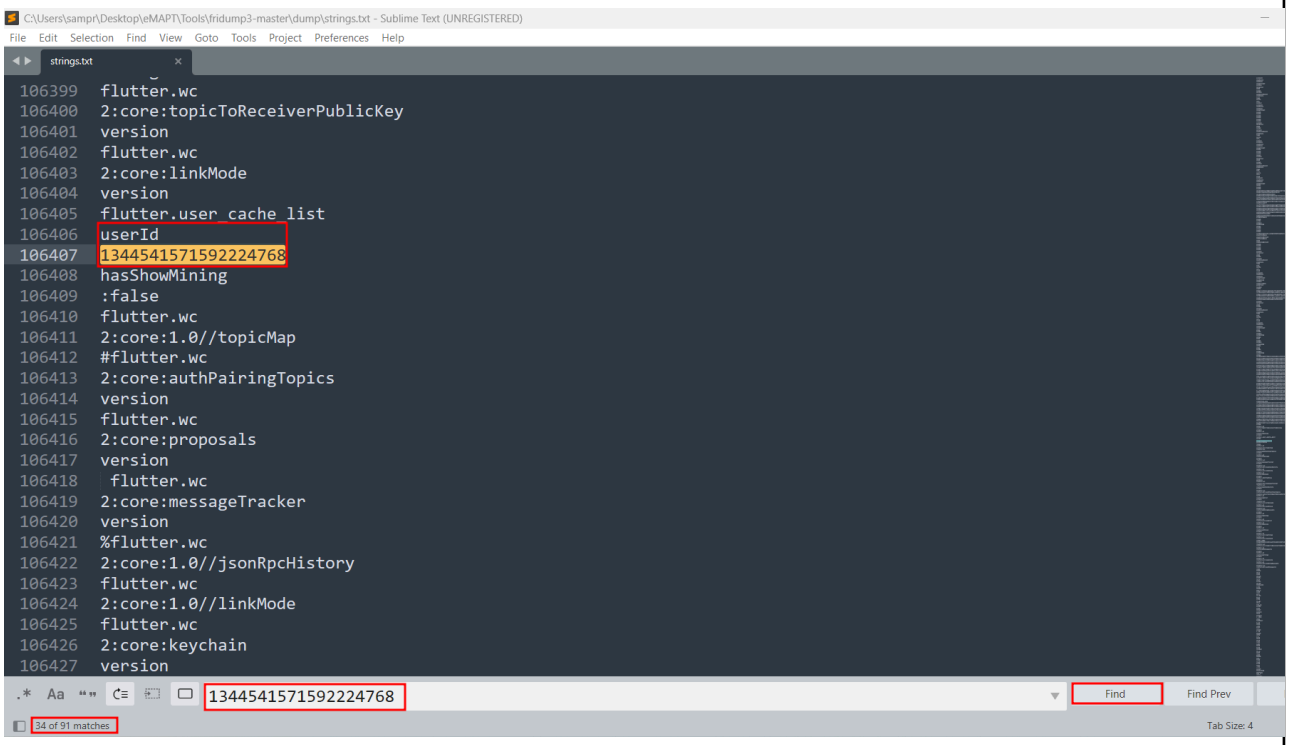
2. Open the application and log out. Run fridump3.py (as shown in the images) to dump the memory of the application process.



3. Open strings.txt using Sublime Text and manually verify the private key stored in memory.



4. Observe the account ID in the file.

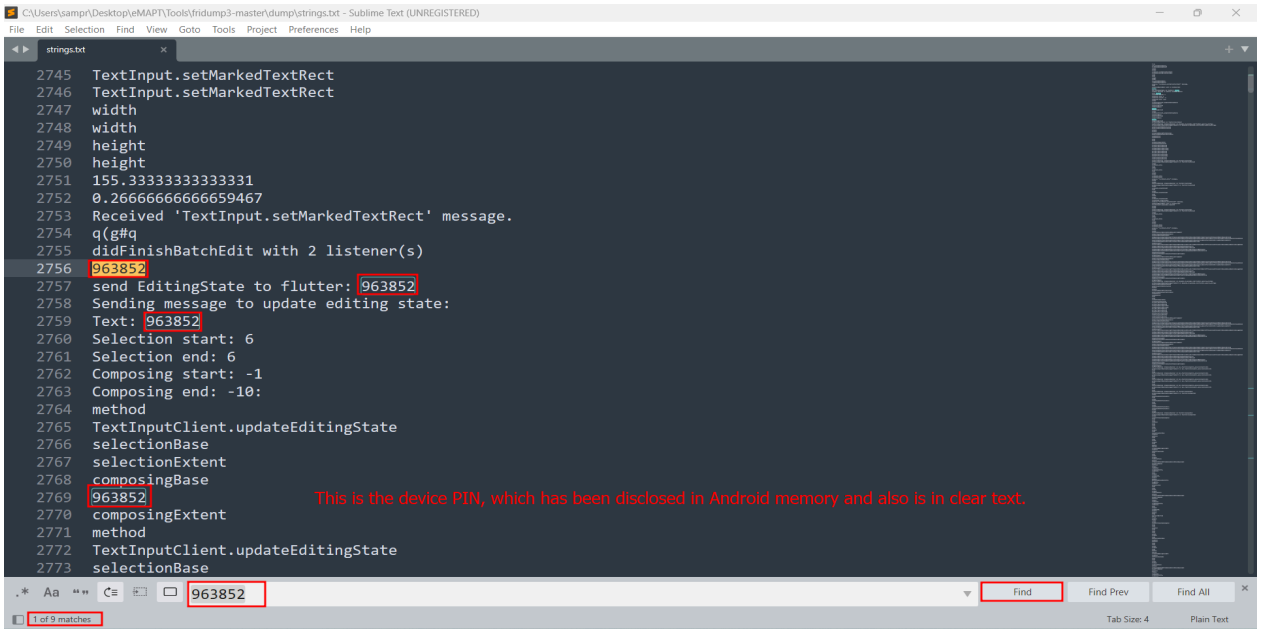


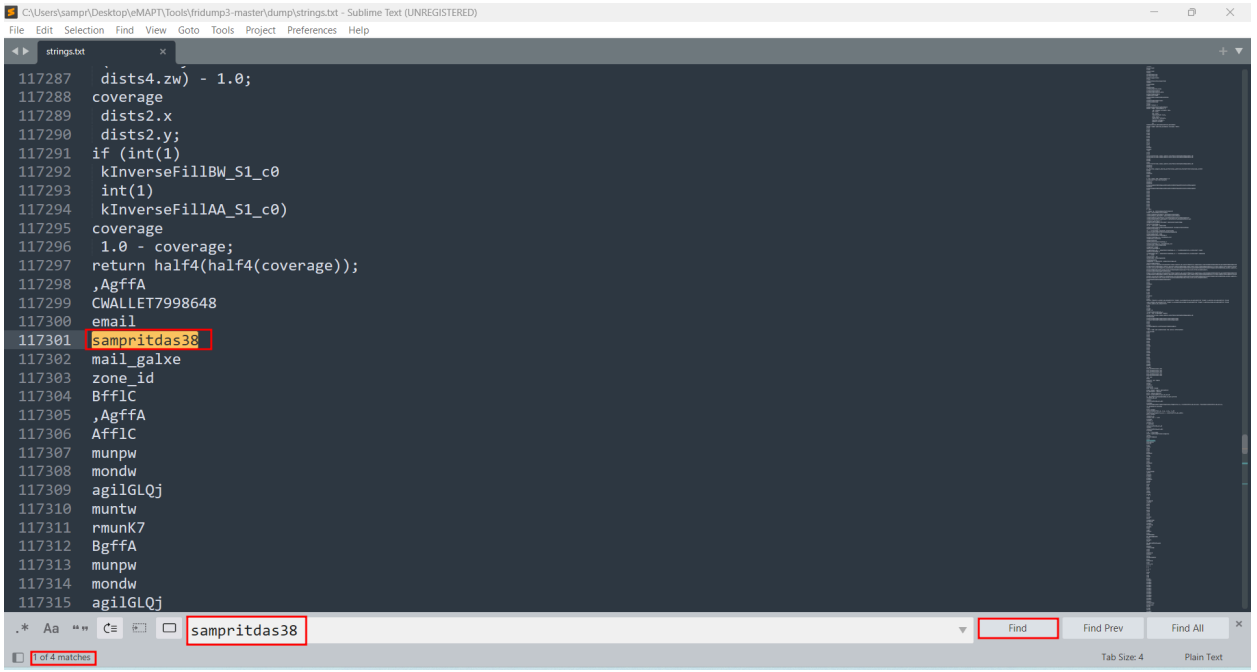
5. The dumped memory also contains:

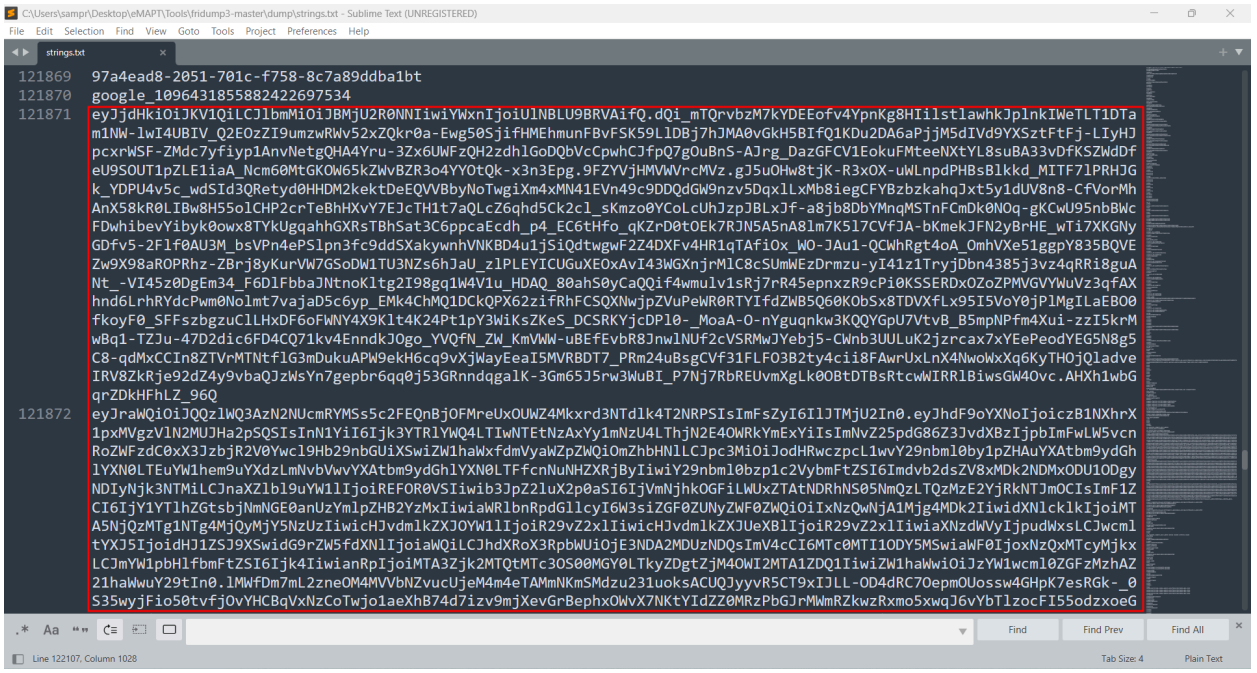
   Wallet app PIN

   User's email ID

   User's authorization token


C:\Users\sampr\Desktop\eMAPT\Tools\fridump3-master\dump\strings.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
strings.txt
2745 TextInput.setMarkedTextRect
2746 TextInput.setMarkedTextRect
2747 width
2748 width
2749 height
2750 height
2751 155.33333333333331
2752 0.26666666666659467
2753 Received 'TextInput.setMarkedTextRect' message.
2754 q(g#q
2755 didFinishBatchEdit with 2 listener(s)
2756 963852
2757 send EditingState to flutter: 963852
2758 Sending message to update editing state:
2759 Text: 963852
2760 Selection start: 6
2761 Selection end: 6
2762 Composing start: -1
2763 Composing end: -10:
2764 method
2765 TextInputClient.updateEditingState
2766 selectionBase
2767 selectionExtent
2768 composingBase
2769 963852
2770 composingExtent
2771 method
2772 TextInputClient.updateEditingState
2773 selectionBase
This is the device PIN, which has been disclosed in Android memory and also is in clear text.
963852
Find
Find Prev
Find All
1 of 9 matches
Tab Size: 4
Plain Text


C:\Users\sampr\Desktop\eMAPT\Tools\fridump3-master\dump\strings.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
strings.txt
117287 dists4.zw) - 1.0;
117288 coverage
117289 dists2.x
117290 dists2.y;
117291 if (int(1)
117292 kInverseFillBW_S1_c0
117293 int(1)
117294 kInverseFillAA_S1_c0)
117295 coverage
117296 1.0 - coverage;
117297 return half4(half4(coverage));
117298 ,AgffA
117299 CWALLET7998648
117300 email
117301 sampritdas38
117302 mail_galxe
117303 zone_id
117304 BfflC
117305 ,AgffA
117306 AfflC
117307 munpw
117308 mondw
117309 agilGLQj
117310 muntw
117311 rmunK7
117312 BgffA
117313 munpw
117314 mondw
117315 agilGLQj
sampritdas38
Find
Find Prev
Find All
1 of 4 matches
Tab Size: 4
Plain Text



5.19 Insecure Random Number Generation Vulnerability in Android App

Vulnerability Severity	**OWASP Category**
Medium	Insufficient Cryptography
Tools Used	**Date of reporting**
jadx-gui	11th March 2025

Vulnerability Observation

We observed that the application uses the insecure random number generator class "java.util.Random". Two instances of the java.util.Random class that is created using the same seed will generate identical sequences of numbers in all Java implementations.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

java.util.Random class using the same seed will generate identical sequences of numbers in all Java implementations. An attacker can learn the value of the seed by performing some reconnaissance on the vulnerable target and can then build a lookup table for estimating future seed values.

Business impact

There is no direct impact on the business, this comes under security best practices.

Remediation

The java.util.Random class must not be used either for security-critical applications or for protecting sensitive data. Use a more secure random number generator, such as the java.security.SecureRandom class.

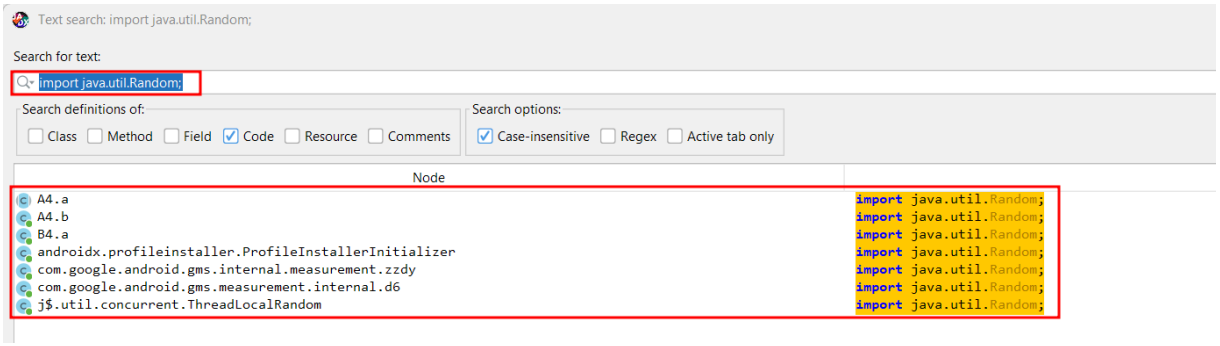
Status - Acknowledge

References

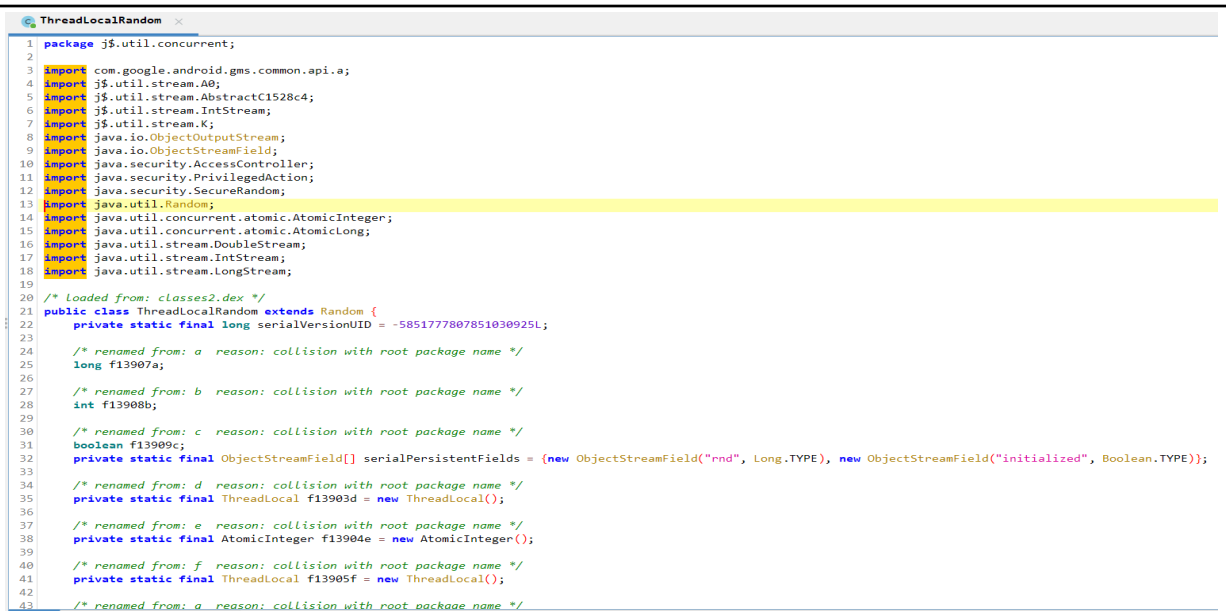
- OWASP Mobile Security Testing Guide (MSTG)
- OWASP Cryptographic Security Guidelines
- Android Security Best Practices
- Java Security Guide (SecureRandom)

Steps to Reproduce

1. Open the apk in jadx-gui application. Search "import java.util.Random" and observe the result.



2. Now open any of the highlighted files and observe that the code contains import java.util.Random functions.



```
ThreadLocalRandom
1  package j$.util.concurrent;
2
3  import com.google.android.gms.common.api.a;
4  import j$.util.stream.A0;
5  import j$.util.stream.AbstractC1528c4;
6  import j$.util.stream.IntStream;
7  import j$.util.stream.K;
8  import java.io.ObjectOutputStream;
9  import java.io.ObjectStreamField;
10 import java.security.AccessController;
11 import java.security.PrivilegedAction;
12 import java.security.SecureRandom;
13 import java.util.Random;
14 import java.util.concurrent.atomic.AtomicInteger;
15 import java.util.concurrent.atomic.AtomicLong;
16 import java.util.stream.DoubleStream;
17 import java.util.stream.IntStream;
18 import java.util.stream.LongStream;
19
20 /* loaded from: classes2.dex */
21 public class ThreadLocalRandom extends Random {
22     private static final long serialVersionUID = -5851777807851030925L;
23
24     /* renamed from: a  reason: collision with root package name */
25     long f13907a;
26
27     /* renamed from: b  reason: collision with root package name */
28     int f13908b;
29
30     /* renamed from: c  reason: collision with root package name */
31     boolean f13909c;
32     private static final ObjectStreamField[] serialPersistentFields = {new ObjectStreamField("rnd", Long.TYPE), new ObjectStreamField("initialized", Boolean.TYPE)};
33
34     /* renamed from: d  reason: collision with root package name */
35     private static final ThreadLocal f13903d = new ThreadLocal();
36
37     /* renamed from: e  reason: collision with root package name */
38     private static final AtomicInteger f13904e = new AtomicInteger();
39
40     /* renamed from: f  reason: collision with root package name */
41     private static final ThreadLocal f13905f = new ThreadLocal();
42
43     /* renamed from: g  reason: collision with root package name */
```

5.20 Unprotected SQLite Database Susceptible to SQL Injection Attacks

Vulnerability Severity	**OWASP Category**
High [redacted]	Insecure Data Storage
Tools Used	**Date of reporting**
jadx-gui	11th March 2025

Vulnerability Observation

During testing, the App uses SQLite Database and executes raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.

Vulnerable location

M.Java

Technical Impact

SQL Injection Vulnerability: Unsanitized user input in raw SQL queries poses a significant security risk, potentially allowing malicious users to manipulate and extract sensitive data from the SQLite database.

Business impact

Using raw SQLite queries can expose the application to SQL injection attacks if user inputs are not properly sanitized. To mitigate this risk, use parameterized queries or prepared statements instead of concatenating user inputs directly into SQL queries. Additionally, implement input validation and least privilege principles to minimize the impact of potential exploitation.

Remediation

Should be encrypted and written to the database.

https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04h-Testing-Code-Quality.md#injection-flaws-mstg-arch-2-and-mstg-platform-2

Status - Acknowledge

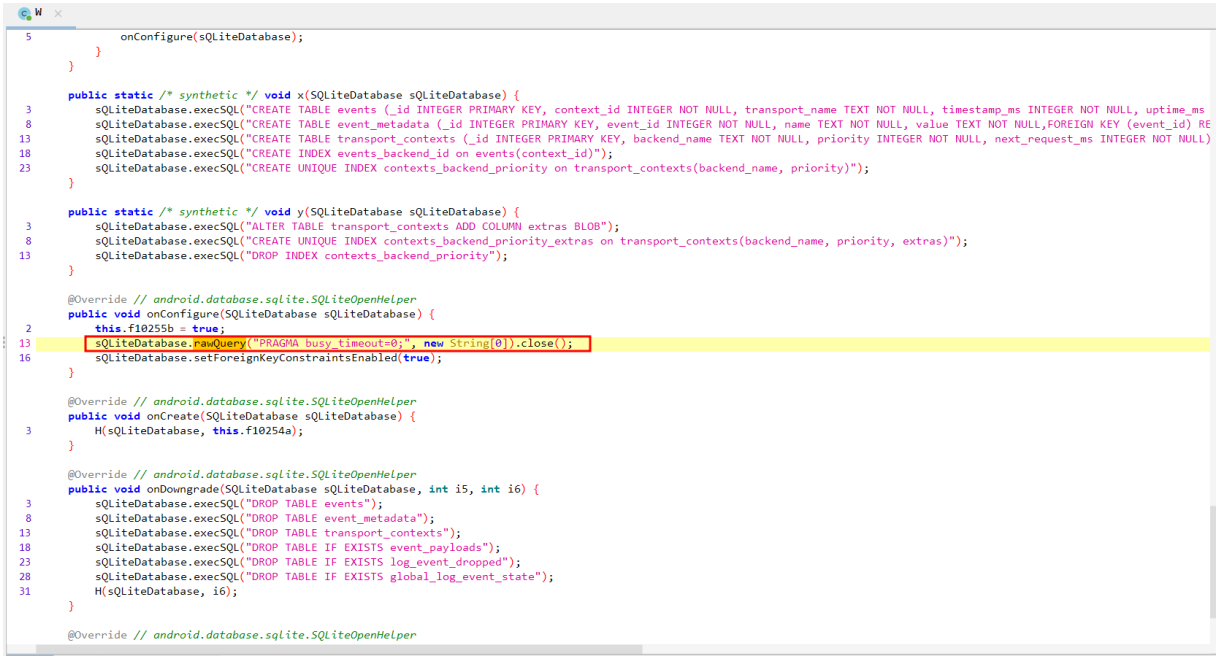
References

SQL Injection Vulnerability: Unsanitized user input in raw SQL queries poses a significant security risk, potentially allowing malicious users to manipulate and extract sensitive data from the SQLite database.

Steps to Reproduce

1. Open the xenea-wallet.apk in JADX-GUI, search for "rawQuery", then navigate to the corresponding M file to observe that it is using raw SQL queries.

```
M
            }
        });
27      sQLiteDatabase.compileStatement("DELETE FROM events WHERE num_attempts >= 16").execute();
 8      return null;
    }

    public static /* synthetic */ Boolean D0(Cursor cursor) {
        boolean z5;
 5      if (cursor.getCount() > 0) {
 7          z5 = true;
        } else {
 9          z5 = false;
        }
10      return Boolean.valueOf(z5);
    }

    public static /* synthetic */ Object E(long j5, V0.p pVar, SQLiteDatabase sQLiteDatabase) {
 1      return F0(j5, pVar, sQLiteDatabase);
    }

    public static /* synthetic */ Object E0(String str, c.b bVar, long j5, SQLiteDatabase sQLiteDatabase) {
35      if (!((Boolean) P0(sQLiteDatabase.rawQuery("SELECT 1 FROM log_event_dropped WHERE log_source = ? AND reason = ?", new String[]{str, Integer.toString(bVar.a())}), new b() {
            @Override // d1.M.b
            public final Object apply(Object obj) {
 3              return M.I((Cursor) obj);
            }
        })).booleanValue()) {
39          ContentValues contentValues = new ContentValues();
44          contentValues.put("log_source", str);
57          contentValues.put("reason", Integer.valueOf(bVar.a()));
66          contentValues.put("events_dropped_count", Long.valueOf(j5));
71          sQLiteDatabase.insert("log_event_dropped", null, contentValues);
        } else {
109         sQLiteDatabase.execSQL("UPDATE log_event_dropped SET events_dropped_count = events_dropped_count + " + j5 + " WHERE log_source = ? AND reason = ?", new String[]{str,
        }
34      return null;
    }

    public static /* synthetic */ byte[] F(Cursor cursor) {
 1      return A0(cursor);
    }
```



```
W
 5          onConfigure(sQLiteDatabase);
        }
    }

    public static /* synthetic */ void x(SQLiteDatabase sQLiteDatabase) {
 3      sQLiteDatabase.execSQL("CREATE TABLE events (_id INTEGER PRIMARY KEY, context_id INTEGER NOT NULL, transport_name TEXT NOT NULL, timestamp_ms INTEGER NOT NULL, uptime_ms
 8      sQLiteDatabase.execSQL("CREATE TABLE event_metadata (_id INTEGER PRIMARY KEY, event_id INTEGER NOT NULL, name TEXT NOT NULL, value TEXT NOT NULL,FOREIGN KEY (event_id) RE
13      sQLiteDatabase.execSQL("CREATE TABLE transport_contexts (_id INTEGER PRIMARY KEY, backend_name TEXT NOT NULL, priority INTEGER NOT NULL, next_request_ms INTEGER NOT NULL)
18      sQLiteDatabase.execSQL("CREATE INDEX events_backend_id on events(context_id)");
23      sQLiteDatabase.execSQL("CREATE UNIQUE INDEX contexts_backend_priority on transport_contexts(backend_name, priority)");
    }

    public static /* synthetic */ void y(SQLiteDatabase sQLiteDatabase) {
 3      sQLiteDatabase.execSQL("ALTER TABLE transport_contexts ADD COLUMN extras BLOB");
 8      sQLiteDatabase.execSQL("CREATE UNIQUE INDEX contexts_backend_priority_extras on transport_contexts(backend_name, priority, extras)");
13      sQLiteDatabase.execSQL("DROP INDEX contexts_backend_priority");
    }

    @Override // android.database.sqlite.SQLiteOpenHelper
    public void onConfigure(SQLiteDatabase sQLiteDatabase) {
 2      this.f10255b = true;
13      sQLiteDatabase.rawQuery("PRAGMA busy_timeout=0;", new String[0]).close();
16      sQLiteDatabase.setForeignKeyConstraintsEnabled(true);
    }

    @Override // android.database.sqlite.SQLiteOpenHelper
    public void onCreate(SQLiteDatabase sQLiteDatabase) {
 3      H(sQLiteDatabase, this.f10254a);
    }

    @Override // android.database.sqlite.SQLiteOpenHelper
    public void onDowngrade(SQLiteDatabase sQLiteDatabase, int i5, int i6) {
 3      sQLiteDatabase.execSQL("DROP TABLE events");
 8      sQLiteDatabase.execSQL("DROP TABLE event_metadata");
13      sQLiteDatabase.execSQL("DROP TABLE transport_contexts");
18      sQLiteDatabase.execSQL("DROP TABLE IF EXISTS event_payloads");
23      sQLiteDatabase.execSQL("DROP TABLE IF EXISTS log_event_dropped");
28      sQLiteDatabase.execSQL("DROP TABLE IF EXISTS global_log_event_state");
31      H(sQLiteDatabase, i6);
    }

    @Override // android.database.sqlite.SQLiteOpenHelper
```

5.21 Remote Access Tool Detection not properly implemented

Vulnerability Severity	OWASP Category

Low	Insecure Communication
Tools Used	**Date of reporting**
Anydesk & Memu	11th March 2025

Vulnerability Observation

During testing, we observed that the application's detection for remote access and screen sharing is not properly implemented. This allows an attacker to potentially perform actions by guessing the application's graphical user interface (GUI).

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

A successful attack could lead to data breaches, impacting sensitive user information. The business may face legal consequences, compliance issues, and a decline in user confidence, potentially affecting customer retention and acquisition..

Business impact

If remote access tool detection is not properly implemented in an application, attackers can gain unauthorized access and control over the app or device. This can lead to data theft, manipulation, or complete device takeover. Malicious actors may exploit this vulnerability for surveillance, deploying malware, or stealing sensitive information. Inadequate detection puts user privacy and app integrity at significant risk, compromising security and trust.

Remediation

To remediate improper remote access tool (RAT) detection, implement robust monitoring and detection mechanisms to identify suspicious activities or unauthorized remote access attempts. Utilize security software and tools designed to detect RATs and continuously scan for unusual patterns. Ensure proper access control measures are in place, like strong authentication and encryption. Regularly update the app’s security features, and implement proactive measures such as anti-tampering and root detection to prevent unauthorized access.

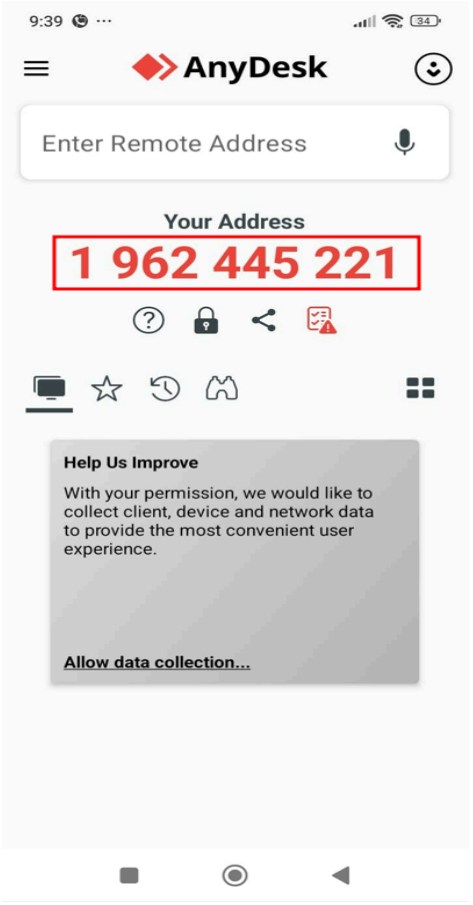
Status - Fixed

References

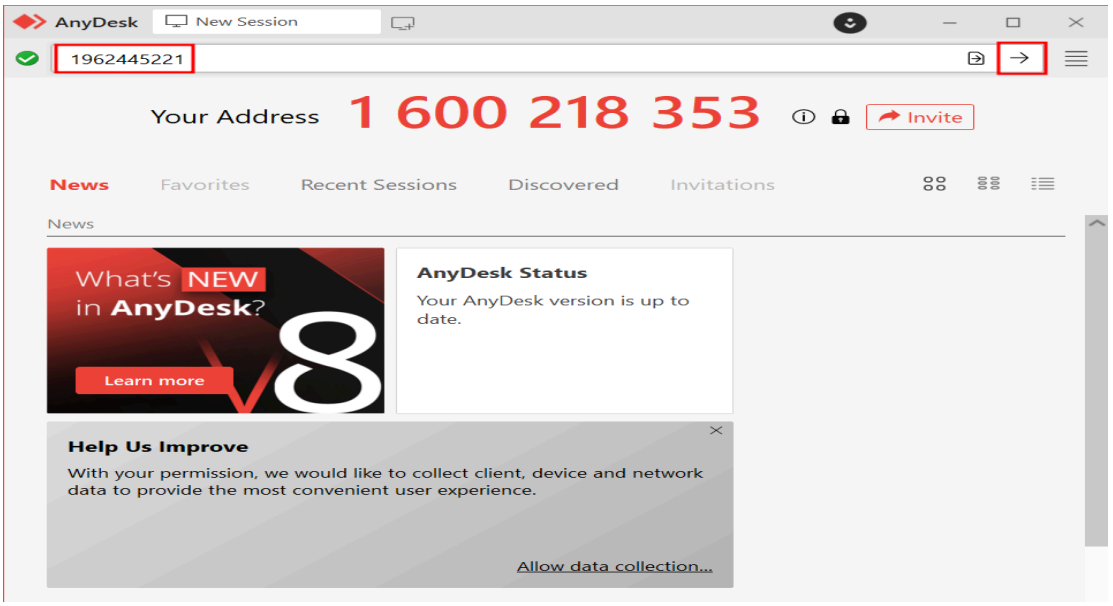
- Stack Overflow Discussion on Detecting Remote Applications in Android Apps

Steps to Reproduce

1. Open the AnyDesk application from an android device.

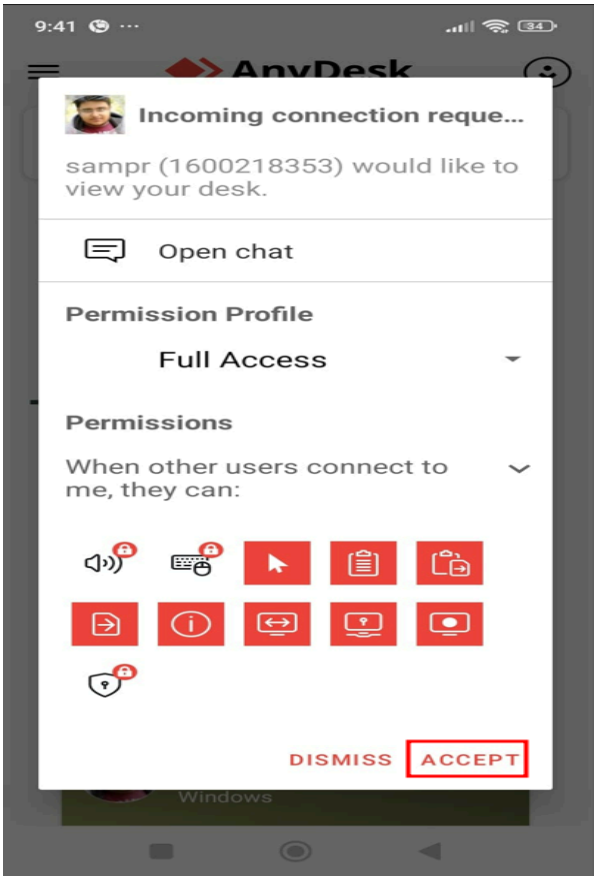


2. Now paste the android device address.

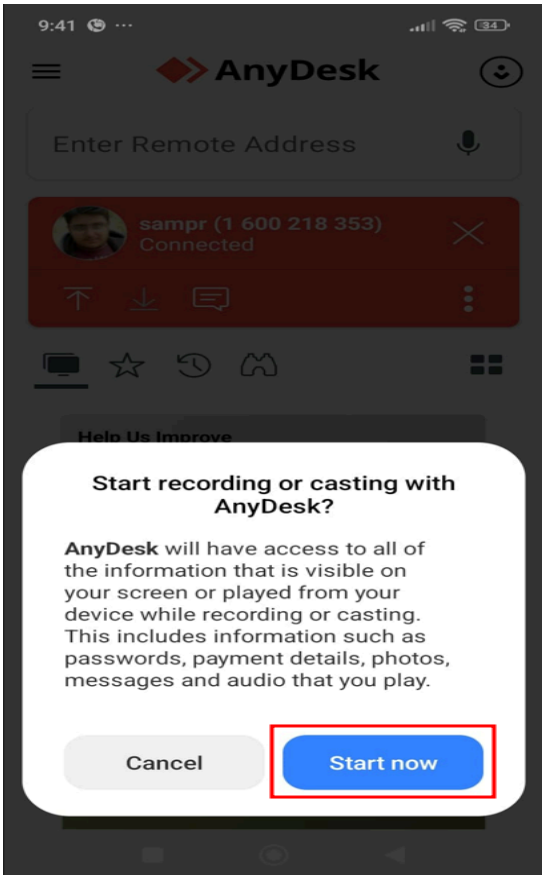


3. Then click on "Start now".

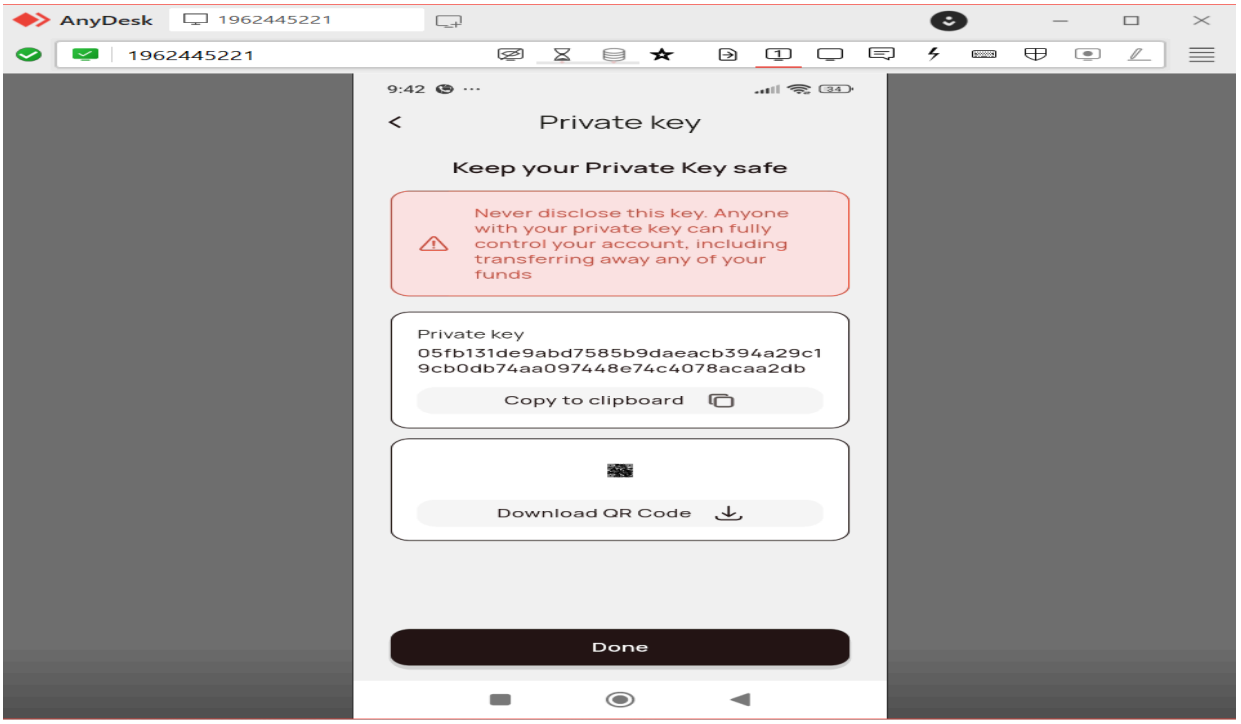
4. Now open the application.



5. Then click on "Start now".



6. Now, open the application and navigate to the Private Key page. Observe the screen through AnyDesk and confirm that the private key is visible remotely.



5.22 Unlocked and Reassigned: Bypassing App Integrity

Vulnerability Severity	OWASP Category
High [redacted]	Insecure Communication
Tools Used	**Date of reporting**
apktool, uber-apk-signer & Memu	11th March 2025

Vulnerability Observation

During testing, it was observed that the application lacks integrity checks. After reversing the APK and modifying the code, attackers can re-sign the application and use it without any restrictions, potentially leading to unauthorized modifications and exploitation.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

Lack of integrity checks allows attackers to modify and re-sign the application, leading to unauthorized code execution, bypassing security mechanisms, and potential malware injection.

Business Impact

Unauthorized modifications can result in data breaches, financial fraud, reputational damage, and loss of user trust.

Remediation

Developers should implement Integrity checks so when the code of an application is modified and resigned again by an attacker, the application should not build again or not run. The app should detect the modification and respond in some way. At the very least, the app should alert the user and/or terminate. Developers can use in-app temper detection or can use CRC mechanism on the app byte-code, Verify apk sign signature etc.

Status - Fixed

References

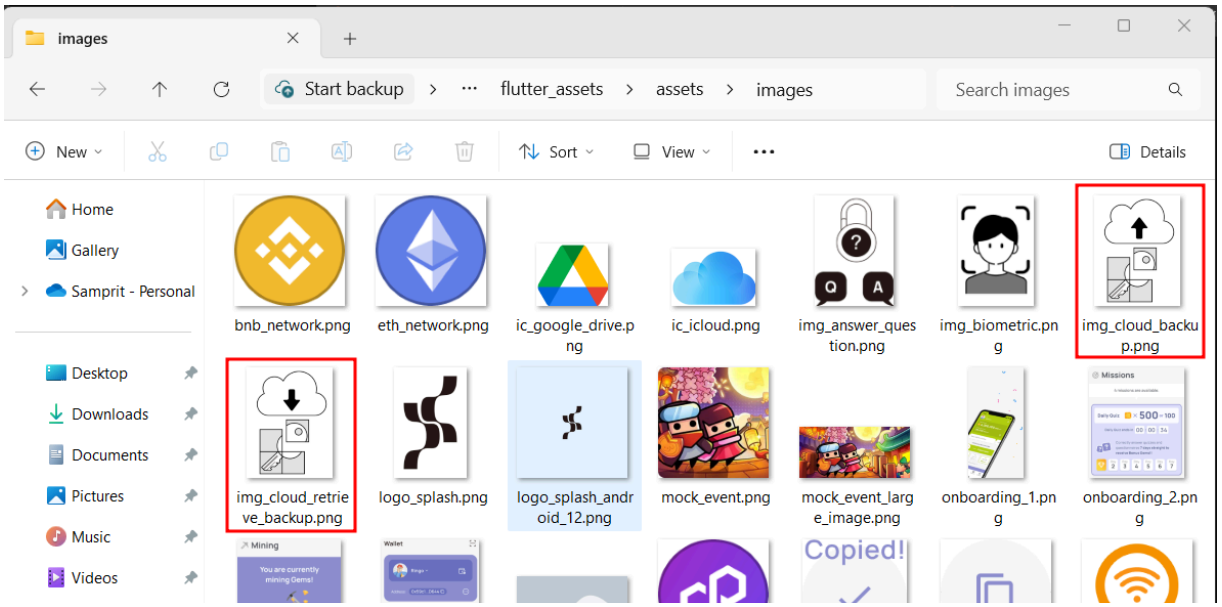
- Android Tamper Detector

Steps to Reproduce

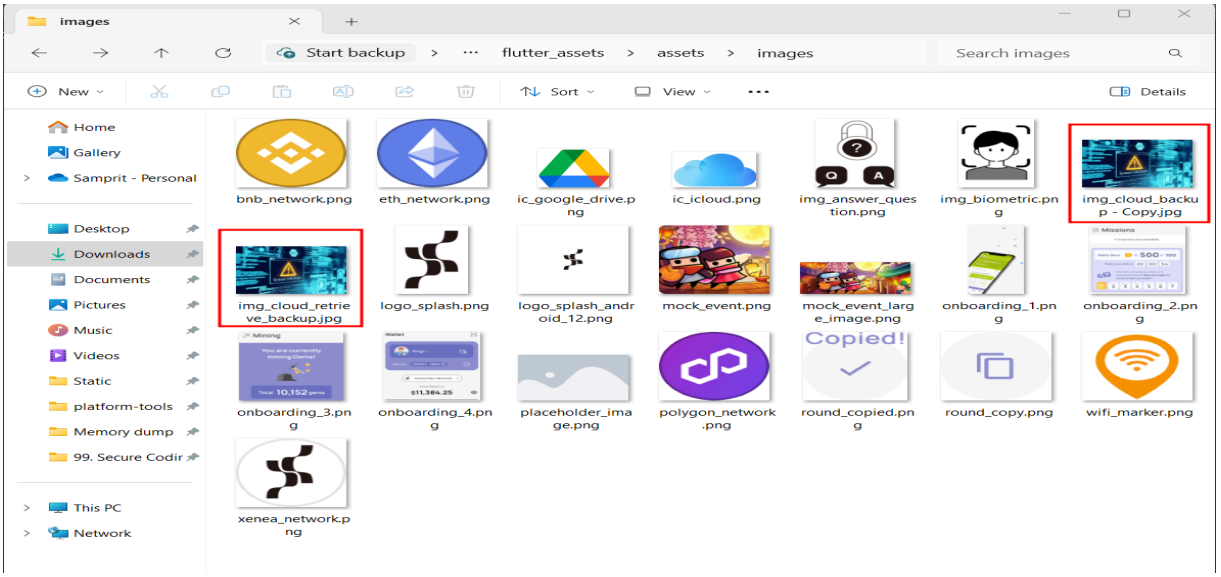
1. Run command "apktool d app-debug.apk" in the terminal to decompile the application.

```
PS C:\Users\sampr\Desktop\eMAPT\Tools> java -jar .\apktool_2.11.0.jar d .\xenea-wallet.apk
I: Using Apktool 2.11.0 on xenea-wallet.apk with 8 threads
I: Baksmaling classes.dex...
I: Loading resource table...
I: Baksmaling classes3.dex...
I: Baksmaling classes2.dex...
I: Decoding file-resources...
I: Loading resource table from file: C:\Users\sampr\AppData\Local\apktool\framework\1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Copying original files...
I: Copying assets...
I: Copying lib...
I: Copying kotlin...
I: Copying META-INF/services...
I: Copying unknown files...
PS C:\Users\sampr\Desktop\eMAPT\Tools> dir


    Directory: C:\Users\sampr\Desktop\eMAPT\Tools


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        24-02-2025     19:24                dex-tools-v2.4
d-----        06-03-2025     02:47                fridump3-master
d-----        27-02-2025     00:40                Mobile-Security-Framework-MobSF-4.3.0
d-----        06-03-2025     02:38                platform-tools
d-----        22-07-2024     22:33                scrcpy-win64-v2.5
d-----        06-03-2025     20:31                xenea-wallet
-a----        24-02-2025     19:17       24480507 apktool_2.11.0.jar
-a----        24-02-2025     19:24          80384 jadx-gui-1.5.1.exe
-a----        24-02-2025     19:22        1608662 jd-gui.exe
-a----        18-07-2024     20:44          14404 maps_api_scanner.py
-a----        28-02-2025     21:58      116061941 release.RE-aligned-debugSigned.apk
-a----        28-02-2025     21:58         923130 release.RE-aligned-debugSigned.apk.idsig
-a----        28-02-2025     21:58      115947941 release.RE.apk
-a----        26-02-2025     20:32        3208194 uber-apk-signer-1.3.0.jar
-a----        26-02-2025     17:38      116459128 xenea-wallet.apk
```

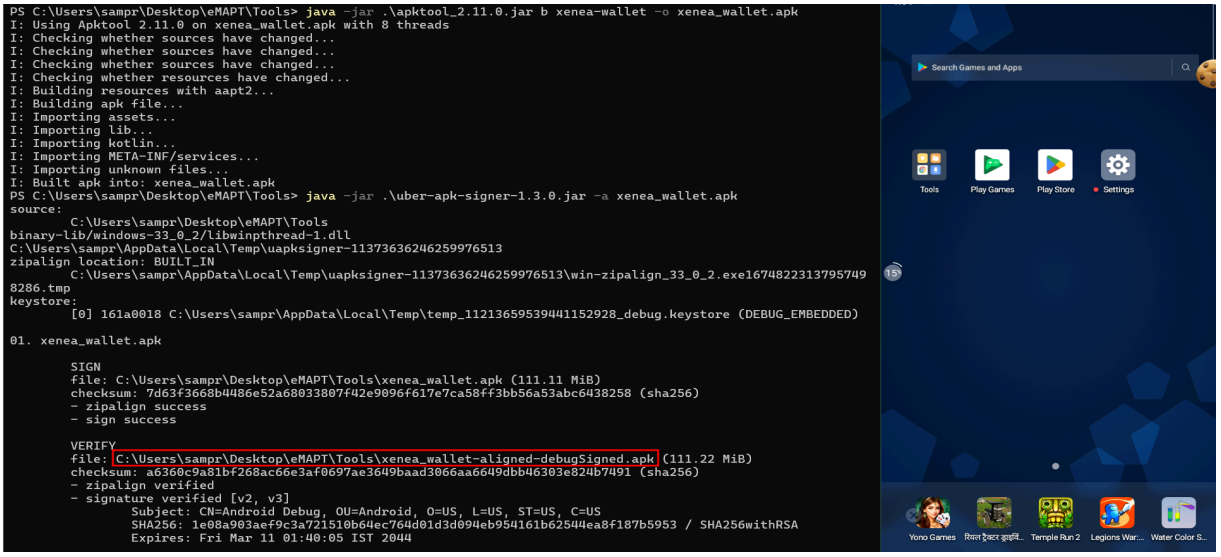
2. Navigate to the "flutter_assets/assets/images" and observe the highlighted images



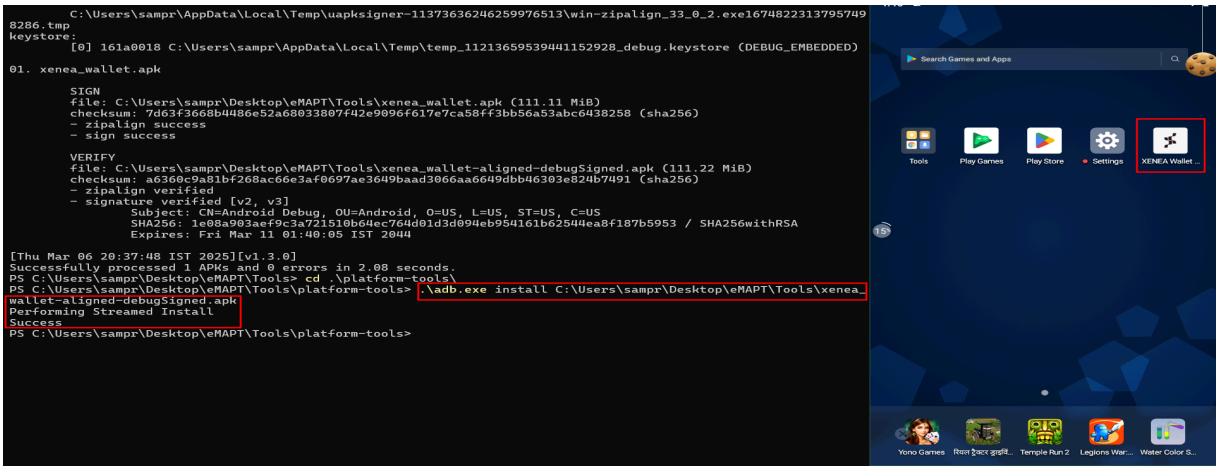
3. Now replace it with another image.



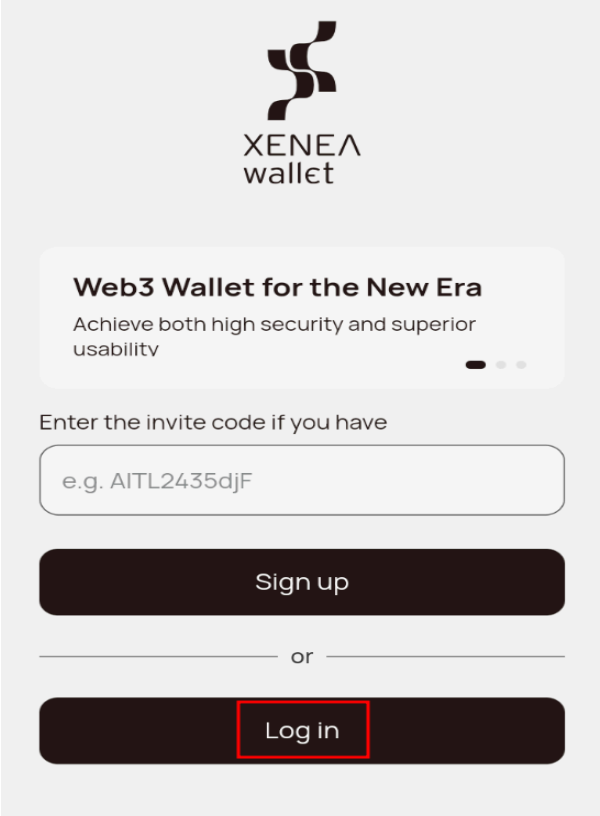
4. Run command "apktool b app-debug/ -o kotak.apk" in the terminal to build the apk file and Sign the application.



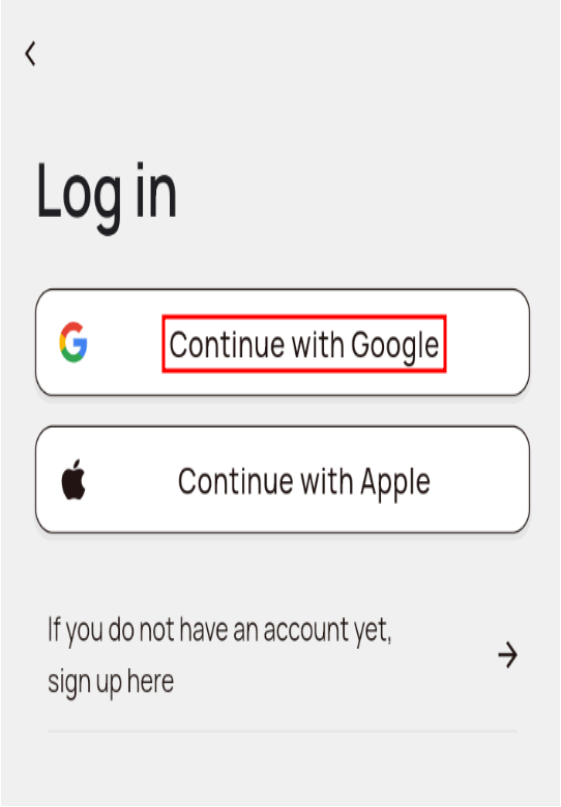
5. Now install the apk with adb.



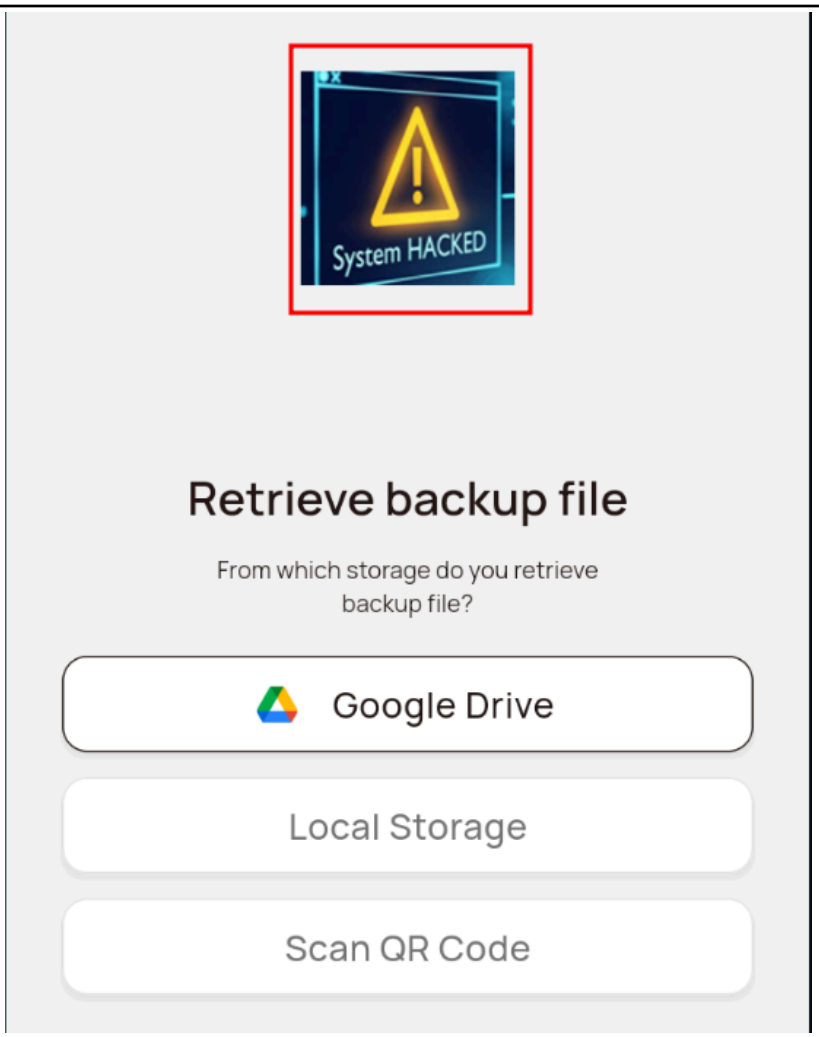
6. Click on login



7. Login with google.



8. Observe the application is running after the modification. Observe the highlighted part.



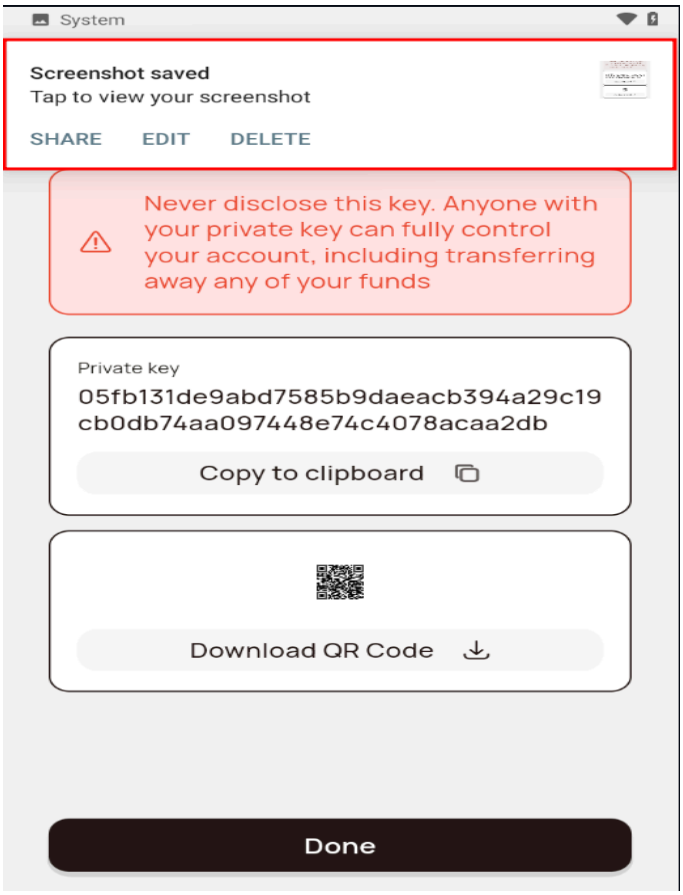
5.23 Sensitive Data leakage via application screenshot (Screen Caching)

Vulnerability Severity	**OWASP Category**
Low	Insecure Data Storage
Tools Used	**Date of reporting**
Memu	11th March 2025

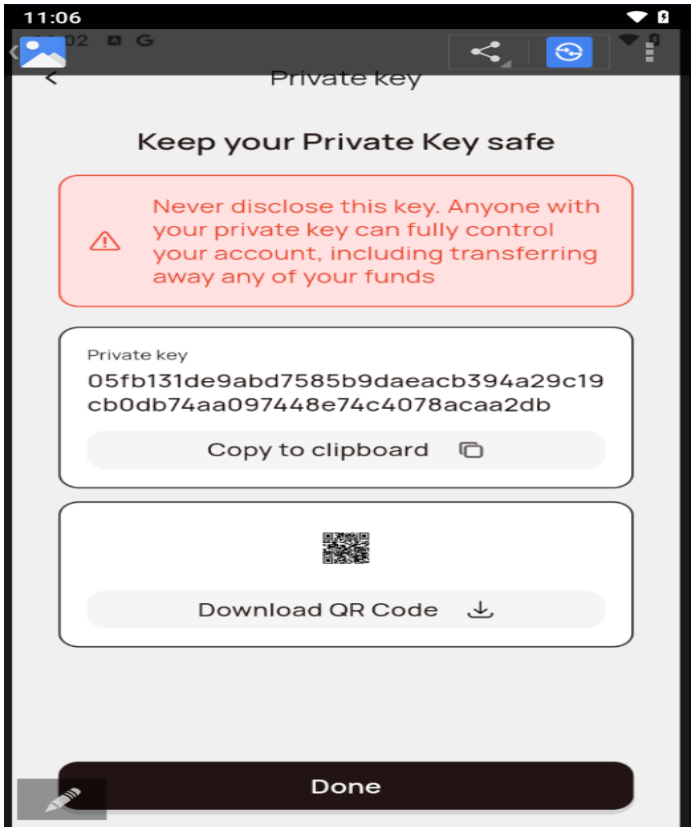
Vulnerability Observation
During the analysis, it was observed that the application allowed users to capture screenshots. Other Malicious applications, users having access to the device could get hold of these screenshots and can leverage this situation threatening data leak.
Vulnerable location
The vulnerability is present throughout the xenea-wallet.apk
Technical Impact
Malicious applications can have access to the photo gallery of the user where the screenshots get saved. Data leakage has several consequences such as compromise in privacy of customer, monetary losses to client, lawsuit against client etc.

Business impact
The exploitation of this vulnerability will require a significant amount of work and time and does not directly affect the organization. Although, it is considered a security best practice to keep the users and their data safe from malicious actors.
Remediation
To prevent sensitive data leakage via screenshots or screen caching, use the FLAG_SECURE flag in the app's activities to block screenshot and screen recording functionality. Avoid displaying sensitive information in notifications or status bars. Additionally, minimize the storage of sensitive data in memory and ensure it’s encrypted both in transit and at rest. Regularly audit the app for potential vulnerabilities related to data exposure and implement secure coding practices.
Status - Fixed
References
• Detect Screenshots in Android • Screenshot detection while user using your app
Steps to Reproduce

1. Open the application and take a screenshot. Note that the application allows taking screenshots.



2. Now open the screenshot.



5.24 Sensitive information disclosure through auto-generated screenshot

Vulnerability Severity	**OWASP Category**
Medium	Insecure Data Storage
Tools Used	**Date of reporting**
Memu & adb	11th March 2025

Vulnerability Observation

It is observed when the user puts the application in background mode after adding sensitive information and captures screen-shot and observes that screenshot which contains all the sensitive information.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

Sensitive information displayed on the screen will be captured in the screenshot and it may disclose sensitive information about the application or application user. So a successful attack of this vulnerability leads to loss of confidentiality.

Business impact

This can affect the customers if any sensitive data is leaked and harm the reputation of the organization.

Remediation

- Developer can implement "FLAG_SECURE" option in application which is something similar to the following code:-

getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,WindowManager.LayoutParams.FLAG_ SECURE);

setContentView(R.layout.activity_main);

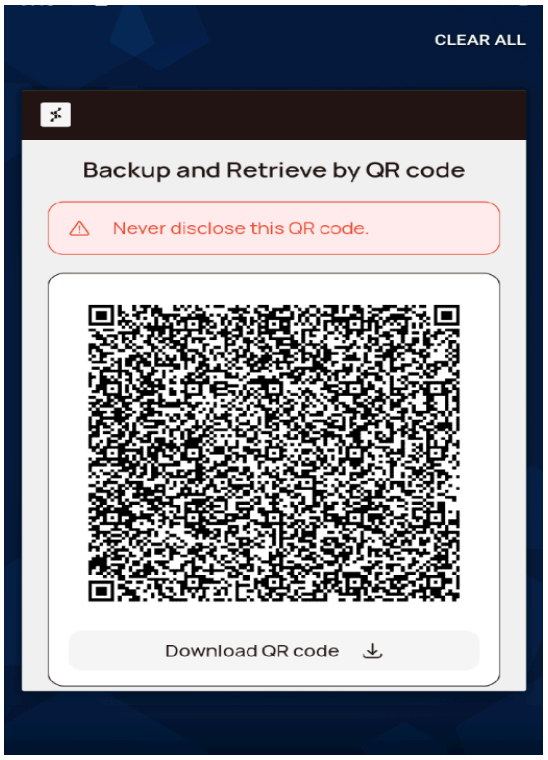
Status - Fixed

References

- Android Data Storage

Steps to Reproduce

1. Open application enter the username and password, now send the application in the background.

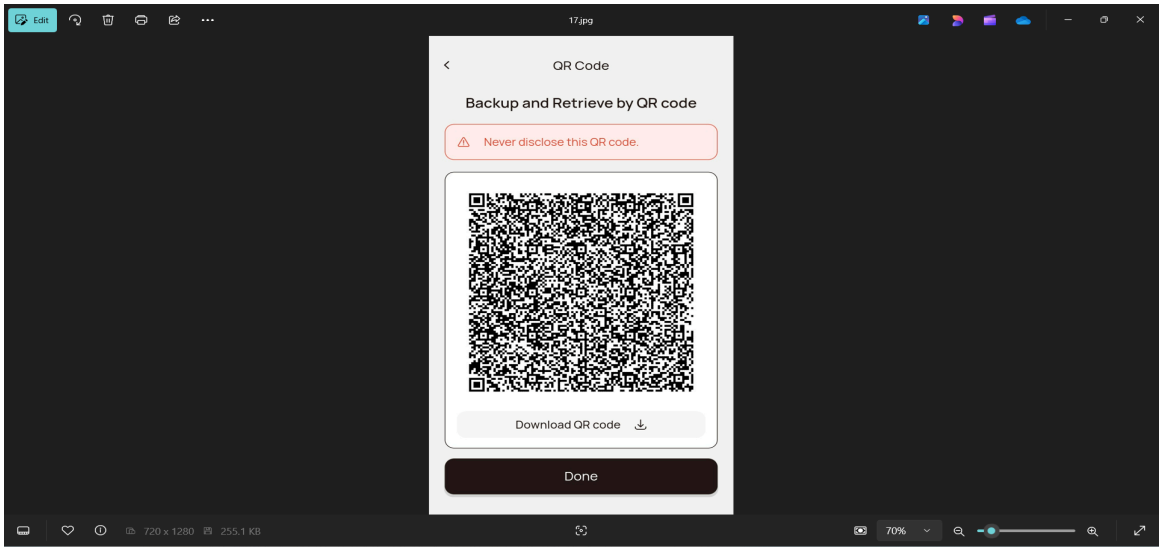


2. Observe the snapshots folder. Run the following command to pull the snapshots folder.

   Command: adb pull /data/system_ce/0/snapshots

```
C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools>adb -s 127.0.0.1:21523 shell
ASUS_Z01QD:/ # cd /data/system_ce/0/snapshots
ASUS_Z01QD:/data/system_ce/0/snapshots # ls
17.jpg 17.proto 17_reduced.jpg
ASUS_Z01QD:/data/system_ce/0/snapshots # cp 17.jpg /sdcard/
ASUS_Z01QD:/data/system_ce/0/snapshots # exit

C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools>adb -s 127.0.0.1:21523 pull /sdcard/17.jpg
/sdcard/17.jpg: 1 file pulled, 0 skipped. 4.0 MB/s (261245 bytes in 0.063s)
```

3. Observe the auto-generated screenshot disclose the sensitive information.



5.25 Sensitive Data Stored Unsecured in Local Database Exposes Privacy Risks

Vulnerability Severity	**OWASP Category**
Critical [redacted]	Insecure Data Storage
Tools Used	**Date of reporting**
adb	11th March 2025

Vulnerability Observation

During testing, it was observed that sensitive information, including the private key, authentication token, access token, user account ID, email ID, and IP address, is being stored in the Android local storage without proper security measures.

Vulnerable location

PersistedInstallation.W0RFRkFVTFRd+MToxMDQ1MTIyNTYwMjI5OmFuZHJvaWQ6Mzk1OWYxMzY0MzlhNmI5ODlmNjQwYw.json
hydrated_box.hive
libCachedImageData.db-wal
com.xena.wallet.stg/cache/*

Technical Impact

Exposure of sensitive data stored in local storage can lead to unauthorized access, credential theft, and account takeover.

Business Impact

Compromised user data can result in financial loss, reputational damage, and non-compliance with data protection regulations.

Remediation

Encrypt sensitive data before storing it locally and use Android's EncryptedSharedPreferences or Keystore for secure storage.

Status - Fixed

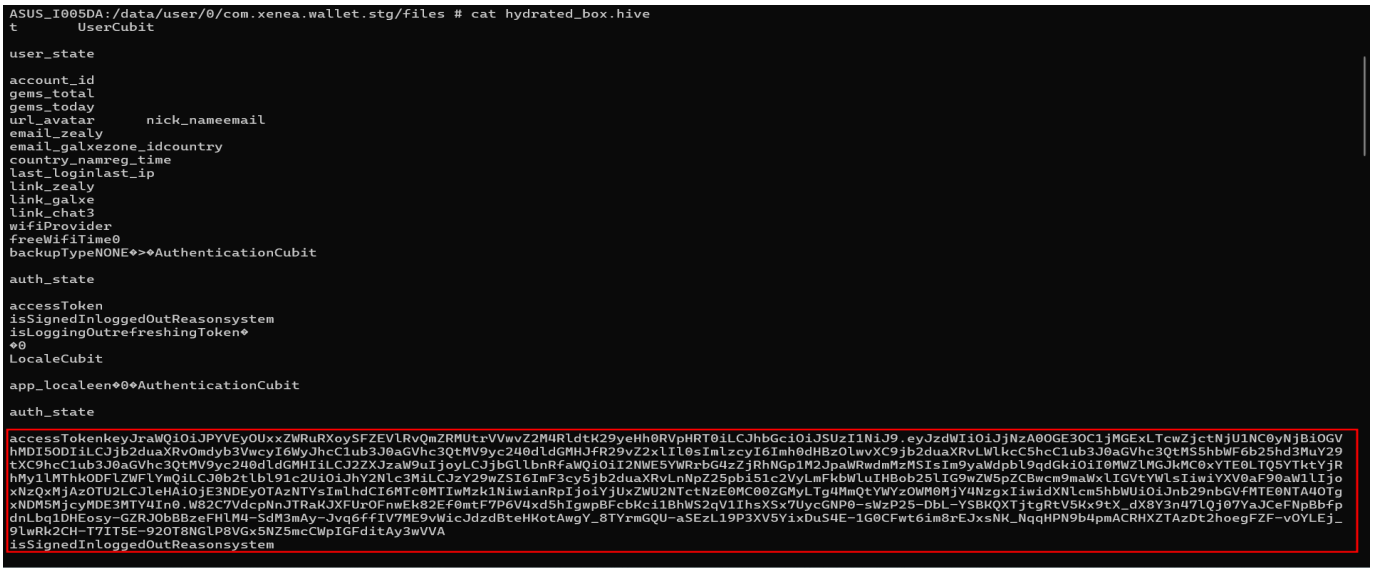
References

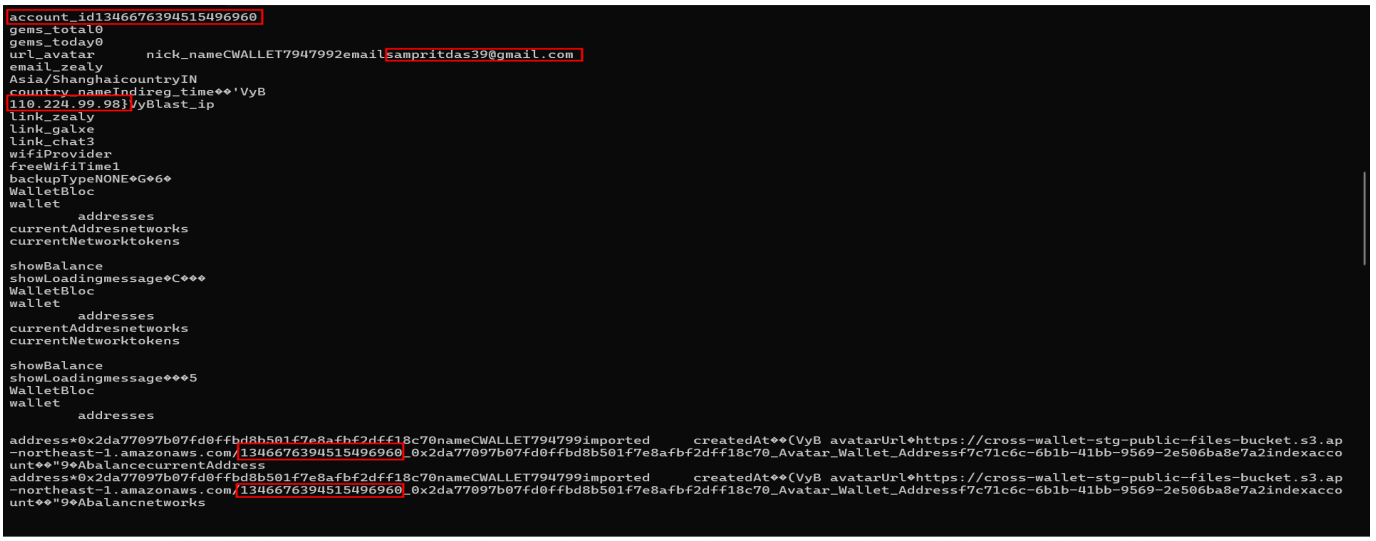
- [Android Data Storage]

Steps to Reproduce

1. First logout the account from the app and navigate to local storage path and observe the PersistedInstallation.W0RFRkFVTFRd+MToxMDQ1MTIyNTYwMjI5OmFuZHJvaWQ6Mzk1OWYxMzY0MzlhNmI5ODlmNjQwYw.json and observe the content.

2. cat hydrated_box.hive and observe the access token, user account id, email id & ip address.



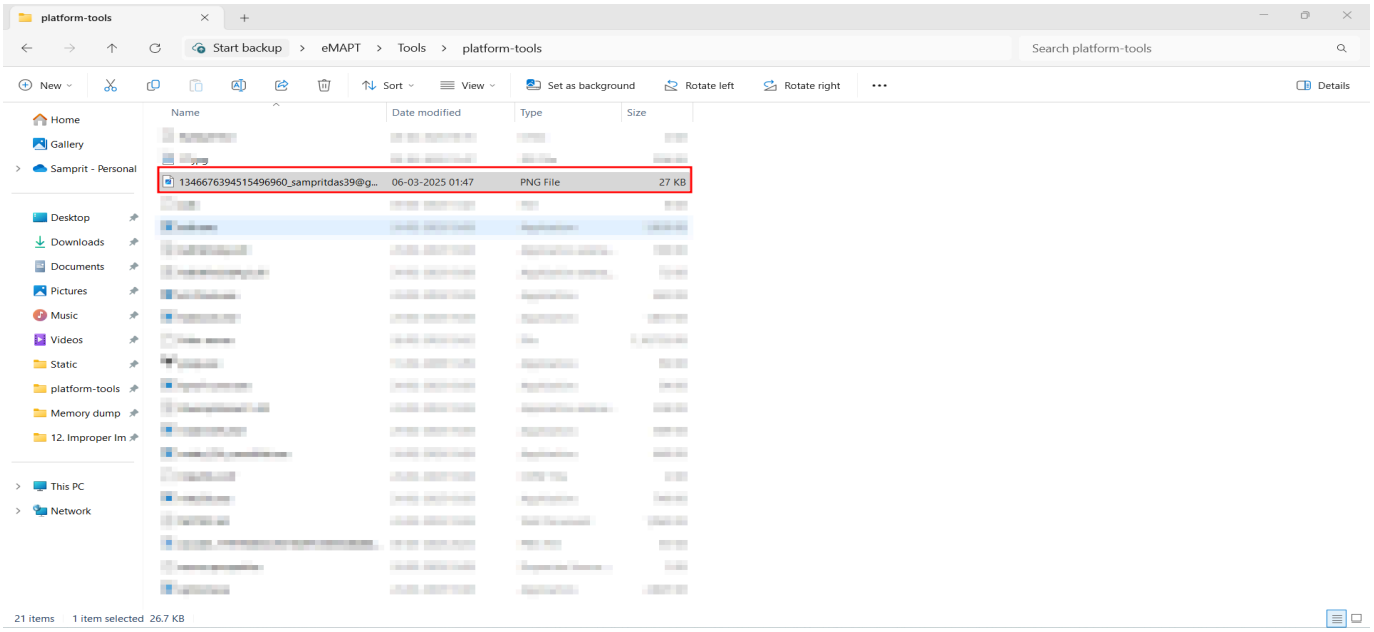


3. cat the libCachedImageData.db-wal and observe that it also disclose the account id.

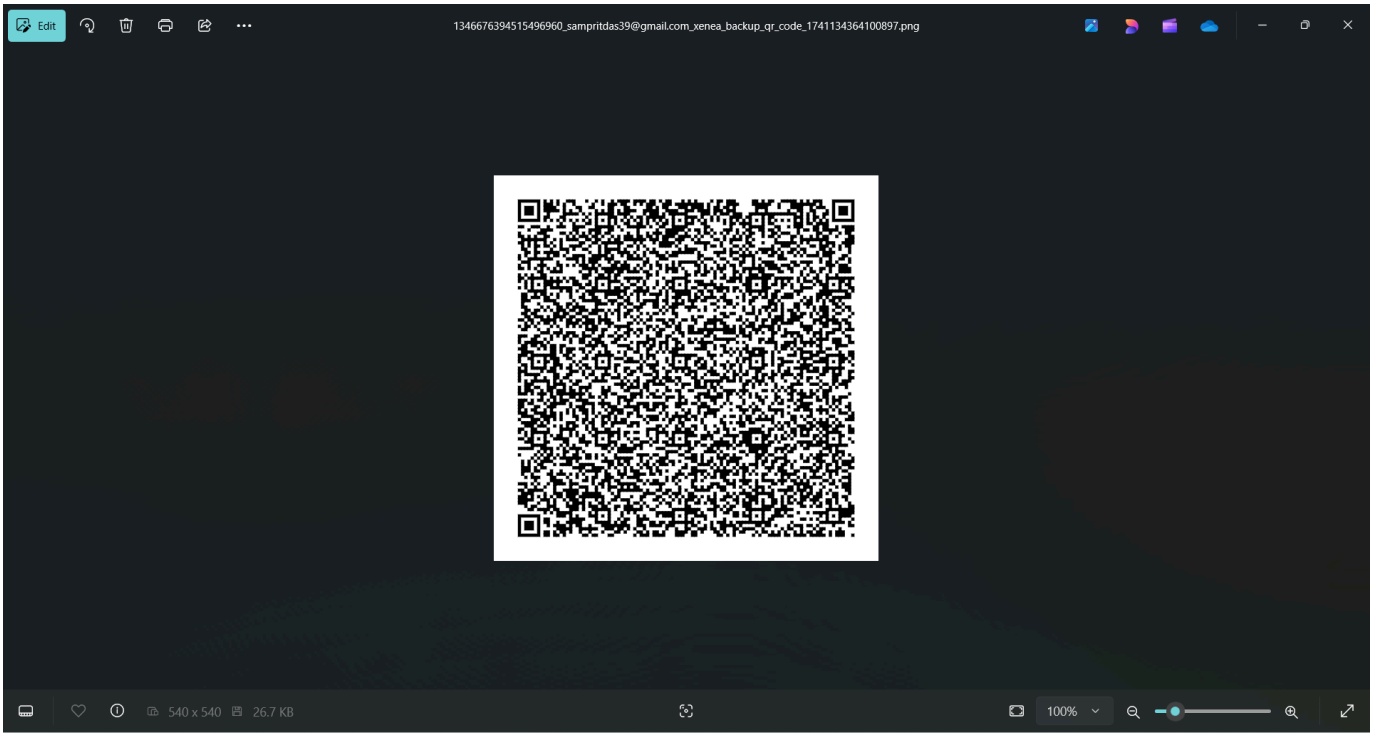
```
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/files # cat libCachedImageData.db-wal
7◆-◆b◆◆.◆◆◆u◆6◆◆b◆◆.◆8'◆◆◆
◆◆◆◆##◆QtablecacheObjectcacheObjectCREATE TABLE cacheObject (id_metadata (locale TEXT)b◆◆.◆O◆◆|f◆◆◆b◆◆.◆◆u6qOI
        _id integer primary key,
        url text,
        key text,
        relativePath text,
        eTag text,
        validTill integer,
        touched integer,
        length integer
◆◆◆.◆A◆$[g?cctableandroid_metadataandroid_metadataCREATE TABLE android_metadata (locale TEXT)b◆◆.◆ ◆◆I◆<◆b◆◆.◆◆◆◆N◆[◆
◆?◆?]Qhttps://xenea-prod-public-files-bucket.s3.ap-southeast-1.amazonaws.com/networks/xenea.pnghttps://xenea-prod-public-files-bucket.s3.ap-southeast-1.amaz
onaws.com/networks/xenea.pngaf739c40-fa28-11ef-ad3b-eb1beb6bc76a.png"e7195cbaa5734d4048f3be5ef4be4d43"◆◆`◆i◆◆b◆◆.◆◆j◆k◆
◆◆
◆◆k
◆◆YQhttps://cross-wallet-stg-public-files-bucket.s3.ap-northeast-1.amazonaws.com/1346676394515496960_0x2da77097b07fd0ffbd8b501f7e8afbf2dff18c70_Avatar_Walle
t_Addressf7c71c6c-6b1b-41bb-9569-2e506ba8e7a2https://cross-wallet-stg-public-files-bucket.s3.ap-northeast-1.amazonaws.com/1346676394515496960_0x2da77097b07f
d0ffbd8b501f7e8afbf2dff18c70_Avatar_Wallet_Addressf7c71c6c-6b1b-41bb-9569-2e506ba8e7a2af9d9270-fa28-11ef-ad3b-eb1beb6bc76a.*"a◆ecd5e4e94aa408e8da70e1abd74b6
7"◆◆a◆i◆ ,◆
◆?◆?]Qhttps://xenea-prod-public-files-bucket.s3.ap-southeast-1.amazonaws.com/networks/xenea.pnghttps://xenea-prod-public-files-bucket.s3.ap-southeast-1.amaz
onaws.com/networks/xenea.pngaf739c40-fa28-11ef-ad3b-eb1beb6bc76a.png"e7195cbaa5734d4048f3be5ef4be4d43"◆◆`◆i◆◆ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/f
iles #
```

4. Navigate to `com.xena.wallet.stg/cache/*` and observe that it discloses the account ID and email ID. Download the file to your system and analyze its contents.

```
C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools>adb -s 127.0.0.1:21503 shell
ASUS_I005DA:/ # cd /data/user/0/com.xenea.wallet.stg/cache/45f2c7fc-fd3b-4060-bfc4-a227914fd074
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/45f2c7fc-fd3b-4060-bfc4-a227914fd074 # ls
1346676394515496960_sampritdas39@gmail.com_xenea_backup_qr_code_1741134364100897.png
p 1346676394515496960_sampritdas39@gmail.com_xenea_backup_qr_code_1741134364100897.png /sdcard/                                <
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/45f2c7fc-fd3b-4060-bfc4-a227914fd074 # exit

C:\Users\sampr\Desktop\eMAPT\Tools\platform-tools>adb.exe pull /sdcard/1346676394515496960_sampritdas39@gmail.com_xenea_backup_qr_code_1741134364100897.png
/sdcard/1346676394515496960_sampritdas39@gmail.com_xenea_backup_qr_code_1741134364100897.png: 1 file pulled, 0 skipped. 0.3 MB/s (27367 bytes in 0.094s)
```



5. Open the file and observe that it contains the private key QR.



6. The QR is disclosing from another directory also.

```
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache # ls
45f2c7fc-fd3b-4060-bfc4-a227914fd074 6c8bb15d-7cc8-4d2f-8a71-47e881377e99 a70277df-a025-4086-a6eb-fcf3fff8e957 libCachedImageData
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache # cd 6c8bb15d-7cc8-4d2f-8a71-47e881377e99/
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/6c8bb15d-7cc8-4d2f-8a71-47e881377e99 # ls
1346676394515496960_sampritdas39@gmail.com_xenea_backup_qr_code_1741134364100897.png
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/6c8bb15d-7cc8-4d2f-8a71-47e881377e99 # cd ..
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache # cd a70277df-a025-4086-a6eb-fcf3fff8e957/
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/a70277df-a025-4086-a6eb-fcf3fff8e957 # ls
qrcode-179193833-f5316b97c565528d86006854f1c59279.png
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/a70277df-a025-4086-a6eb-fcf3fff8e957 # cd ..
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache # cd libCachedImageData/
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/libCachedImageData # ls
af739c40-fa28-11ef-ad3b-eb1beb6bc76a.png af9d9270-fa28-11ef-ad3b-eb1beb6bc76a.*
ASUS_I005DA:/data/user/0/com.xenea.wallet.stg/cache/libCachedImageData #
```

5.26 Permission Overload: Dangerous Access Granted

Vulnerability Severity	**OWASP Category**
Low	Improper Platform Usage
Tools Used	**Date of reporting**
jadx-gui	11th March 2025

Vulnerability Observation

During testing, it was observed that the application has unnecessary permissions enabled, including:

- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.MANAGE_EXTERNAL_STORAGE

- android.permission.POST_NOTIFICATIONS

These excessive permissions may pose security and privacy risks if exploited by malicious applications or attackers.

Vulnerable location

AndroidManifest.xml

Technical Impact

Unnecessary permissions can increase the application's attack surface, potentially allowing unauthorized access to sensitive user data, file manipulation, and exploitation by malicious applications.

Business Impact

A security breach due to excessive permissions can lead to data leakage, regulatory non-compliance, reputational damage, and potential legal consequences, affecting user trust and business credibility.

Remediation

- Follow the principle of least privilege and remove any unnecessary permissions from the AndroidManifest.xml file.
- Use Scoped Storage to limit file access instead of READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE.
- Avoid MANAGE_EXTERNAL_STORAGE unless absolutely necessary and provide proper justification if used.
- Restrict POST_NOTIFICATIONS permission only to required scenarios and request runtime permissions only when needed.
- Regularly audit and review permissions to ensure minimal exposure to security risks.

Status - Acknowledge

References

- Permissions Overview
- How Malicious Applications Abuse Android Permissions

Steps to Reproduce

1. Open the APK in JADX-GUI and analyze the `AndroidManifest.xml` file.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="8" android:versionName="2.0.0" android:compileSdkVersion="35" android:compile
    <uses-sdk android:minSdkVersion="24" android:targetSdkVersion="34"/>
    <uses-feature android:name="android.hardware.camera" android:required="false"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" android:maxSdkVersion="32"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="32"/>
    <uses-permission android:name="android.permission.USE_BIOMETRIC"/>
    <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
    <queries>
        <intent>
            <action android:name="android.intent.action.PROCESS_TEXT"/>
            <data android:mimeType="text/plain"/>
        </intent>
        <intent>
            <action android:name="android.intent.action.GET_CONTENT"/>
            <data android:mimeType="*/*"/>
        </intent>
    </queries>
    <queries>
        <intent>
            <action android:name="android.intent.action.VIEW"/>
            <data android:scheme="https"/>
        </intent>
    </queries>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
    <uses-permission android:name="android.permission.ACCESS_ADSERVICES_ATTRIBUTION"/>
    <uses-permission android:name="android.permission.ACCESS_ADSERVICES_AD_ID"/>
    <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
    <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <permission android:name="com.xenea.wallet.stg.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION" android:protectionLevel="signature"/>
    <uses-permission android:name="com.xenea.wallet.stg.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"/>
    <application android:label="XENEA Wallet Staging" android:icon="@mipmap/ic_launcher" android:name="android.app.Application" android:extractNativeLibs="false" androi
        <activity android:theme="@style/LaunchTheme" android:name="com.xenea.app.MainActivity" android:exported="true" android:launchMode="singleTop" android:configChar
            <meta-data android:name="io.flutter.embedding.android.NormalTheme" android:resource="@style/NormalTheme"/>
            <intent-filter>
```

5.27 Weak Encryption Algorithm Compromises App Data Security

Vulnerability Severity	**OWASP Category**
Low	Insecure Communication
Tools Used	**Date of reporting**
jadx-gui	11th March 2025

Vulnerability Observation

During testing, it was observed that the application uses a weak encryption algorithm (which is MD5,SHA1) to protect sensitive data.

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

If an application uses a weak encryption algorithm, sensitive data becomes vulnerable to attacks such as brute-force or cryptographic cracking. Attackers can easily decrypt confidential information like passwords, personal details, or financial data. This compromises user privacy, allows unauthorized access, and can lead to data breaches or financial losses. The overall integrity and trustworthiness of the application are severely undermined, exposing users to significant security risks.

Business impact

A weak implementation of a hashing algorithm can lead to security vulnerabilities, such as password cracking or data integrity breaches. To mitigate this risk, use strong, industry-standard hashing algorithms

like bcrypt, Argon2, or PBKDF2 with proper salting and key stretching. Avoid outdated algorithms like MD5 or SHA-1, as they are susceptible to collision and brute-force attacks.

Remediation

To remediate the use of a weak encryption algorithm, replace it with a modern, secure algorithm like AES-256 or RSA with appropriate key management practices. Ensure that encryption keys are stored securely, and implement strong key derivation techniques such as PBKDF2 or bcrypt for password storage. Regularly audit cryptographic libraries and algorithms to stay compliant with industry standards and best practices, ensuring the application's security remains robust.

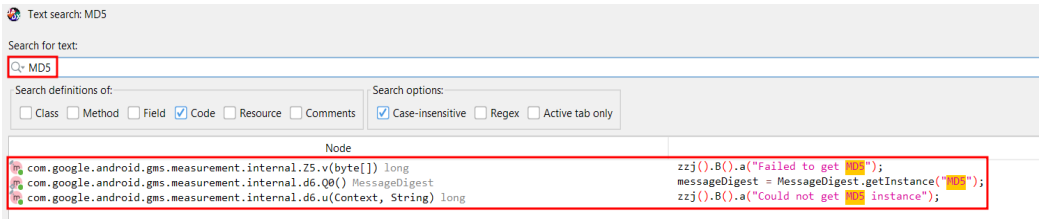
Status - Fixed

References

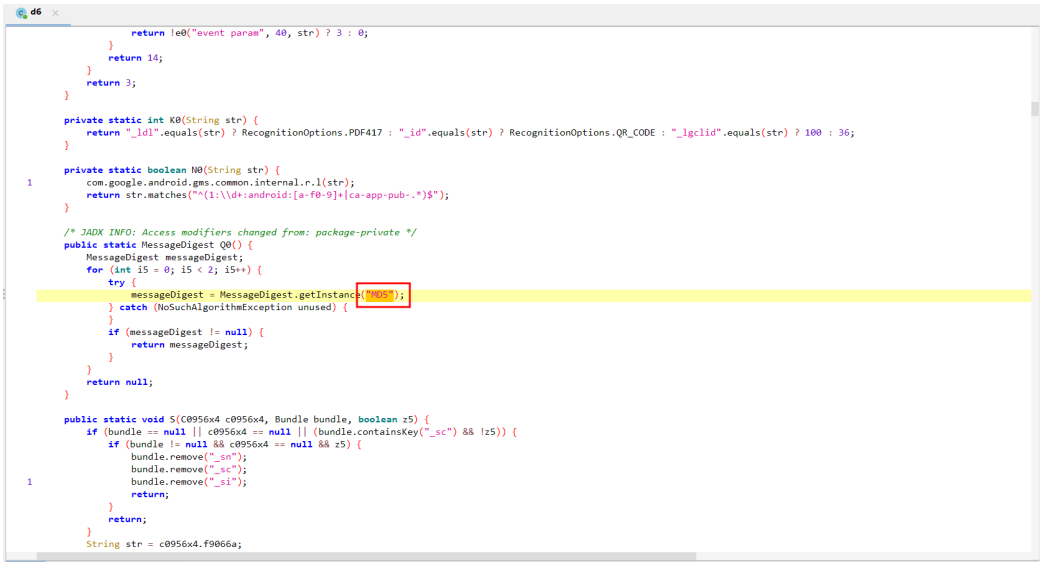
- MD5 vs SHA-1 vs SHA-2 – Which is the Most Secure Encryption Hash and How to Check Them
- Weak Hashing Algorithm Vulnerability
- What is MD5? Understanding Message-Digest Algorithms

Steps to Reproduce

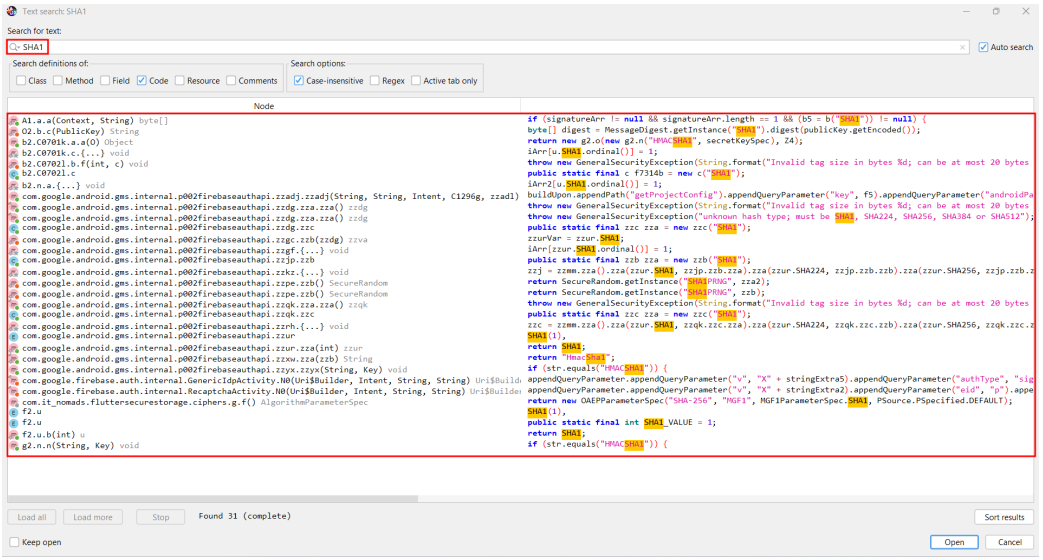
1. Open the app in the jadx-gui and search "MD5"



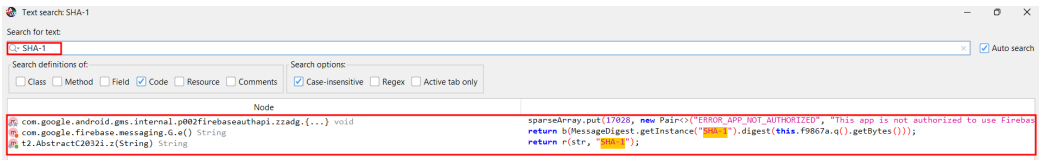
2. Now, open any of the highlighted value and observe it

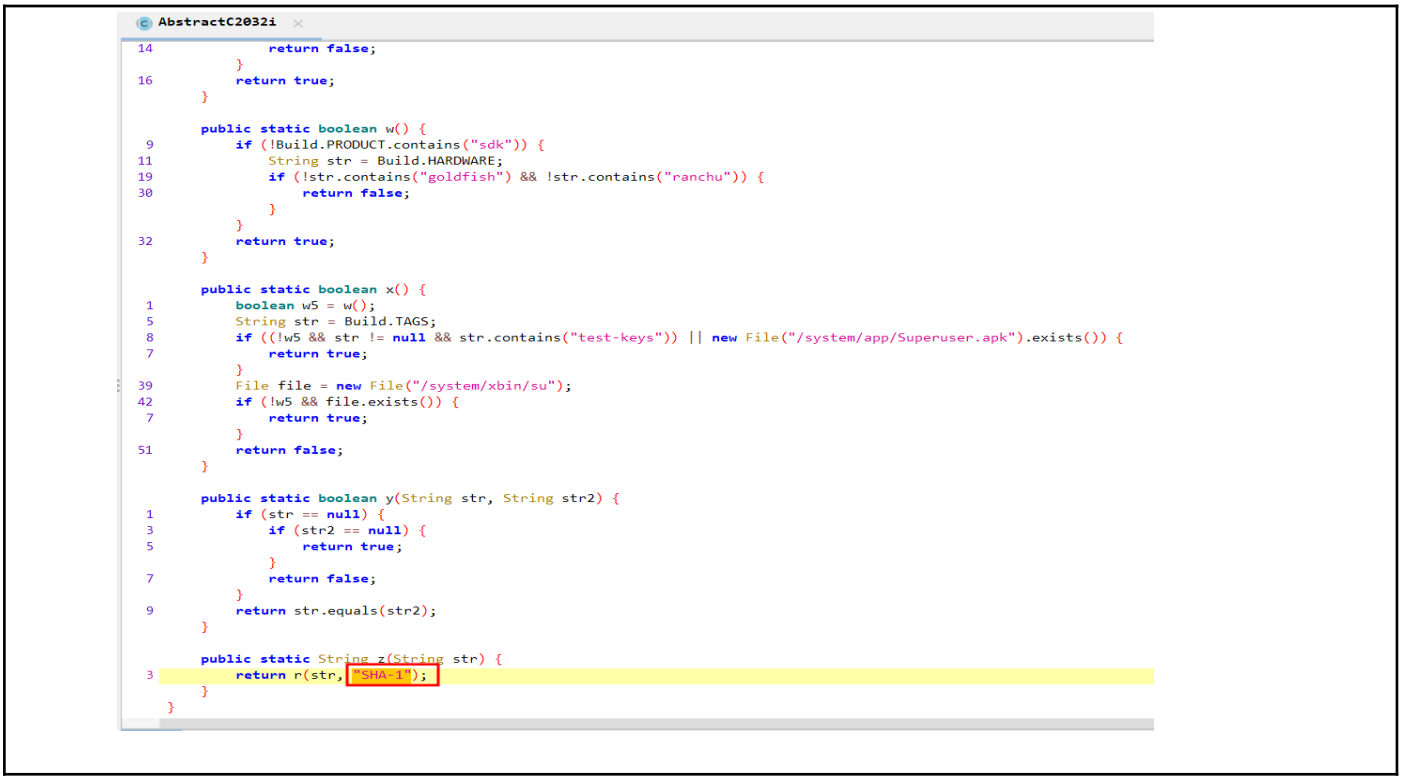


3. Do same for SHA1



```
        String f5 = c1296g.r().f();
        if (f5 != null) {
            return f5;
        }
        String c5 = c1296g.r().c();
        if (!c5.startsWith("1:") && !c5.startsWith("2:")) {
            return c5;
        }
        String[] split = c5.split(":");
        if (split.length != 4) {
            return null;
        }
        String str = split[1];
        if (str.isEmpty()) {
            return null;
        }
        return str;
    }

    private static String c(PublicKey publicKey) {
        try {
            byte[] digest = MessageDigest.getInstance("SHA1").digest(publicKey.getEncoded());
            digest[0] = (byte) (((digest[0] & 15) + 112) & 255);
            return Base64.encodeToString(digest, 0, 8, 11);
        } catch (NoSuchAlgorithmException unused) {
            Log.w("ContentValues", "Unexpected error, device missing required algorithms");
            return null;
        }
    }

    private String d(String str) {
        try {
            return new JSONObject(str).getString("token");
        } catch (JSONException unused) {
            return null;
        }
    }

    private PublicKey e(String str) {
        try {
            return KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(Base64.decode(str, 8)));
        } catch (IllegalArgumentException | NoSuchAlgorithmException | InvalidKeySpecException e5) {
```





5.28 Missing Security Headers

Vulnerability Severity	**OWASP Category**
Medium	Missing Security Headers
Tools Used	**Date of reporting**
Burp Suite	10th March 2025

Vulnerability Observation

While intercepting the API request in Burp Suite and sending it to Repeater, I observed that several critical security headers were missing in the response.The absence of Strict-Transport-Security (HSTS) allows attackers to perform Man-in-the-Middle (MITM) attacks by forcing users to connect over an insecure HTTP connection.The Content-Security-Policy (CSP) header was missing, making the application vulnerable to Cross-Site Scripting (XSS) attacks and data injection threats.The X-Frame-Options header was not present, leaving the application susceptible to Clickjacking attacks, where attackers can trick users into performing unintended actions.Additionally, the missing Referrer-Policy header increases the risk of data leakage, as sensitive information could be exposed when users navigate between different sites.

Vulnerable location

http://api-stg.xenea.network/*

https://mobile-api.staging.xenea.app/*

Technical Impact

The absence of Strict-Transport-Security (HSTS) allows attackers to perform Man-in-the-Middle (MITM) attacks, enabling data interception and downgrade attacks to force insecure HTTP connections.Without

Content-Security-Policy (CSP), the application is vulnerable to Cross-Site Scripting (XSS), where attackers can inject malicious scripts to steal user data or hijack sessions.The missing X-Frame-Options header makes the application susceptible to Clickjacking attacks, where an attacker can embed the API response in an iframe and trick users into executing unintended actions.The absence of Referrer-Policy increases the risk of data leakage, as sensitive information (e.g., API keys, tokens) may be exposed when navigating between sites.These missing security headers significantly increase the attack surface, making the API and its users vulnerable to multiple exploitation techniques.

Business impact

The missing security headers increase the risk of MITM attacks, XSS, and Clickjacking, leading to data breaches, account takeovers, and financial fraud.Attackers can exploit these weaknesses to steal user data, manipulate API responses, or launch phishing attacks, damaging customer trust and brand reputation.Service disruptions and unauthorized access could lead to financial losses, operational downtime, and competitive risks.

Remediation

Enable Strict-Transport-Security (HSTS) to enforce secure HTTPS connections and prevent MITM attacks.Implement Content-Security-Policy (CSP) to mitigate XSS attacks by restricting script execution.Add X-Frame-Options to prevent Clickjacking attacks by blocking unauthorized iframes.Set Referrer-Policy to limit data leakage when users navigate between sites

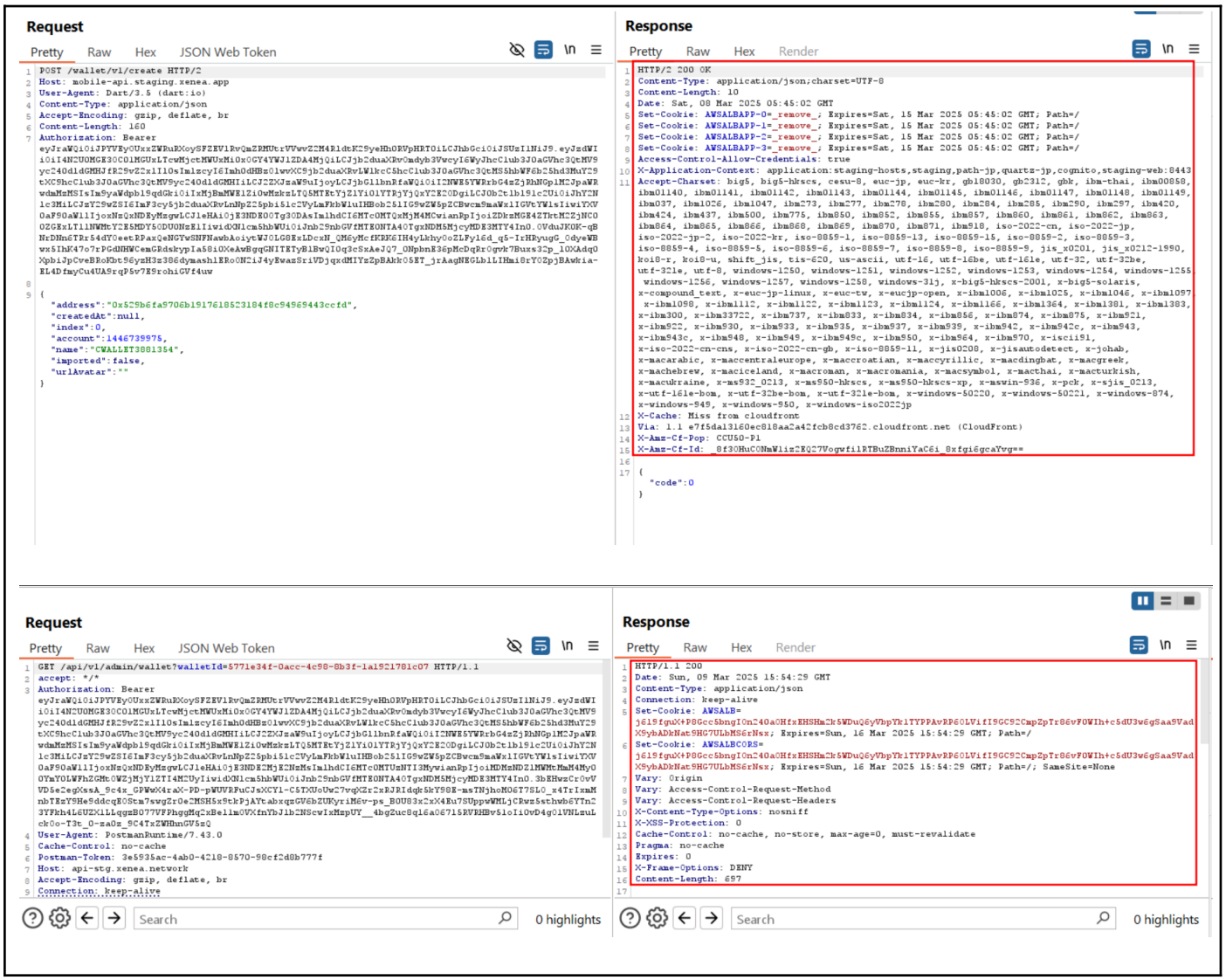
Status - Fixed

References

- www-project-secure-headers
- Strict-Transport-Security
- Content-Security-Policy
- X-Frame-Options
- Referrer-Policy

Steps to Reproduce

1. Capture the API Request in Burp Suite. Send the Request to Burp Suite Repeater. Analyze the Response for Missing Security Headers.



5.29 Missing cookie attributes - httponly,secure,SameSite and path set to root

Vulnerability Severity	**OWASP Category**
Low	Missing cookie attributes
Tools Used	**Date of reporting**
Burp Suite	10th March 2025

Vulnerability Observation

The API response contains a Set-Cookie header with missing security attributes:

- HttpOnly flag is not set, making the cookie accessible via JavaScript, increasing the risk of XSS-based session hijacking.

- Secure flag is not set, allowing the cookie to be transmitted over an unencrypted HTTP connection, making it vulnerable to MITM attacks.
- SameSite is set to None, making the application susceptible to Cross-Site Request Forgery (CSRF) attacks.
- Path is set to /, increasing the attack surface by allowing access to the cookie across all endpoints.

Vulnerable location

http://api-stg.xenea.network/*

https://mobile-api.staging.xenea.app/*

Set-Cookie headers in the response.

Technical Impact

Session Hijacking: Attackers can steal session cookies via JavaScript execution (XSS).MITM Attacks: Cookies can be intercepted if transmitted over an unencrypted connection.CSRF Attacks: The application is vulnerable to unauthorized actions performed on behalf of an authenticated user.

Business impact

User Account Takeover due to session hijacking.Data Leakage if cookies are intercepted.Unauthorized Transactions via CSRF exploitation.

Remediation

Set HttpOnly flag to prevent JavaScript access.Set Secure flag to enforce transmission over HTTPS.Use SameSite=Strict or Lax to mitigate CSRF.Restrict Path to the necessary scope to reduce exposure.

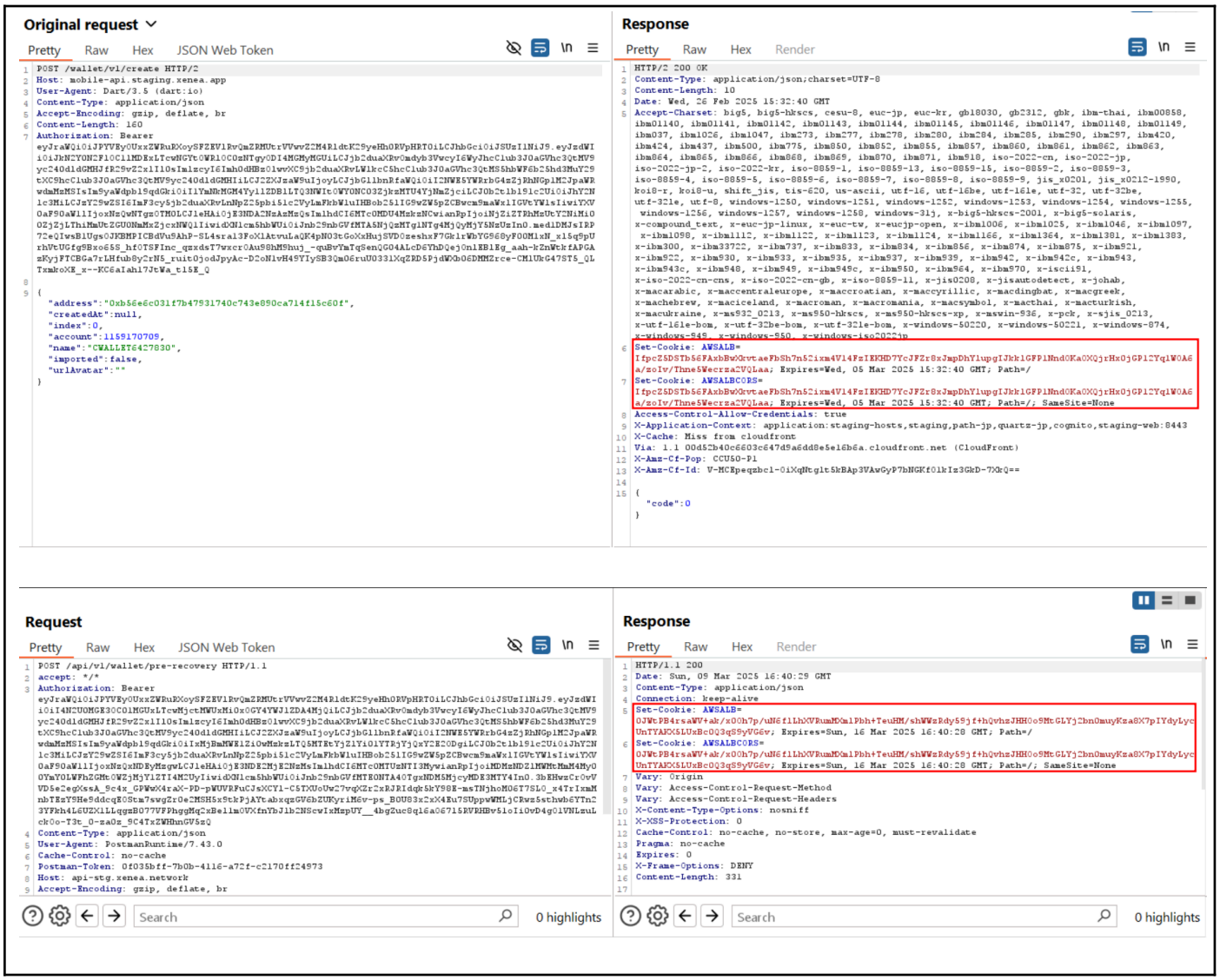
Status - Acknowledge

References

- Mozilla HTTP Cookie Security
- Secure Headers

Steps to Reproduce

1. Capture the API Request in Burp Suite. Send the Request to Burp Suite Repeater. Analyze the Response for Missing cookie attributes - httponly, secure, SameSite and path set to root.



5.30 Account lockout not properly configured

Vulnerability Severity	OWASP Category
Medium	Account lockout not properly configured
Tools Used	**Date of reporting**
Memu	11th March 2025

Vulnerability Observation

During testing, it was observed that the account lockout mechanism in xenea-wallet.apk is not properly enforced. Even after the account is locked, users can still log in using the correct PIN, bypassing the intended security restriction.

Vulnerable location

All the login with pin feature is implemented in xenea-wallet.apk

Technical Impact

Weak account lockout mechanisms allow brute-force attacks, enabling attackers to guess PINs and gain unauthorized access. Compromised user accounts can lead to unauthorized transactions, data theft, and identity fraud.

Business Impact

Increased risk of account takeovers and financial losses for users. Non-compliance with security standards may lead to regulatory fines. Loss of user trust, damaging the company's reputation.

Remediation

Implement a proper account lockout mechanism that temporarily locks the account after a defined number of failed login attempts.

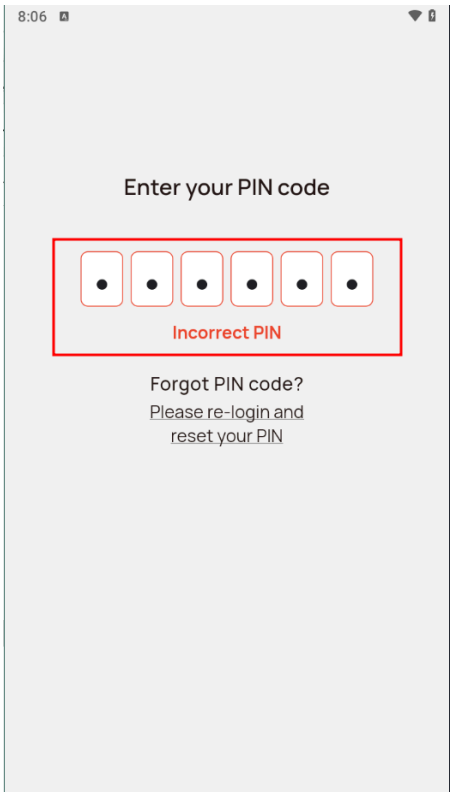
Status - Acknowledge

References

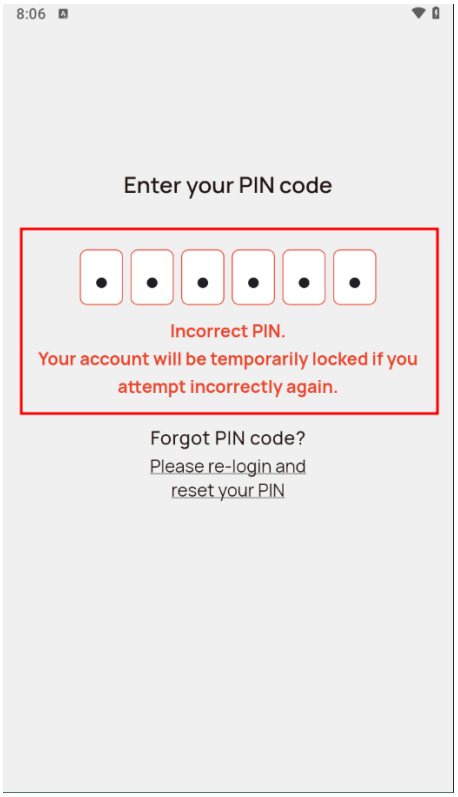
- OWASP Authentication Cheat Sheet
- NIST Guidelines on Account Lockout

Steps to Reproduce

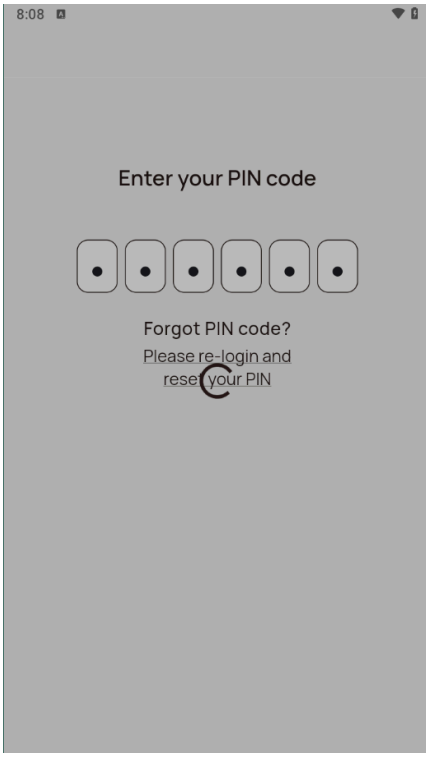
1. Open the Android application and navigate to the reset pin module where the password is required.



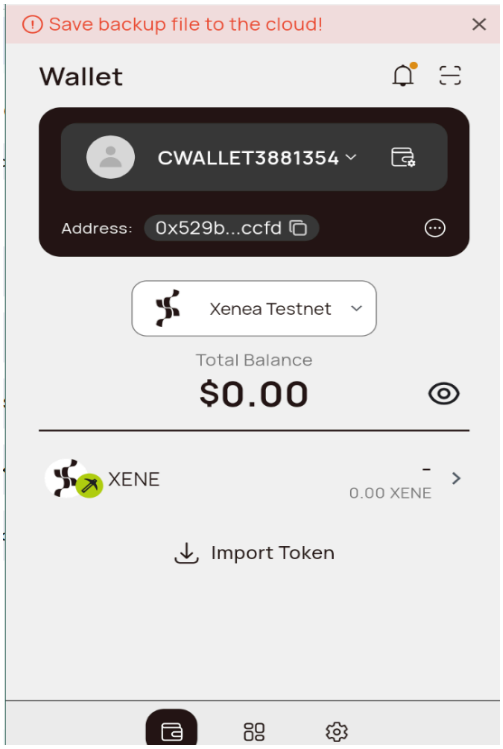
2. Enter an incorrect password multiple times (e.g., 3-5 attempts) until the application shows a message stating that the password attempt limit has been exceeded.



3. Observe if the system restricts further login attempts or prevents password input as expected. Now enter the correct password after exceeding the limit.



4. Notice that despite the system indicating that the account is locked, it still allows login with the correct password.



5.31 No rate limit leads to Dos.

Vulnerability Severity	OWASP Category
Medium	No ratelimit
Tools Used	**Date of reporting**
Burp Suite Intruder	11th March 2025

Vulnerability Observation

The application allows users to add notes at a specific location. However, during testing, it was discovered that there is no rate-limiting mechanism in place, allowing an attacker to create multiple notes from the same address without any restrictions.

Vulnerable location

https://mobile-api.staging.xenea.app/wallet/v1/create

Parameter: address

Technical Impact

The absence of rate limiting on the note creation functionality allows attackers to automate requests and create an unlimited number of notes from the same address. This can lead to resource exhaustion, impacting application performance and stability.Attackers can exploit this to spam the system, manipulate data integrity, or overload the database, potentially causing denial-of-service (DoS) attacks. Additionally, automated bot abuse can degrade user experience and make legitimate use of the feature

difficult.Without proper restrictions, this vulnerability can be combined with scripted attacks to exploit business logic flaws, leading to unauthorized bulk data insertion and system abuse

Business impact

The absence of rate limiting allows attackers to flood the system with excessive notes, leading to database bloating and potential service degradation. If exploited, this could slow down or crash the application, impacting legitimate users and overall system performance. Attackers can also abuse this flaw for spam, fraudulent content creation, or data manipulation, reducing trust in the platform. This not only increases operational costs but also risks reputational damage and non-compliance with security best practices like OWASP API Security Top 10.

Remediation

To mitigate this vulnerability, the application should implement rate limiting to restrict the number of notes that can be created from a single address within a specific time frame. Server-side validation should enforce limits on repeated requests to prevent automated abuse. Implementing CAPTCHA, authentication-based throttling, or IP-based request restrictions can further enhance security. Additionally, monitoring and logging mechanisms should be deployed to detect and block excessive requests, ensuring system stability and compliance with security best practices

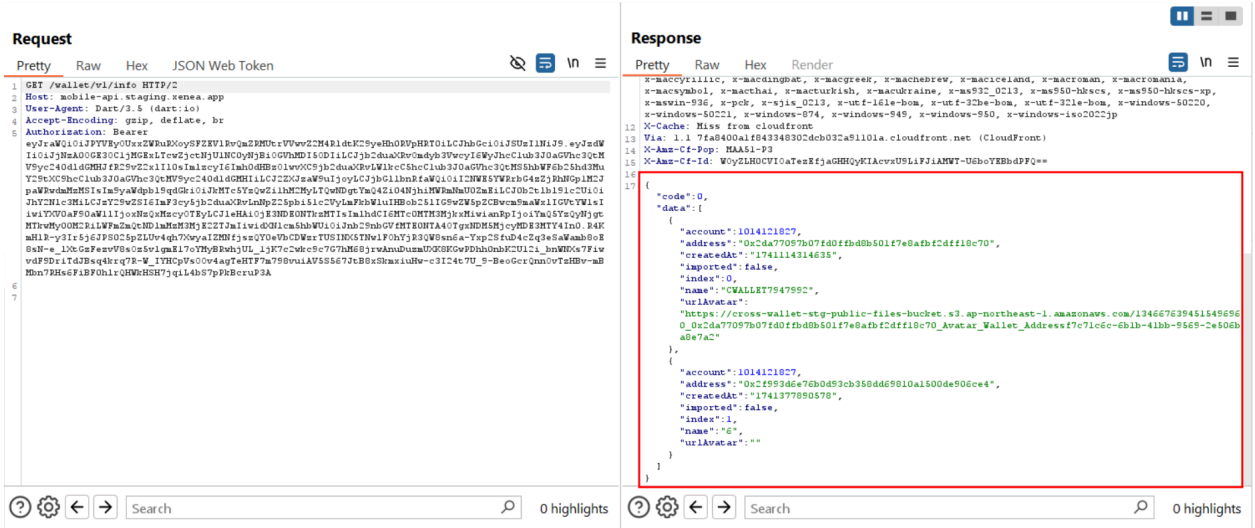
Status - Fixed

References

- Rate_Limiting_Cheat_Sheet
- rate-limiting
- X-RateLimit-Limit
- rate-limiting-strategies-techniques

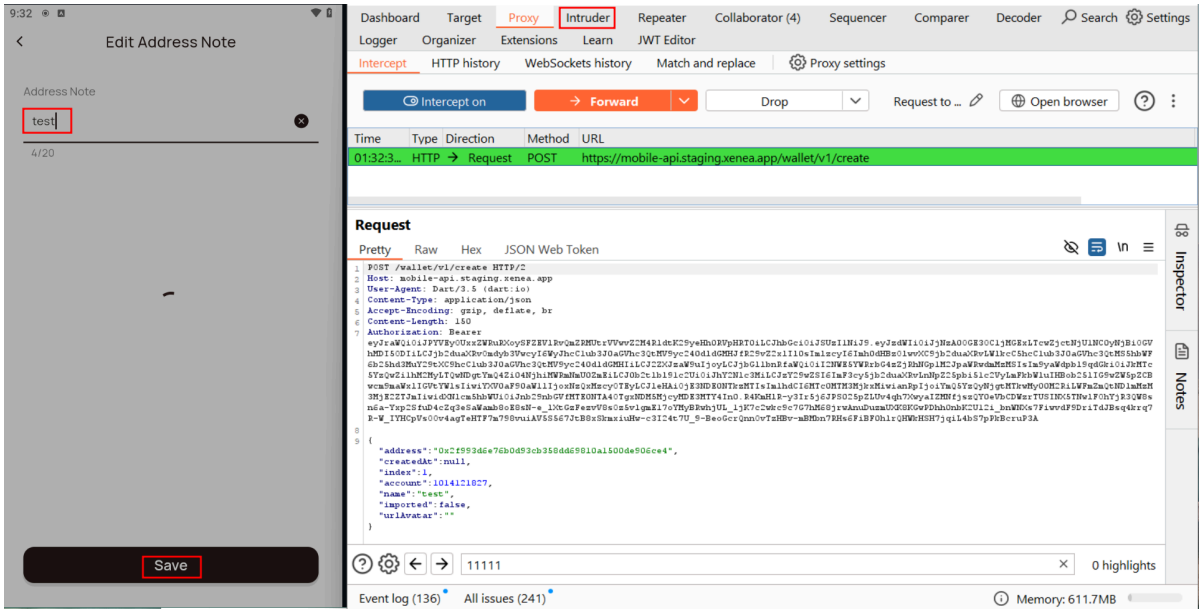
Steps to Reproduce

1. Observe that currently there are only 2 notes created.

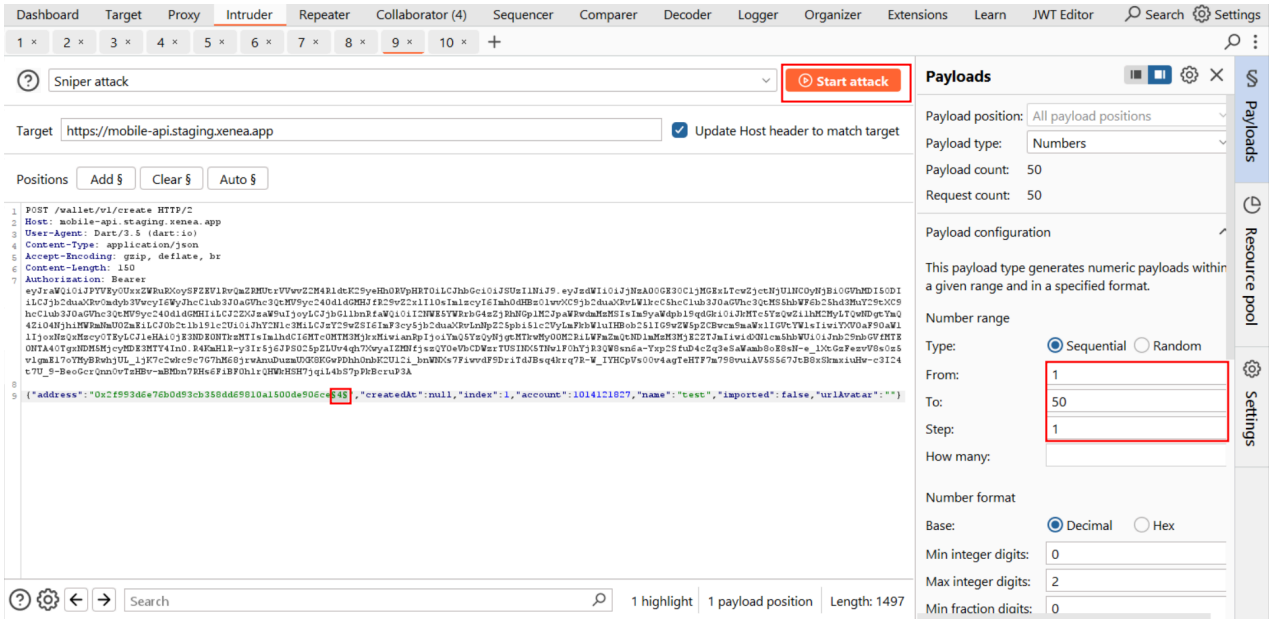


2. Now navigate to the **"Add Note"** section. Enter a sample note and submit it, Capture the request

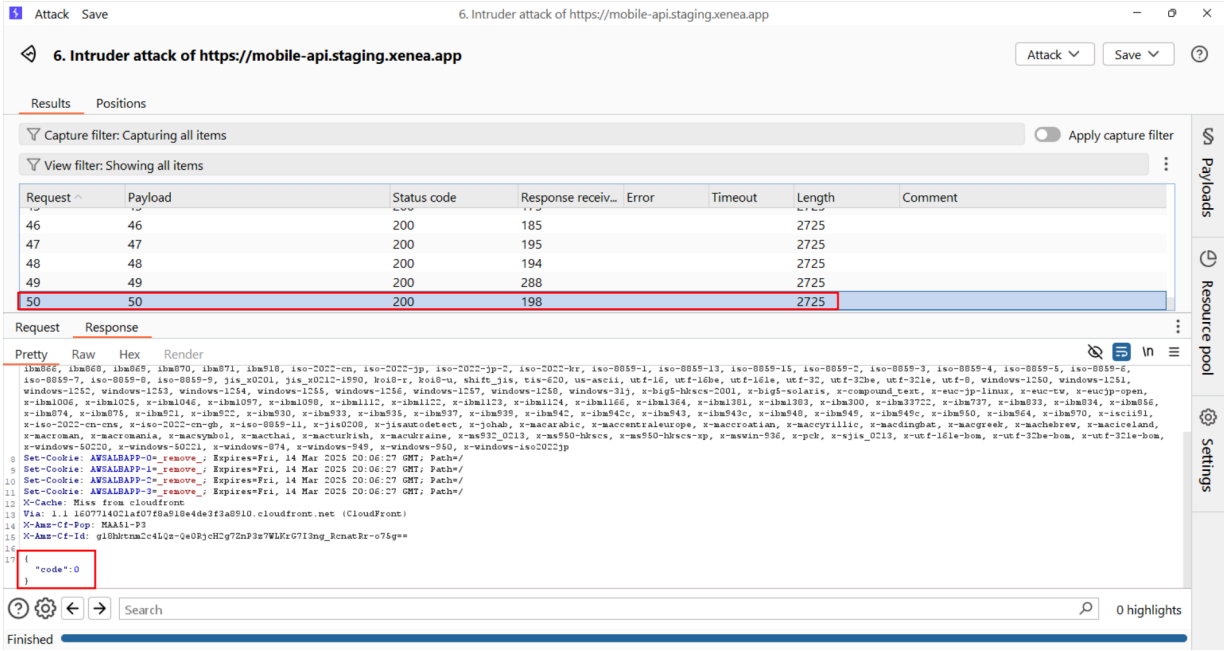
on the burpsuite and send it to the intruder.



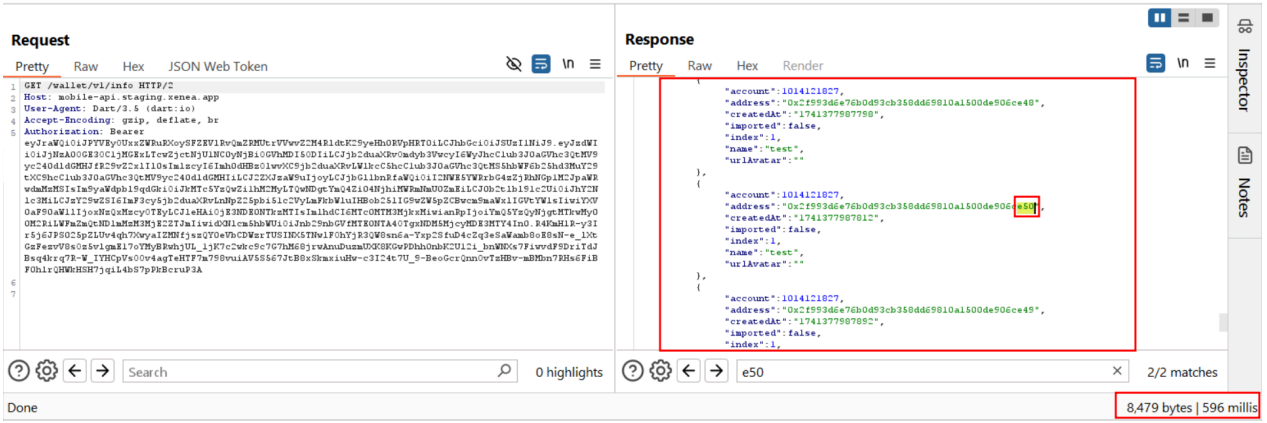
3. Set the Insertion Point. the note_address or note_id field and highlight it. "Add §" to set it as a variable parameter for payload injection and run it for 50 times and click on start attack.



4. Now check the server responses and it will allow you to create 50 notes.



5. Now check that the multiple duplicate notes were created under the *50 address.



5.32 PIN History Not Enforced

Vulnerability Severity	**OWASP Category**
Medium	PIN History Not Enforced
Tools Used	**Date of reporting**
Memu	11th March 2025

Vulnerability Observation

The application allows users to reset their PIN but does not enforce PIN history restrictions. This means a user can reset the PIN to the same one previously used, which defeats the purpose of a secure reset mechanism. A strong authentication system should prevent the reuse of old PINs to enhance security and mitigate risks like credential stuffing or unauthorized access.

Vulnerable location

Affected Module: Authentication & Security

Feature: PIN Reset

Technical Impact

The lack of PIN history enforcement in the reset functionality significantly weakens authentication security. Attackers or users can reuse previously compromised PINs, making brute-force attacks easier and reducing the effectiveness of a PIN reset. Since the application does not check against previously used PINs, a malicious user who gains access once can reset and reuse the same PIN, maintaining persistent access. This violates authentication best practices and increases the risk of unauthorized access, credential stuffing, and account takeovers. Additionally, the absence of PIN history validation fails to comply with security standards like OWASP Authentication Guidelines and PCI-DSS, which recommend preventing credential reuse to enhance security.

Business impact

Allowing PIN reuse weakens account security, making it easier for attackers to regain access after an initial compromise. This increases the risk of account takeovers, fraud, and unauthorized transactions, leading to financial losses. Failure to enforce PIN history also violates compliance standards like OWASP, PCI-DSS, and GDPR, which could result in regulatory penalties or legal consequences. Additionally, if exploited at scale, this vulnerability could lead to customer distrust, reputational damage, and increased support costs due to frequent account compromises and security incidents.

Remediation

- The application should enforce PIN history restrictions by storing a hash of the last 3–5 previously used PINs and preventing users from reusing them. Implement server-side validation to check new PINs against stored hashes before allowing a reset.
- To further strengthen security, enforce PIN complexity rules (e.g., minimum length, no sequential digits) and rate-limiting to prevent brute-force attempts. Additionally, logging and monitoring should be implemented to detect repeated reset attempts, ensuring compliance with OWASP, PCI-DSS, and other security standards.

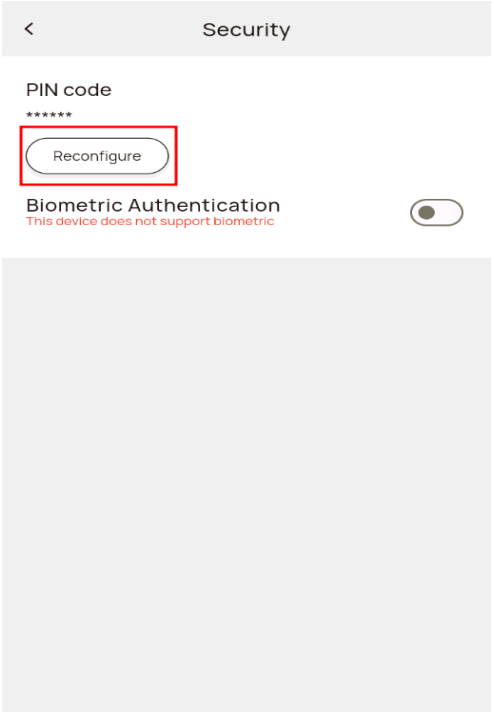
Status - Fixed

References

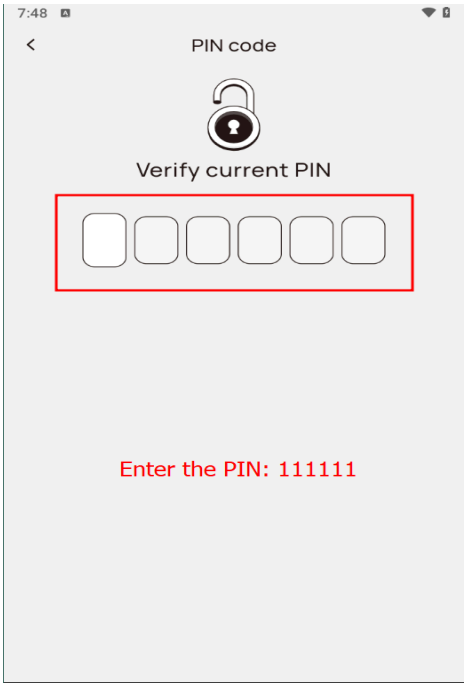
- Password_Storage_Cheat_Sheet
- PCI_DSS_v4
- Enforce password history

Steps to Reproduce

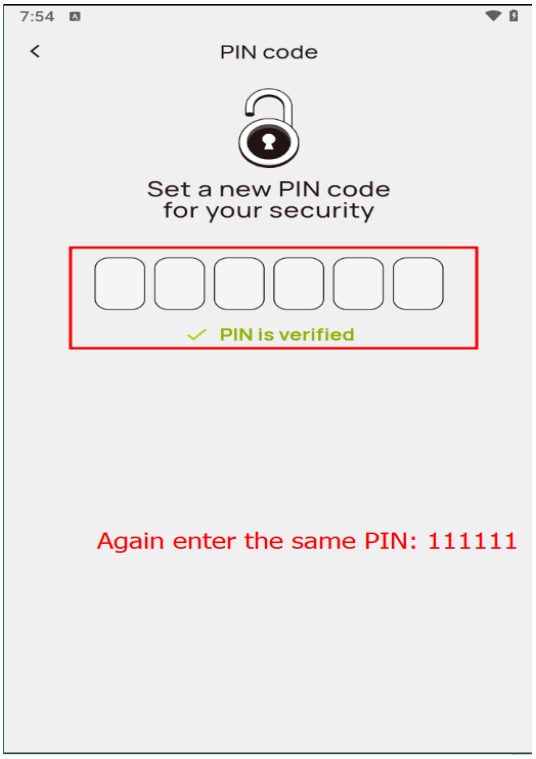
1. Open the application and navigate to the PIN Reset section and click on it.



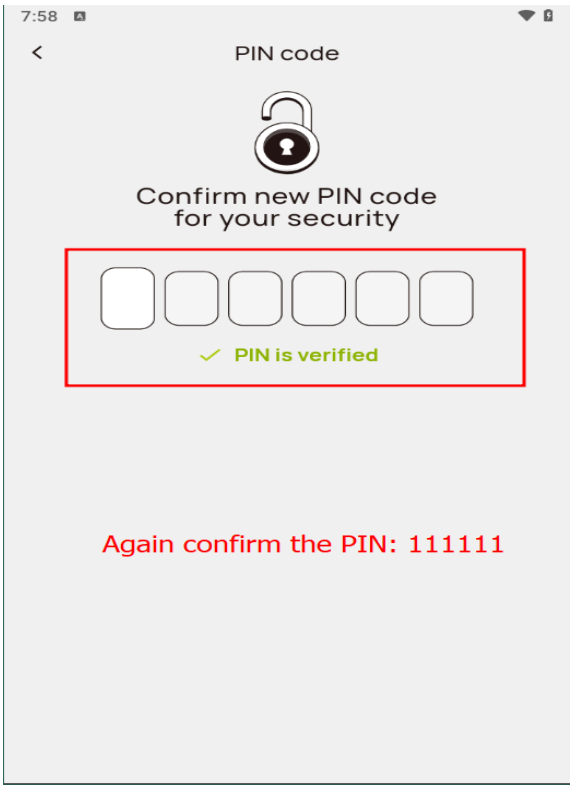
2. Now enter the current PIN.

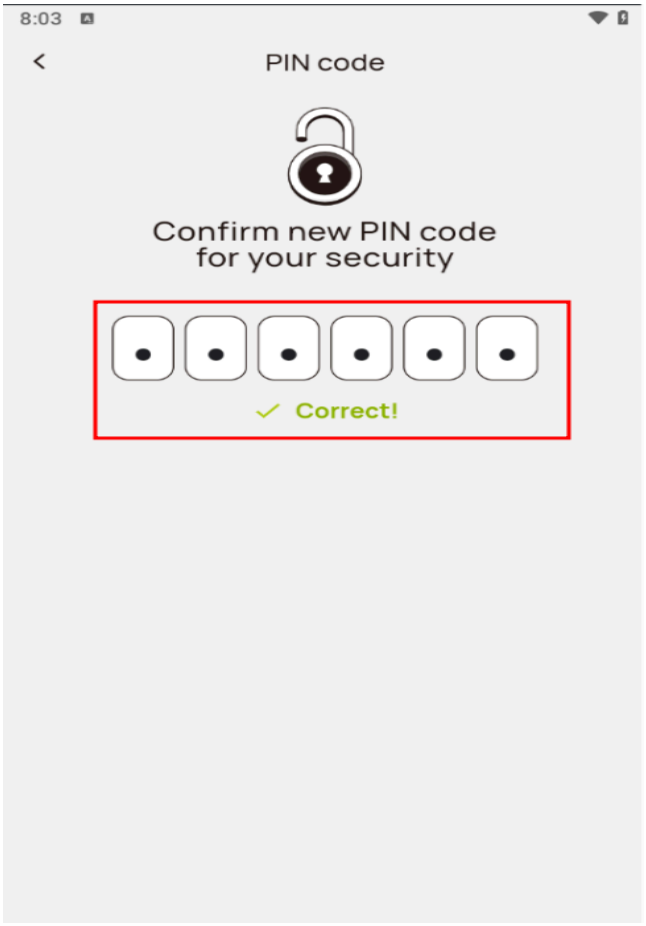


3. Try changing the PIN again and set the same PIN as before.



4. Observe that the application accepts the old PIN and ask you to enter the pin again , Confirming that no PIN history enforcement exists.





5.33 Public JWKS Exposure

Vulnerability Severity	OWASP Category
Medium	Information Disclosure
Tools Used	**Date of reporting**
Burp Suite	11th March 2025

Vulnerability Observation

The API endpoint http://api-stg.xenea.network/api/v1/auth/jwks.json exposes sensitive information, including API details, JWT token algorithm, key ID (kid), and staging environment configurations. This could allow attackers to enumerate API endpoints, exploit weak security controls, or manipulate JWT authentication. The goal of this test is to identify these risks, assess potential misconfigurations such as CORS issues, and recommend security improvements to restrict unauthorized access, secure sensitive data, and prevent API abuse

Vulnerable location

http://api-stg.xenea.network/api/v1/auth/jwks.json

Technical Impact

The exposure of JWKS (/api/v1/auth/jwks.json) poses a critical security risk, as it allows attackers to manipulate JWT authentication mechanisms. By exploiting the exposed key ID (kid), an attacker could potentially forge valid authentication tokens, leading to unauthorized access and privilege escalation. If

the API blindly trusts all JWKS keys, this could result in authentication bypass, enabling attackers to impersonate users, including administrators. Additionally, the exposure of staging environment details increases the risk of API enumeration, allowing attackers to map API endpoints and system architecture for further exploitation.

Business impact

The exposure of JWKS (/api/v1/auth/jwks.json) can lead to unauthorized access, data breaches, and financial losses by allowing attackers to forge JWTs and impersonate users. This could result in fraudulent transactions, service disruptions, and system compromise, affecting business operations. Additionally, regulatory non-compliance (GDPR, ISO 27001, PCI-DSS) could lead to legal penalties and reputational damage, eroding customer trust and investor confidence. If exploited, this vulnerability could cause severe financial, operational, and competitive risks, making immediate remediation crucial.

Remediation

Restrict access to /api/v1/auth/jwks.json by requiring authentication and limiting access to trusted services. Ensure the server strictly validates JWT signatures and does not automatically trust all JWKS keys. Regularly rotate JWKS keys and store private keys securely using HSM or secret management tools. Enforce strong CORS policies to prevent unauthorized cross-origin requests. Monitor API requests for suspicious activity and set up alerts for unexpected key access. Use short-lived JWT tokens with proper permission controls to minimize security risks.

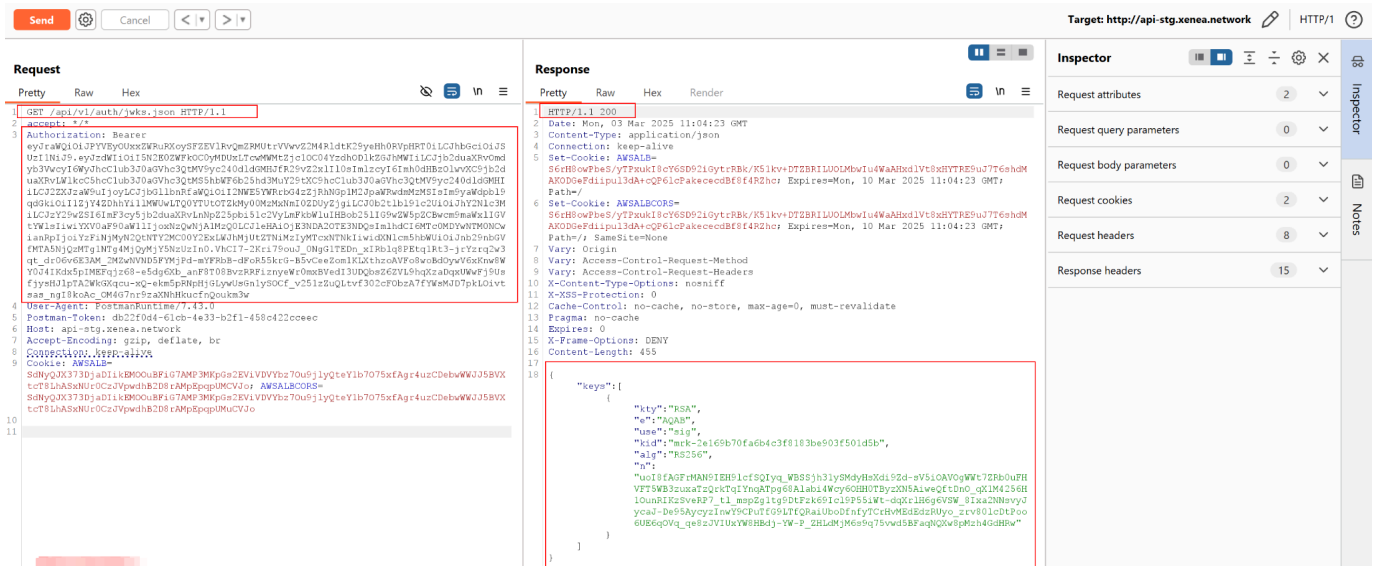
Status - Acknowledge

References

- [json-web-key-sets]
- [token-types]
- [rfc7517]

Steps to Reproduce

1. Intercept the API Request in Burp Suite. Send the Request to Burp Suite Repeater. Analyze the Leaked Information.



5.34 Concurrent login allowed

Vulnerability Severity	OWASP Category
Low	Concurrent login allowed
Tools Used	**Date of reporting**
Memu	10th March 2025

Vulnerability Observation

The application allows a single user account to be logged in from multiple devices simultaneously without any session management restrictions. During testing, it was observed that a user can log in from two or more different devices using the same credentials without being logged out from the previous session. A secure authentication system should restrict concurrent logins or provide an option for users to view and manage active sessions to prevent unauthorized access.

Vulnerable location

Module: Authentication & Session Management

Affected Feature: User Login & Session Handling

Technical Impact

Account Takeover Risk: If an attacker gains access to a user's credentials, they can log in from another device and maintain access even if the real user changes their password (if sessions are not properly revoked). Weak Session Security: Without session tracking, users cannot see or manage active logins, increasing the risk of unauthorized access persistence. No Session Expiry Control: The system does not enforce session invalidation after password changes or logouts, making it possible for old sessions to remain active indefinitely.

Business impact

Allowing concurrent logins weakens account security, increasing the risk of unauthorized access, credential sharing, and fraudulent activities. Attackers or malicious users can maintain persistent access, leading to data breaches, session hijacking, and identity theft. This vulnerability also violates compliance standards like GDPR, PCI-DSS, and ISO 27001, which mandate strong session management policies. Failure to address this issue may result in regulatory fines, reputational damage, and loss of customer trust if exploited at scale.

Remediation

To mitigate this issue, implement strong session management by restricting the number of active sessions per user. Enforce device-based session tracking, allowing only one active session at a time or providing users with the ability to view and manage their active sessions. Enable automatic session invalidation when a user logs in from a new device, prompting them to confirm whether they want to terminate the previous session. Implement session expiration and token revocation mechanisms to ensure that old sessions become invalid after password changes or inactivity.

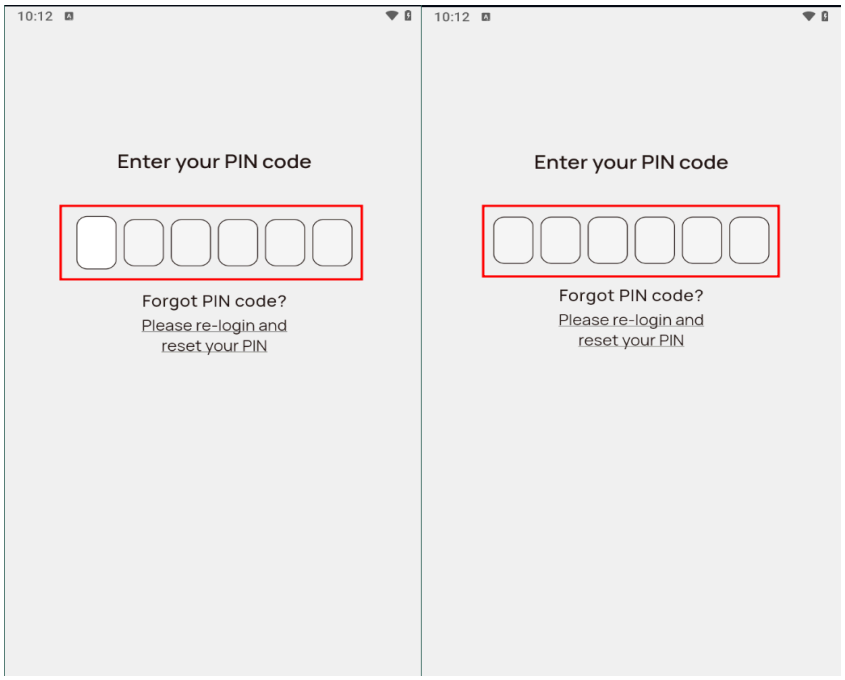
Status - Acknowledge

References

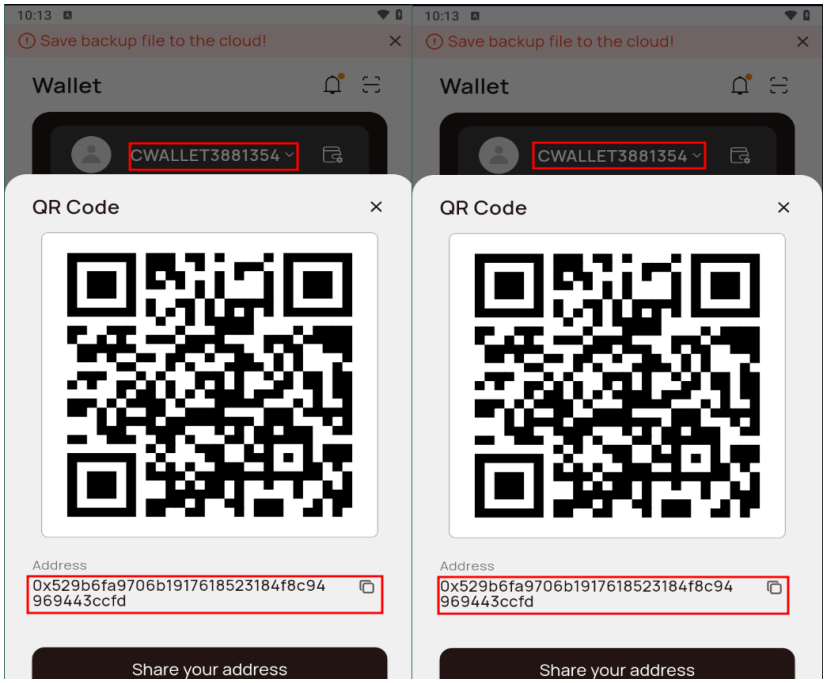
- session-management

Steps to Reproduce

1. Open the Android application and navigate to the login screen. Enter valid credentials and log in with Device 1 (Mobile A or Emulator A).



2. Take the same credentials and log in from Device 2 (Mobile B or Emulator B). Perform the same actions on Device 2 and observe that both sessions remain active simultaneously.



5.35 JWT injection

Vulnerability Severity	OWASP Category
Low	injection
Tools Used	**Date of reporting**
Burp Suite, jwt.io(website)	10th March 2025

Vulnerability Observation

The objective of this test is to assess the security of JWT implementation by identifying vulnerabilities that could lead to authentication bypass, privilege escalation, or unauthorized access. The focus is on detecting weak signature validation, algorithm (alg) and key ID (kid) manipulation, improper token expiration handling, and insecure claim modifications. The goal is to ensure that JWTs are securely validated, protected against forgery, and properly enforced to prevent unauthorized API access, data exposure, and security breaches.

Vulnerable location

The vulnerable endpoint http://api-stg.xenea.network/api/v1/auth/jwks.json exposes sensitive information, including JWT token algorithm and key ID (kid). The affected server URL http://api-stg.xenea.network/api/v1/ reveals a staging environment, which may have weaker security controls. This exposure could allow attackers to enumerate API endpoints, exploit authentication mechanisms, or manipulate JWT validation.

Technical Impact

Manipulating multiple fields within a JWT can lead to authentication bypass, privilege escalation, and unauthorized access if the server does not properly validate token integrity. Modifying the alg field to "none" could allow the server to accept unsigned tokens, effectively disabling signature verification. Exploiting the kid field, an attacker could point to an attacker-controlled key, forging valid JWTs and gaining unauthorized access. Adjusting iat (Issued At) or exp (Expiration) could extend session validity indefinitely, preventing automatic logouts and session expiration. Changing sub (Subject/User ID) could enable impersonation of other users, including administrators, while altering scope or role may grant higher privileges, allowing unauthorized actions within the application. Additionally, modifying aud (Audience) could allow tokens to be misused across unintended services, leading to cross-system authentication bypass. If these manipulations are not properly restricted, attackers can exploit JWT weaknesses to access restricted APIs, escalate privileges, hijack sessions, and compromise sensitive data, making this a critical security vulnerability.

Business impact

Unauthorized JWT manipulation can lead to account takeovers, financial fraud, and data breaches, severely impacting business operations. Attackers can exploit forged tokens to access restricted systems, modify transactions, or extract sensitive user data, leading to regulatory non-compliance (GDPR, PCI-DSS, ISO 27001) and potential legal penalties. A security breach due to JWT forgery could result in reputation damage, customer trust loss, and competitive risks. Exploiting session expiration manipulation may lead to persistent unauthorized access, increasing operational security risks. If left unpatched, this vulnerability could cause severe financial and legal consequences, making immediate remediation essential.

Remediation

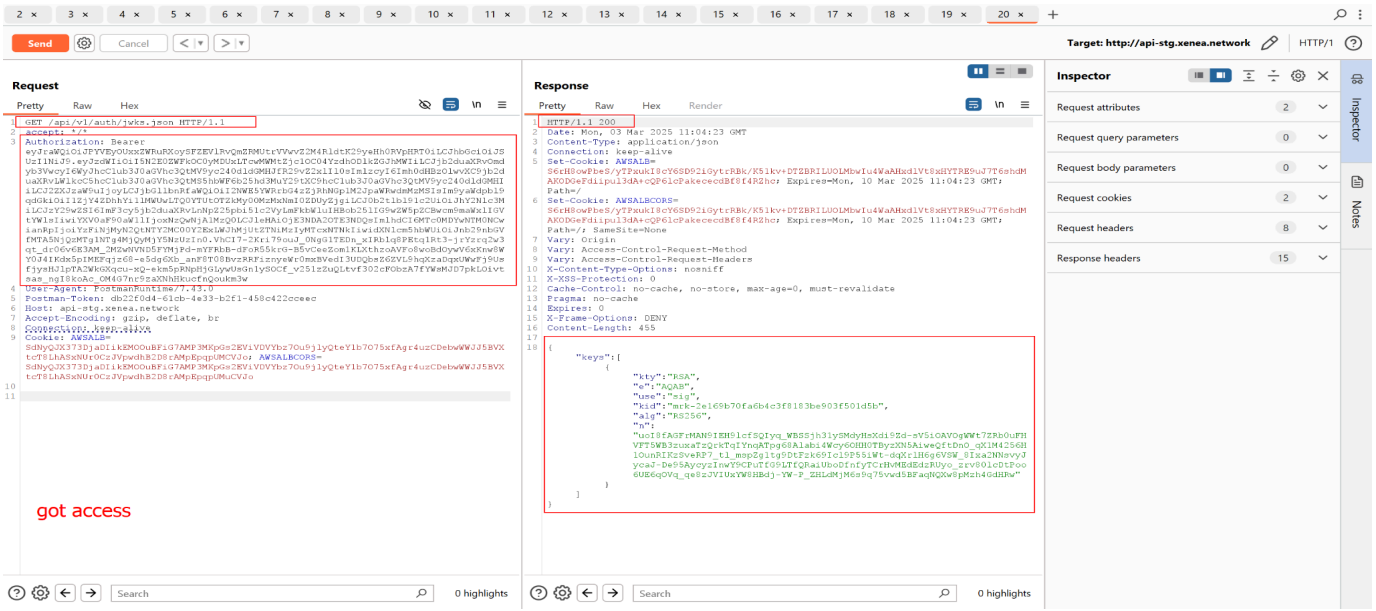
Enforce strict signature validation to prevent the acceptance of unsigned or altered tokens. Disallow the use of alg: none and only permit strong, predefined algorithms such as RS256 or ES256. Implement proper key validation by ensuring the server verifies kid values and does not automatically trust external JWKS sources. Restrict JWT claim modifications by validating iat, exp, sub, aud, and role to prevent privilege escalation, session hijacking, or unauthorized access. Use short-lived JWTs and enforce automatic token expiration and refresh mechanisms. Monitor and log JWT-related authentication failures to detect suspicious activity and potential abuse.

References

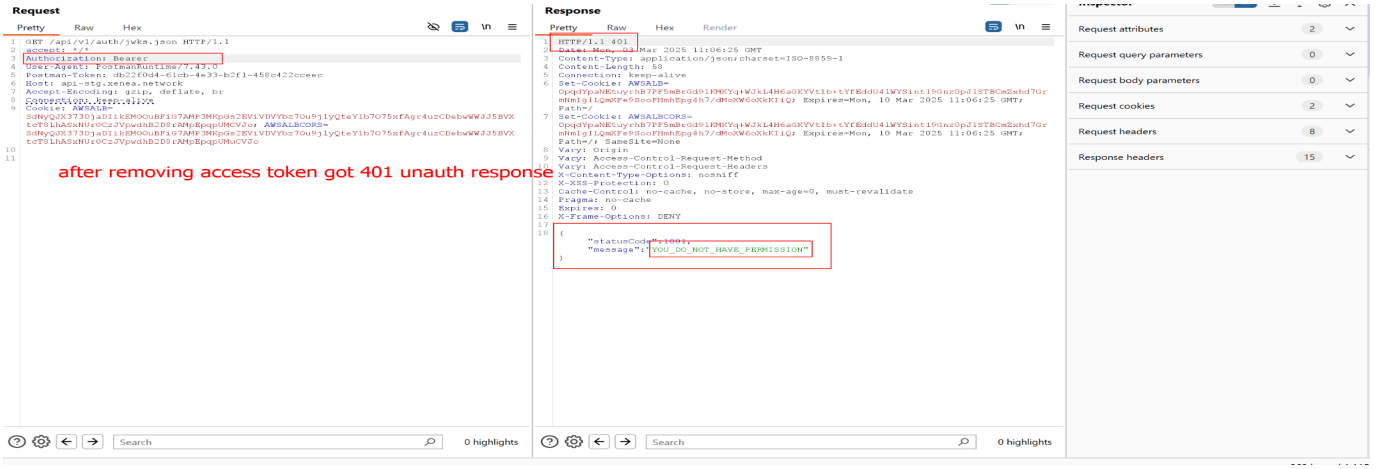
1. [json-web-key-sets]
2. [rfc7517]

Steps to Reproduce

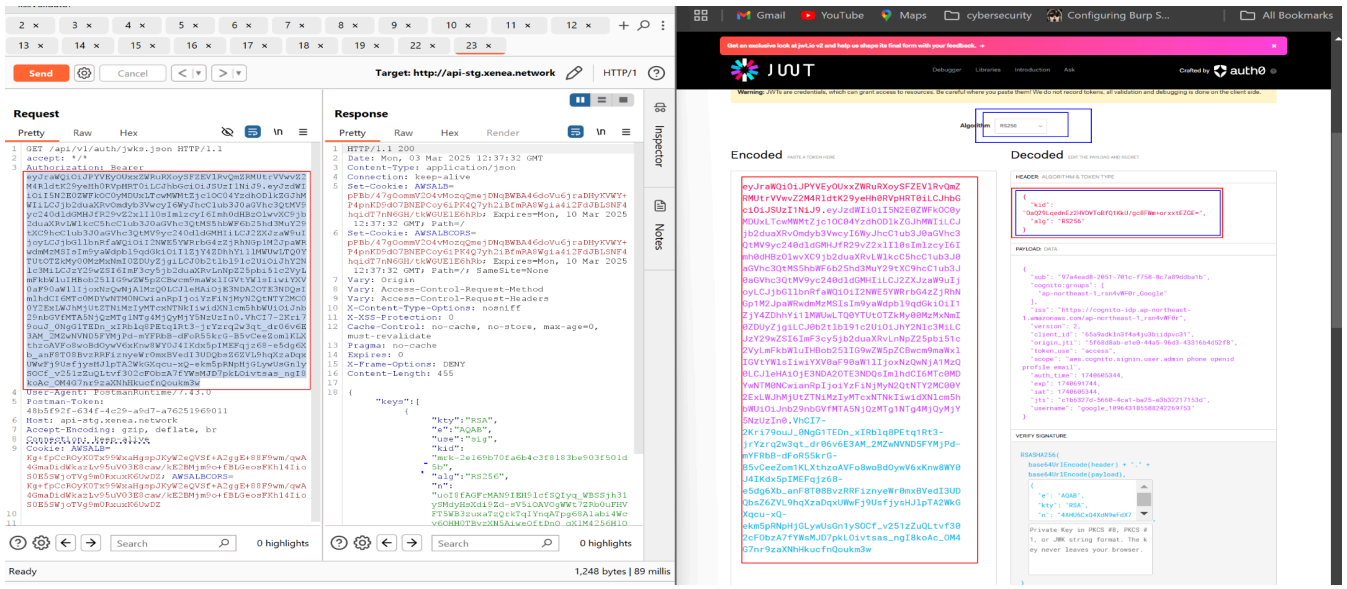
1. Intercept the API Request in Burp Suite. Send the Request to Burp Suite Repeater. got an access 200 ok

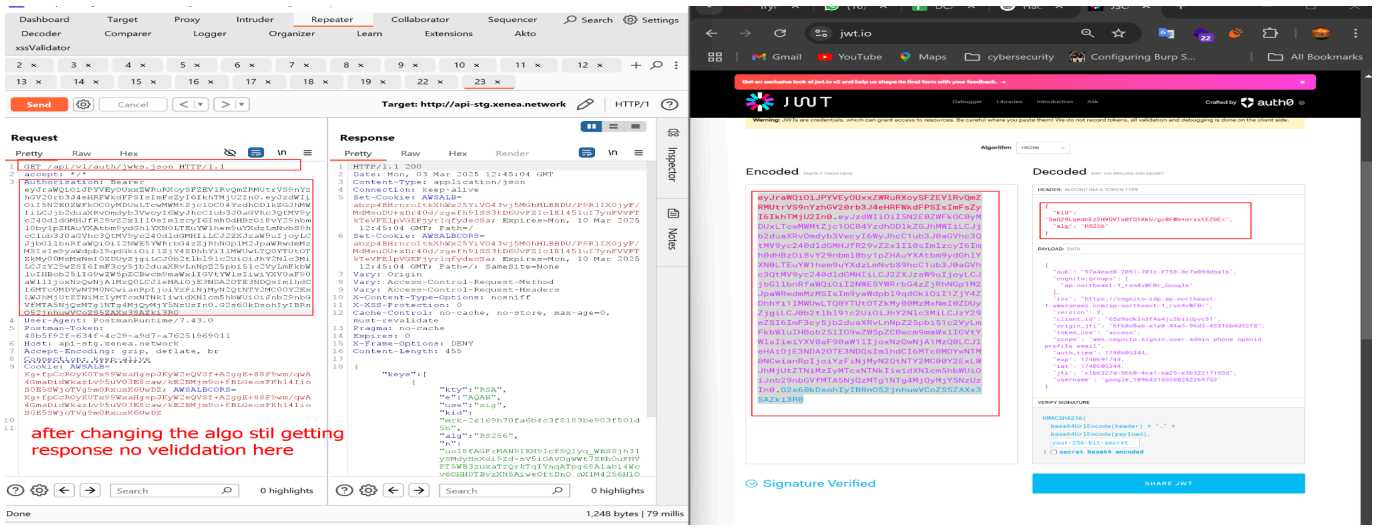


2. Try to login with no token then we will get a no_permission response.

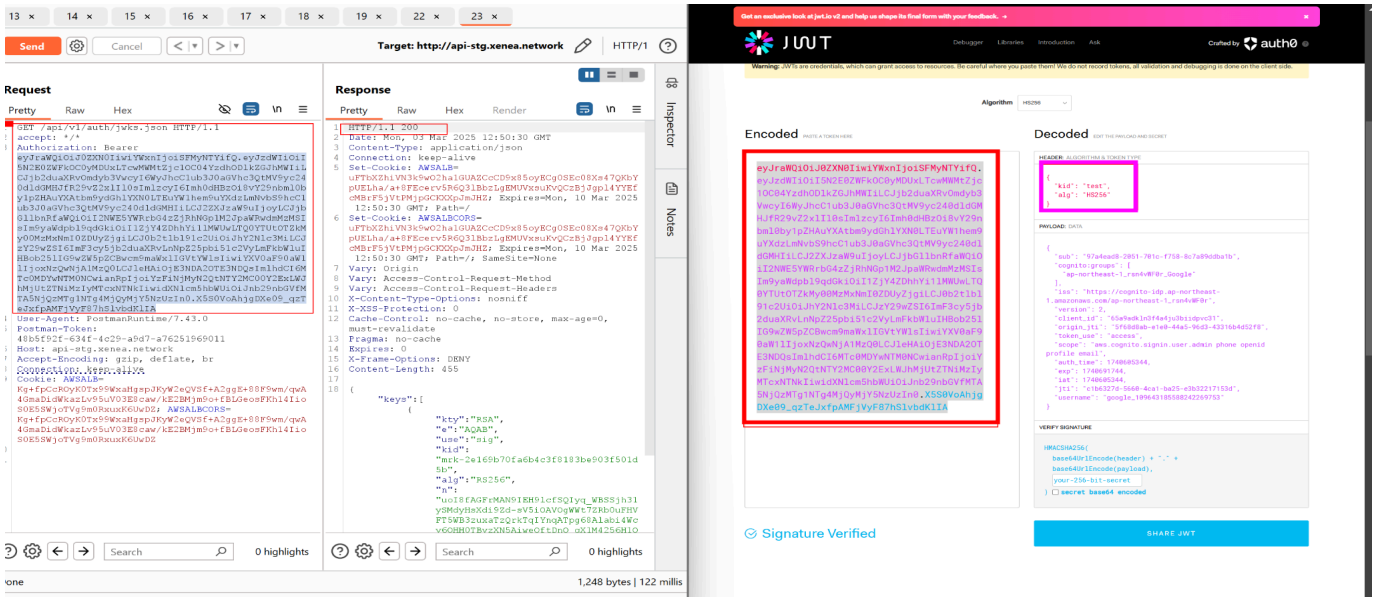


3. change algo from RS250 → HS256





4. change kid value to any string. we can modify other fields as well and we will get access.



5.36 Session token in URL

Vulnerability Severity	OWASP Category
High [redacted]	Session token in URL
Tools Used	**Date of reporting**
Burp Suite	11th March 2025

Vulnerability Observation

The session token is being passed in the URL parameter (accessToken or token). This can expose session data

Vulnerable location

URLs observed in the Burp Suite request:

1. GET /wallet/v1/authorize?inviterCode=&accessToken=<token>
2. POST /wallet/v1/notification/update-device-info?token=<token>

Technical Impact

Session Hijacking: Attackers can steal session tokens if exposed in logs or referrer headers.

Token Leakage: If the URL is shared or logged somewhere, an attacker can hijack the session.

Replay Attacks: The token can be reused by an attacker to gain unauthorized access.

Business impact

Data Breach: Attackers can take over user sessions, leading to account compromise.

Regulatory Violations: Storing session tokens in URLs can violate security best practices (e.g., OWASP, GDPR, PCI DSS).

Loss of Trust: Users may lose trust in the security of the application.

Remediation

- Use Secure Headers: Always send session tokens in the Authorization: Bearer header instead of the URL.
- Enforce Token Expiry: Implement short-lived session tokens with refresh tokens.

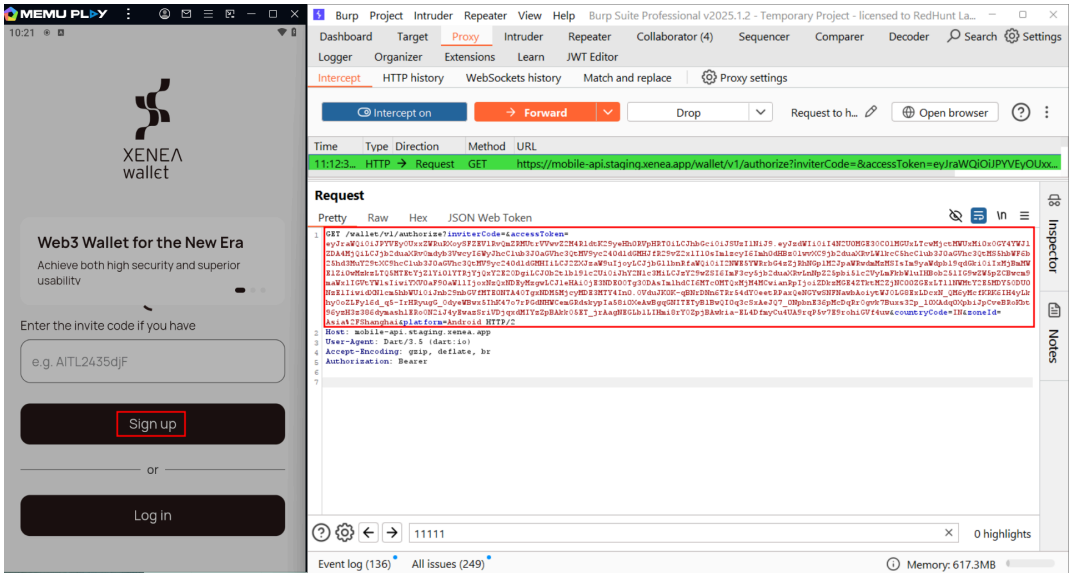
Status - Fixed

References

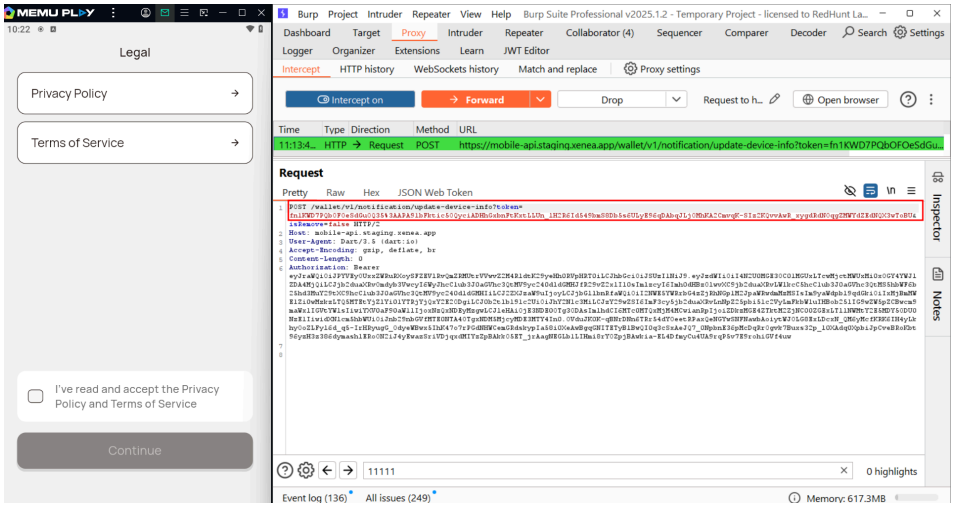
- OWASP Session Management Cheat Sheet
- Exposed Session Token

Steps to Reproduce

1. Capture Sign-Up/Login Request. Click on the **Sign Up** button. Capture the request in Burp Suite.



2. Send Request to Repeater. Forward the request and observe the response. Identify if the session token is passed in the URL.



5.37 Cross-Origin Resource Sharing (CORS) Vulnerable

Vulnerability Severity	**OWASP Category**
Medium	Cross-Origin Resource Sharing (CORS) Vulnerable
Tools Used	**Date of reporting**
Burp Suite	11th March 2025
Vulnerability Observation	
During testing, it was observed that the session token is being exposed in the URL when a user logs into the application. This was identified by capturing the request using Burp Suite while signing up and	

analyzing the request sent to the server. The session token was found embedded within the URL, which poses a significant security risk.

Vulnerable location

Resources Affected: https://mobile-api.staging.xenea.app/*

Technical Impact

Cross-Origin Data Theft: Attackers can host malicious websites that make authenticated requests to the vulnerable API and steal user data.
Session Hijacking: If an attacker exploits this on an API that uses session-based authentication (cookies), they could perform unauthorized actions on behalf of a logged-in user.
API Abuse: Attackers can load restricted API responses on their own domains and use them for malicious purposes (e.g., scraping, phishing).

Business impact

Loss of Sensitive Data: Confidential user data could be exposed to unauthorized third parties.
Compliance Violations: Exposure of personal or financial data may violate regulations like GDPR, CCPA, or PCI-DSS.
Reputation Damage: If exploited, customers may lose trust in the platform, impacting business operations.

Remediation

- Do not reflect the request's Origin header in the response dynamically. Instead, use a fixed, predefined list of allowed origins.
- Restrict cross-origin requests to only trusted domains by explicitly setting: Access-Control-Allow-Origin: https://trusted-website.com
- Disable Access-Control-Allow-Credentials: true unless strictly required, as it allows cookies to be sent across origins.
- Use a Content Security Policy (CSP) to prevent unauthorized JavaScript execution that exploits CORS issues.
- Conduct security testing regularly to detect misconfigurations in CORS settings.

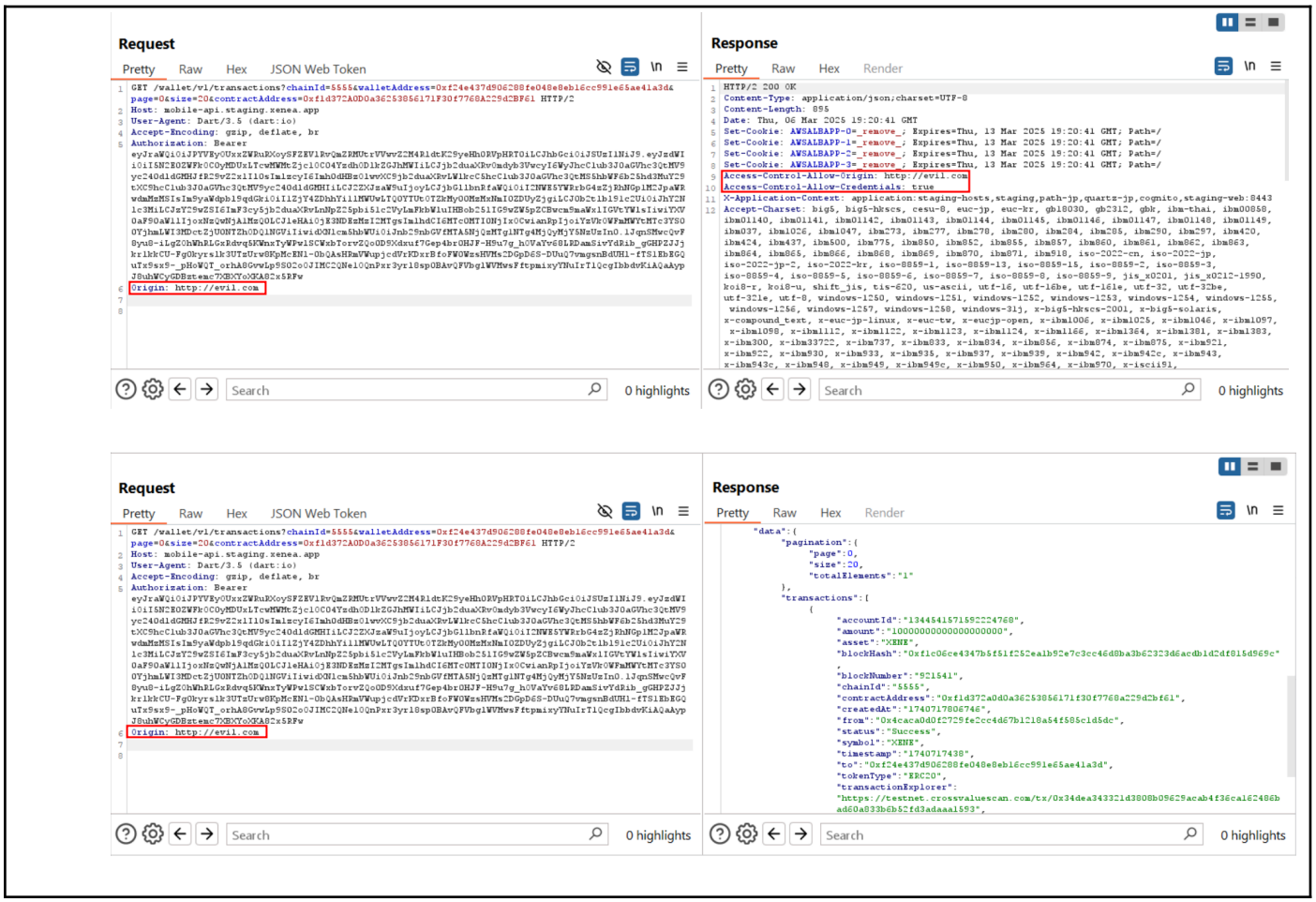
Status - Fixed

References

- cors
- Cross-Origin Resource Sharing (CORS)

Steps to Reproduce

1. Add the Origin: https://evil.com header in the request and observe its response that reflects back in the response in Access-Control-Allow-Origin.



5.38 Disabled X-XSS-Protection Header Leading to Increased XSS Risk

Vulnerability Severity	**OWASP Category**
Low	Vulnerable and Outdated Components
Tools Used	**Date of reporting**
Burp Suite	11th March 2025

Vulnerability Observation

During testing, it was observed that the **X-XSS-Protection: 0** response header, which disables the browser's built-in XSS filter. This could expose users to cross-site scripting (XSS) attacks by preventing the browser from blocking malicious script injections.

Vulnerable location

Throughout the url: http://api-stg.xenea.network/*

Technical Impact

Disabling XSS filters increases the attack surface for malicious payloads to execute in users' browsers.

Potential exploitation of stored or reflected XSS attacks, leading to unauthorized actions on behalf of users.

Increased risk of phishing or credential theft through injected scripts.

Business impact

Potential risk of data theft, session hijacking, and reputation damage if attackers exploit XSS vulnerabilities.
Compliance issues with security best practices, which could impact trust and credibility.
Increased likelihood of regulatory non-compliance (e.g., GDPR, PCI DSS) due to weak security measures.

Remediation

Enable the browser's XSS protection by setting the following response header:
X-XSS-Protection: 1; mode=block
This ensures that if an XSS attack is detected, the browser will block the response entirely instead of attempting to sanitize it.

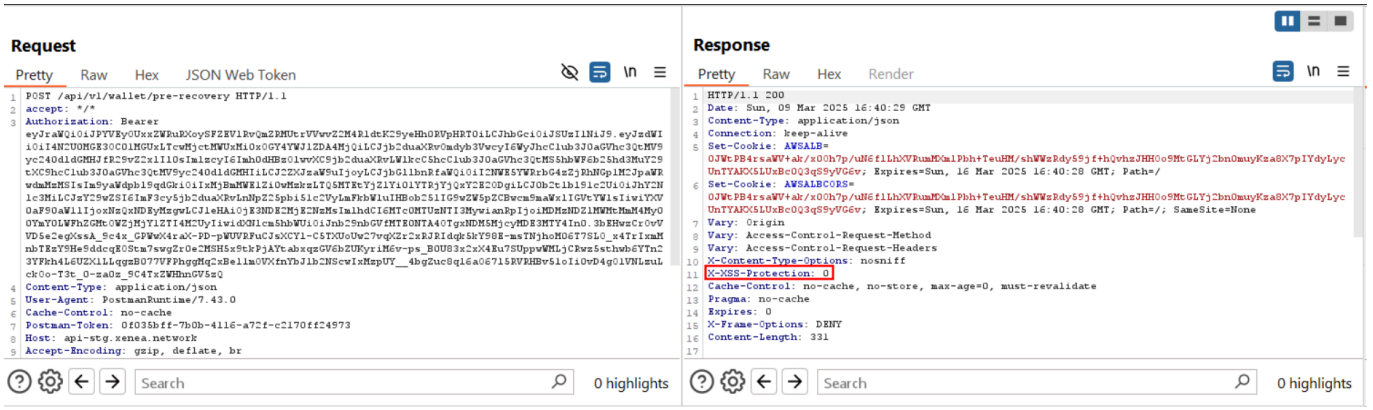
Status - Fixed

References

- reducing-risk-of-xss-with-modern
- CSP

Steps to Reproduce

1. See the response, there is response header X-XSS-Protection: 0



5.39 Session active after logout

Vulnerability Severity	OWASP Category
Medium	Session active after logout
Tools Used	**Date of reporting**
Burp Suite	10th March 2025

Vulnerability Observation

During testing, it was observed that JWT tokens remain valid even after the user logs out. This means an attacker or unauthorized user can reuse a previously captured token to access the application for 24 hours, leading to session hijacking and persistent access. A secure authentication system should invalidate session tokens upon logout to prevent unauthorized access. The failure to do so allows

attackers to maintain access even after the user logs out, increasing the risk of account takeover and data compromise.

Vulnerable location

Resources Affected:

https://mobile-api.staging.xenea.app/wallet/v1/nfts?chainIds=5555&chainIds=11155111&chainIds=97&chainIds=80002&address=0xf24e437d906288fe048e8eb16cc991e65ae41a3d&searchName=test&sortType=OLDEST_RECEIVED and throughout the mobile apis.

Affected API Endpoint: Authorization: Bearer <JWT Token>

Vulnerable Parameter: JWT Token used for authentication

Technical Impact

Session Hijacking: If a JWT token remains valid after logout, an attacker with access to the token can continue making authenticated requests.

Unauthorized Access: Users who log out but share a device (e.g., public or shared computers) remain vulnerable if the JWT is stored in local storage or cookies.

Privilege Escalation Risks: If an admin JWT is stolen and not invalidated on logout, it can be used indefinitely.

Business impact

Account Takeovers: If an attacker gains access to a valid JWT, they can impersonate the user.

Data Breaches: Persistent access may lead to exposure of sensitive user data.

Regulatory Non-Compliance: Keeping expired JWTs active may violate security compliance standards like GDPR, PCI-DSS, and OWASP recommendations.

Remediation

- Use a Token Blacklist or Revocation StrategyImplement SameSite Cookie Policy
- Implement Short-Lived Access Tokens with Refresh Tokens
- Remove JWT from Client-Side Storage on Logout
- Use jti (JWT ID) to Track and Revoke Tokens
- Implement Server-Side Session Management

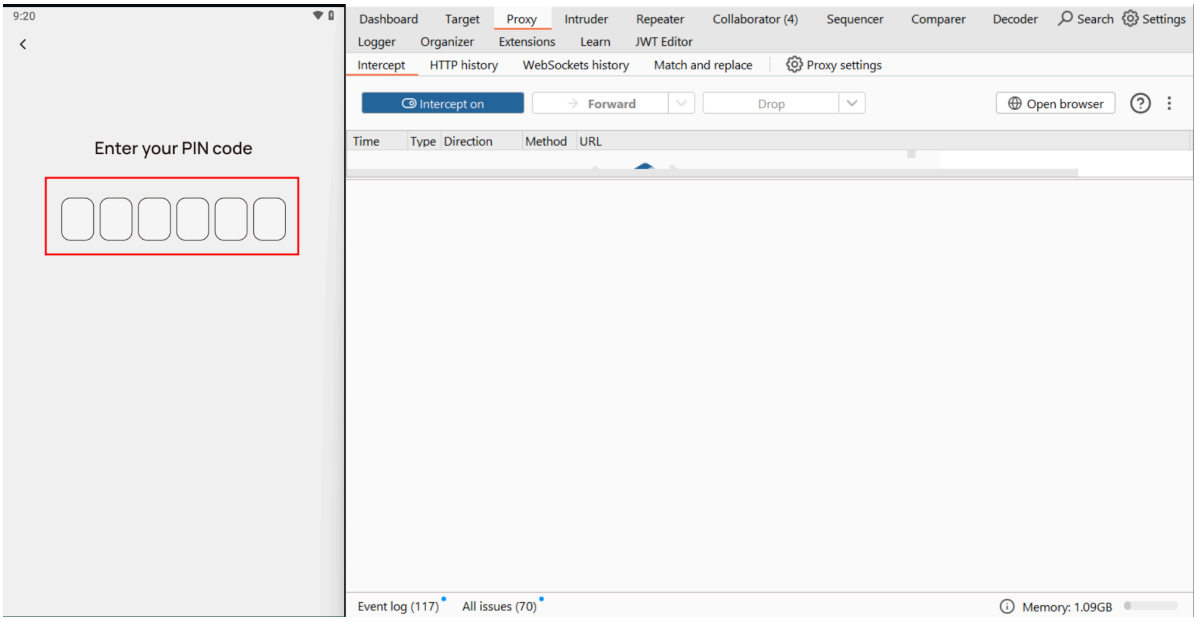
Status - Fixed

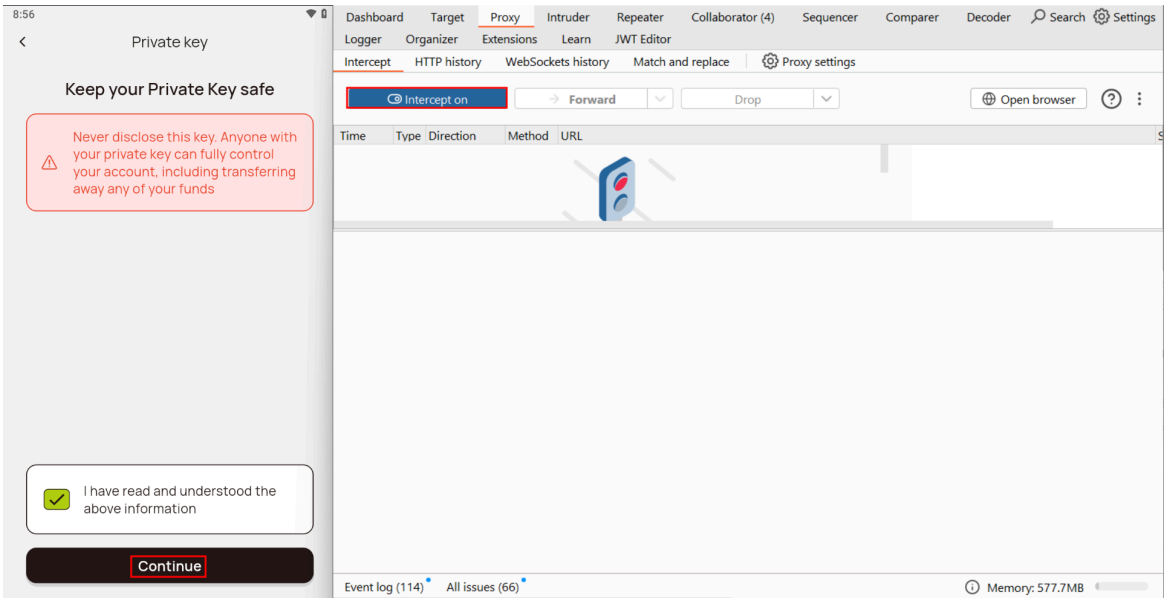
References

- OWASP Session Management Cheat Sheet
- OWASP API Security Top 10
- RFC 7519 – JSON Web Token (JWT)
- OWASP Authentication Cheat Sheet

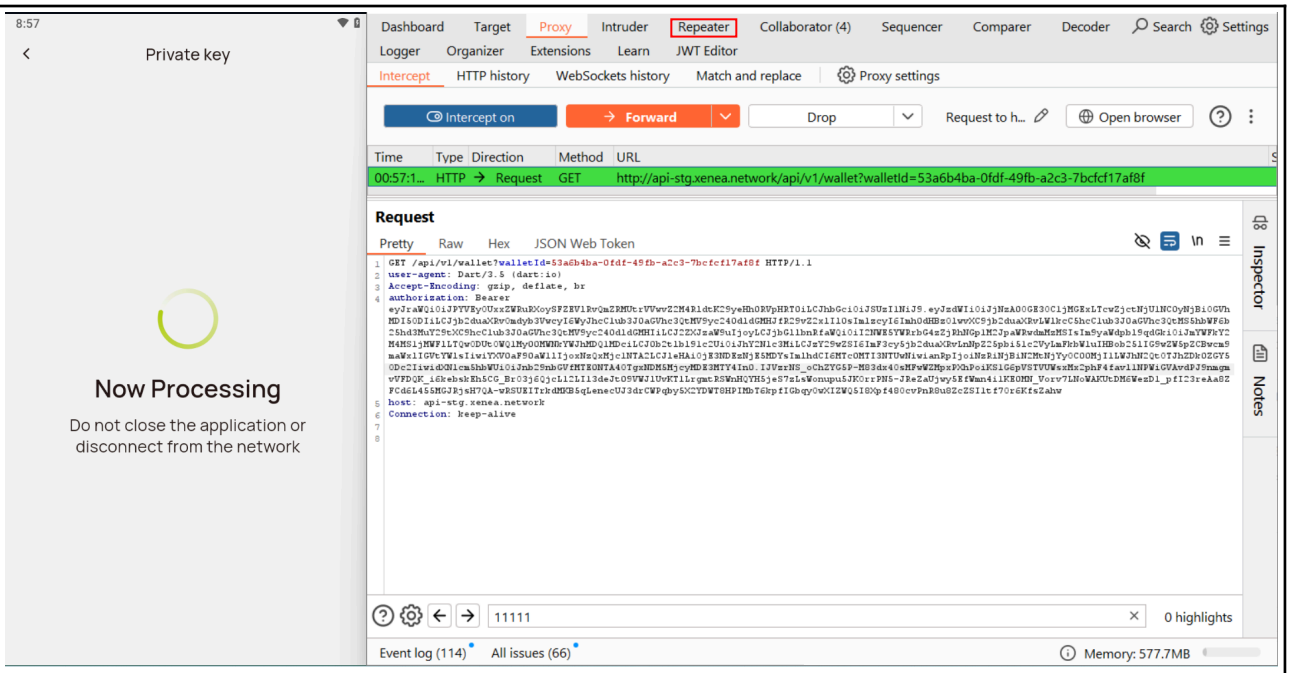
Steps to Reproduce

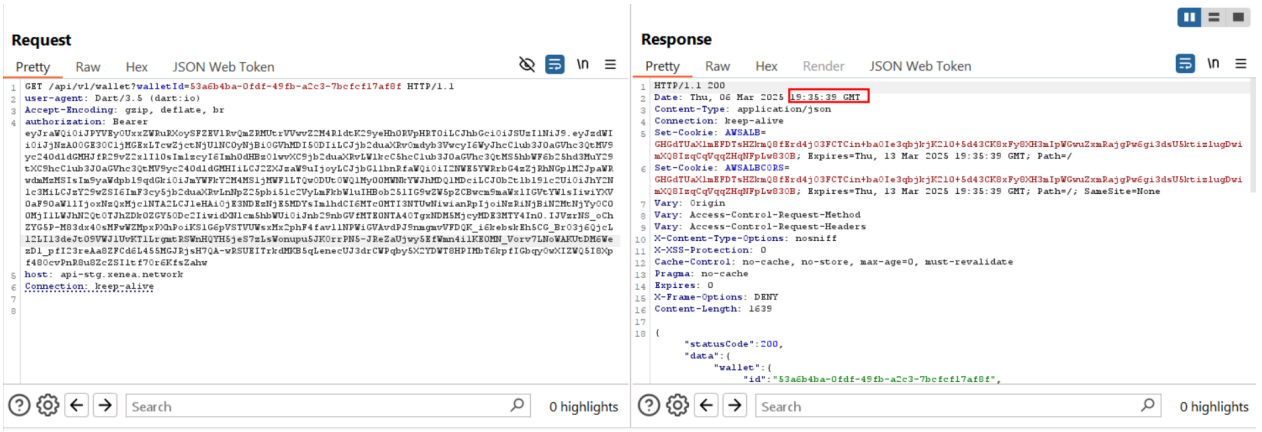
1. Login to the Application – Capture an API request containing the JWT token.



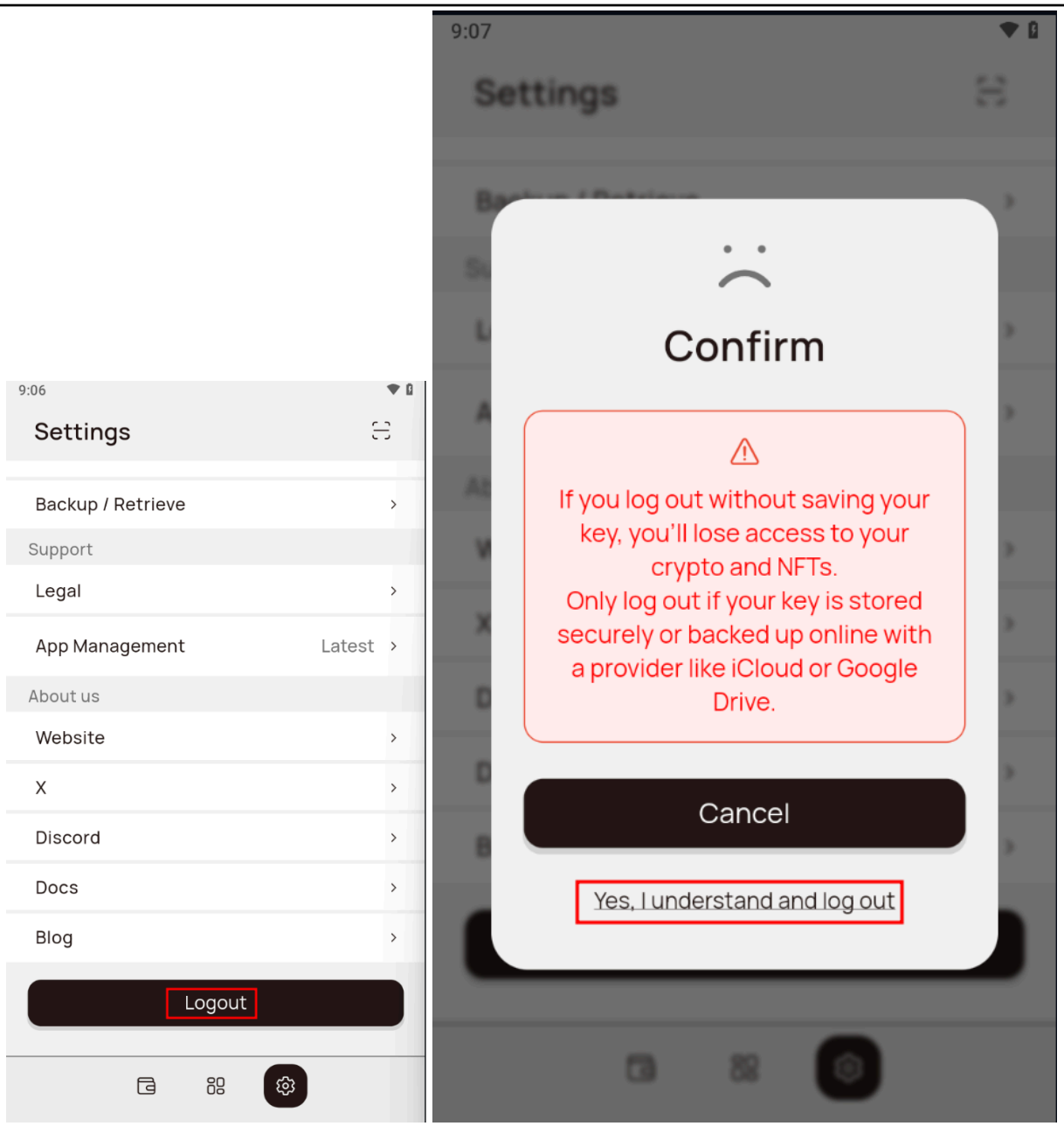


2. Send Request to Repeater – Identify the Authorization: Bearer <JWT Token> header and forward it to Burp Repeater.

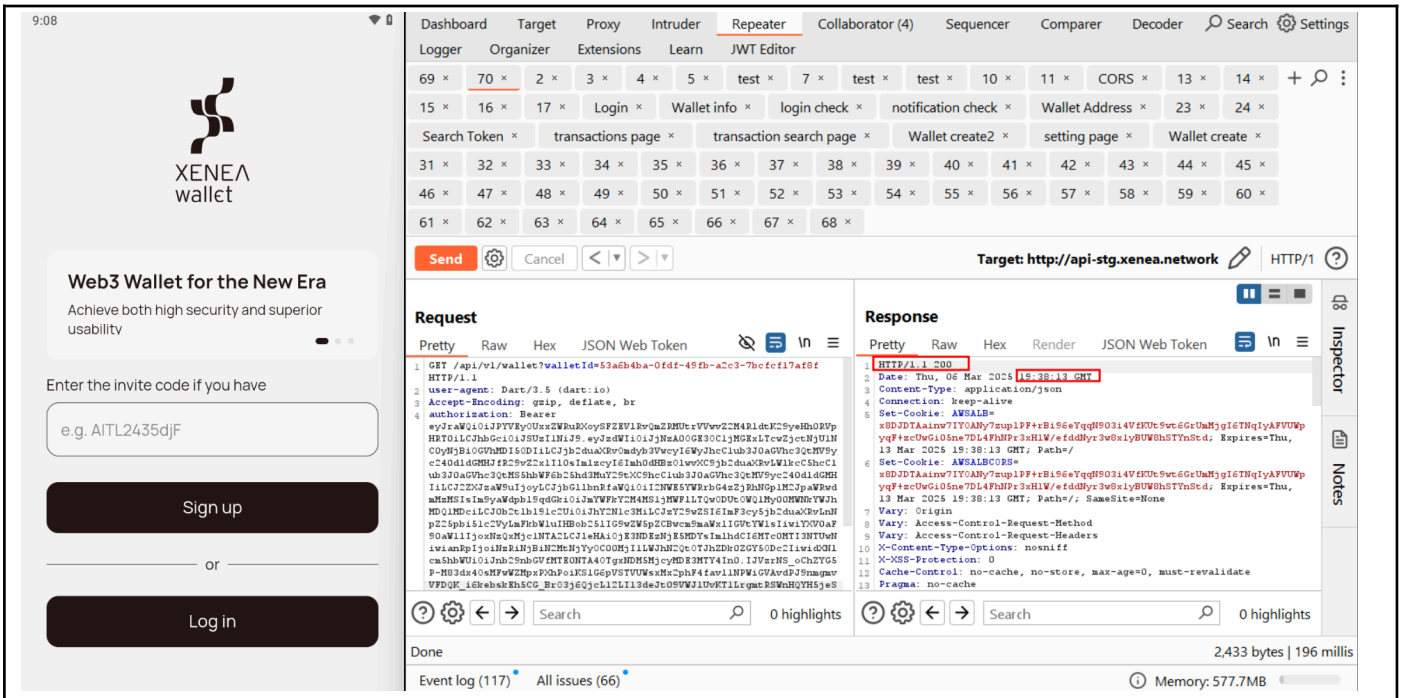




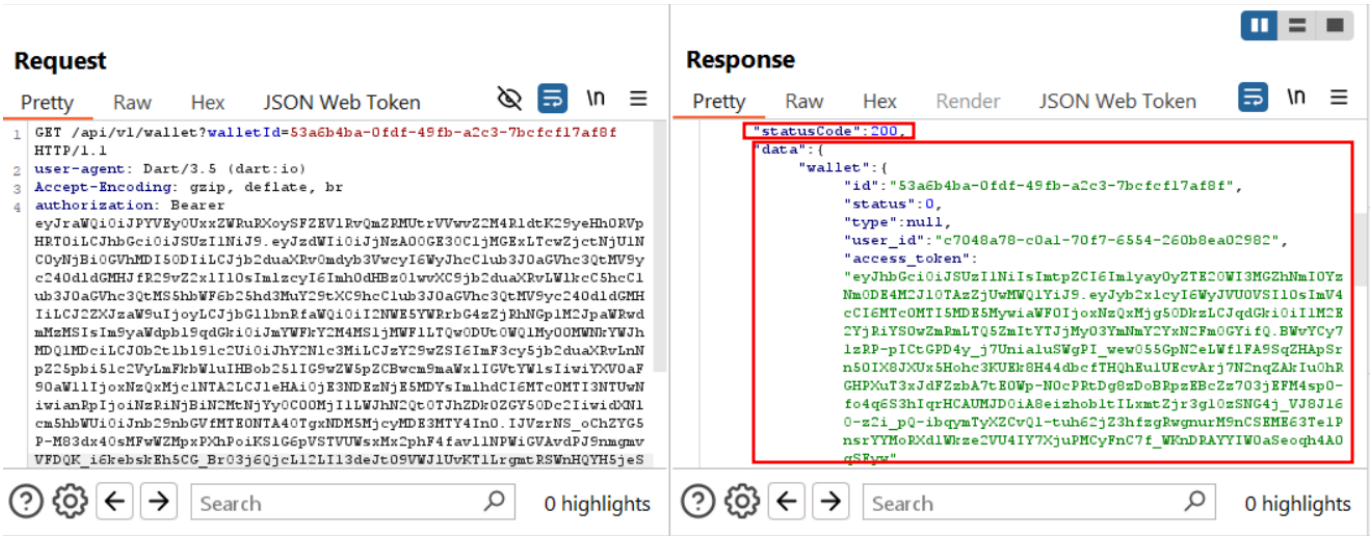
3. Logout from the Application – Log out and verify the session is closed.



4. Resend the Captured Request – Use Burp Repeater to send the old session token again.



5. data is still accessible, the session is still active despite logging out.



5.40 Improper use of Printstacktrace function

Vulnerability Severity	OWASP Category
Medium	Extraneous Functionality
Tools Used	**Date of reporting**
jadx-gui	11th March 2025
Vulnerability Observation	

During analysis, it was observed that the application used insecure functions and methods like, "printStackTrace()".

Vulnerable location

The vulnerability is present throughout the xenea-wallet.apk

Technical Impact

This method reveals the errors of the application. Error disclosures of applications help an attacker in getting specific information on the applications being used in the network. This would enable the attacker to concentrate more on the vulnerabilities of that application.

Business impact

There is no direct business impact of this vulnerability. However, the information disclosed can be used for further exploitation which may lead to an impact on the business.

Remediation

Remove and not use the stack trace method in the application.

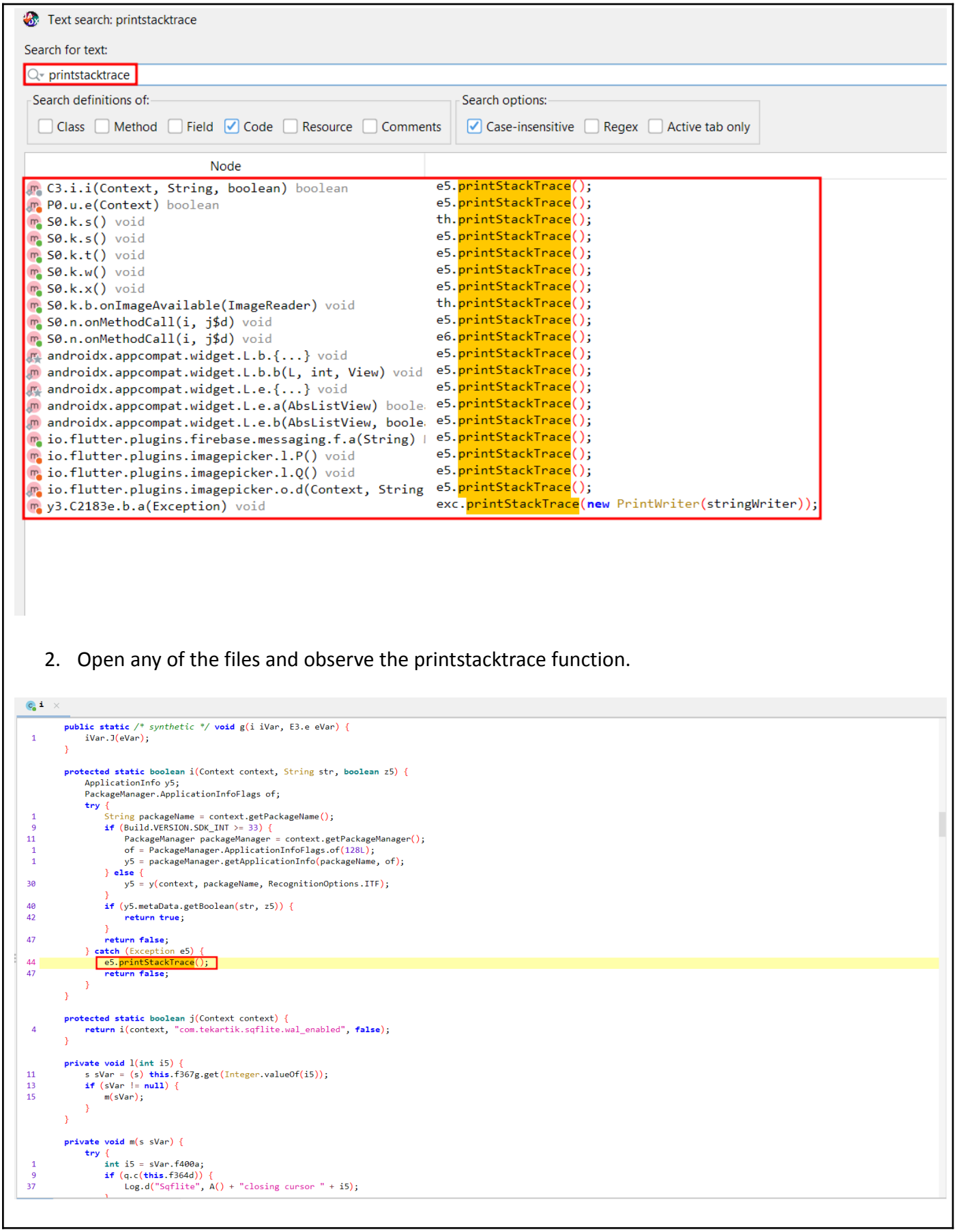
Status - Acknowledge

References

- Why is exception.printStackTrace() considered bad practice?
- Why is printStackTrace() a security risk for mobile (Android) applications?
- Remove printStackTrace method calls from the source code

Steps to Reproduce

1. Open the apk in the jadx-gui and search “printstacktrace”.



2. Open any of the files and observe the printstacktrace function.

8. Summary of Recommendations

To mitigate the identified vulnerabilities and enhance the security posture of Xenea Wallet, the following recommendations should be implemented:

1. Secure Communication & Application Integrity

- Implement SSL Pinning correctly to prevent Man-in-the-Middle (MITM) attacks.
- Strengthen root and emulator detection mechanisms to prevent attackers from running the app on compromised devices.
- Use Android Play Integrity API to detect tampering and unauthorized modifications.
- Implement anti-hooking techniques to prevent attackers from modifying app behavior at runtime.

2. Authentication & Authorization Hardening

- Enforce strong authentication mechanisms, such as multi-factor authentication (MFA).
- Implement account lockout mechanisms after multiple failed login attempts.
- Enforce session expiration and invalidation after logout to prevent unauthorized access.
- Prevent concurrent logins from multiple devices unless explicitly required.

3. Secure Storage & Data Protection

- Store sensitive data in encrypted form using AES-GCM (256-bit encryption) instead of weak algorithms like MD5 and SHA-1.
- Do not store sensitive data in local storage, logs, or shared preferences. Use Android Keystore System for cryptographic keys.
- Prevent backup of sensitive application data by setting android:allowBackup="false".
- Implement data masking techniques for sensitive user information (e.g., account ID, email, and private keys).

4. Secure API & Server-Side Controls

- Implement proper authorization checks for all sensitive API endpoints to prevent IDOR attacks.
- Restrict access to the admin API by implementing role-based access control (RBAC).
- Implement rate limiting to prevent brute-force attacks and DoS attacks.
- Use JWT best practices (e.g., secure storage, token expiration, and proper signing with HS256/RS256).
- Ensure public JWKS keys are not exposed to prevent cryptographic attacks.

5. Secure Cryptography & Input Validation

- Replace weak encryption algorithms (MD5, SHA1) with AES-GCM (256-bit) and SHA-256/SHA-512 for hashing.
- Implement input validation and sanitization to prevent SQL Injection, XSS, and template injection attacks.
- Use parameterized queries and ORM to prevent SQL Injection.

- Implement output encoding to prevent cross-site scripting (XSS) attacks.

6. Secure Permissions & Component Exposure

- Remove unnecessary permissions such as READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, and MANAGE_EXTERNAL_STORAGE.
- Disable Android export flag for sensitive activities and services to prevent unauthorized access.
- Implement secure deep link handling to prevent unauthorized navigation and intent hijacking.
- Restrict broadcast receivers and ensure file writes are properly secured to prevent arbitrary file manipulation.

7. Secure Logging & Debugging Controls

- Disable debugging symbols in production builds to prevent attackers from extracting sensitive information.
- Remove printStackTrace() and other debugging functions that expose internal errors.
- Ensure sensitive data (tokens, passwords, API keys) are never logged.
- Implement secure logging practices by using log sanitization and log rotation techniques.

8. Secure Cookies & Headers

- Implement security headers such as:
    - Content-Security-Policy (CSP)
    - Strict-Transport-Security (HSTS)
    - X-Content-Type-Options: nosniff
    - X-Frame-Options: DENY
    - X-XSS-Protection: 1; mode=block
- Configure cookies with HttpOnly, Secure, SameSite=Strict attributes to prevent session hijacking.

9. Secure Deployment & Monitoring

- Implement Web Application Firewall (WAF) to protect against common web attacks.
- Conduct regular security testing (penetration testing, code reviews, and SAST/DAST scans).
- Continuously monitor logs and security events to detect anomalies.
- Automate security updates and patches for both backend and frontend applications.

By implementing these recommendations, Xenea Wallet can significantly enhance its security posture, protect user data, and reduce the risk of exploitation by attackers.

Disclaimer

We conducted our comprehensive review of the project, including smart contract codes, frontend, and backend components, based on the materials and documentation provided by the project under audit (the "Project").

Our audit was primarily conducted through manual inspection by our expert team, with supplementary assistance from Artificial Intelligence (AI) systems. These AI systems incorporate (i) a database of known vulnerability patterns that we have collected up until the date of our review for this audit, and (ii) results from a selection of existing analysis tools available as of the date of our review for this audit. While we endeavor to ensure the highest possible quality and accuracy in our manual review process, and the results produced by our supplementary AI tools, we must clarify that absolute completeness and infallibility cannot be guaranteed.

Our manual audits are grounded in the knowledge and expertise we have accumulated up to the date of our review for this audit. The purpose of these audits is to identify and assess vulnerabilities specific to all components of the project under audit.

Blockchain and software technologies continue to evolve and may be susceptible to unforeseen risks and flaws. Consequently, while it is possible to minimize security risks, their complete elimination is inherently unattainable. Therefore, our audit does not claim to provide an exhaustive or all-encompassing review of all potential vulnerabilities.

The scope of our audit includes the analysis of smart contracts, frontend interfaces, and backend systems. However, our audit may not extend to other layers or components, including but not limited to hardware, operating systems, programming languages, compilers, protocols, platforms, virtual machines, and imported libraries, unless explicitly stated otherwise.

DISCLAIMER: You agree to the terms set forth in this disclaimer by reading this report or any part of it. Bunzz Pte. Ltd. and its affiliates (including shareholders, subsidiaries, employees, directors, and other representatives if any) ("Bunzz") provide this report for information purposes only. No party, including third parties, has the right to rely on this report or its contents. Bunzz assumes no responsibility or duty of care to any person and makes no warranty or representation about the accuracy or completeness of this report to any person. Bunzz provides this report "as-is," without any conditions, warranties, or other terms of any kind, and hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, fitness for purpose, merchantability, or non-infringement). Except to the extent that it is prohibited by law, Bunzz excludes all liability and responsibility, and you or any other person will not have any claim against Bunzz for any amount or kind of loss or damage that may result to you or any other person (including, without limitation, any direct, indirect, special, punitive, consequential, or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, whether in tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report, its use, the inability to use it, the results of its use, and any reliance on this report. For the avoidance of doubt, no one should consider or rely upon this report, its content, access, or usage as any form of financial, investment, tax, legal, regulatory, or other advice.