QuillAudits

Audit Report
April, 2024

For

Table of Content

Table of Content

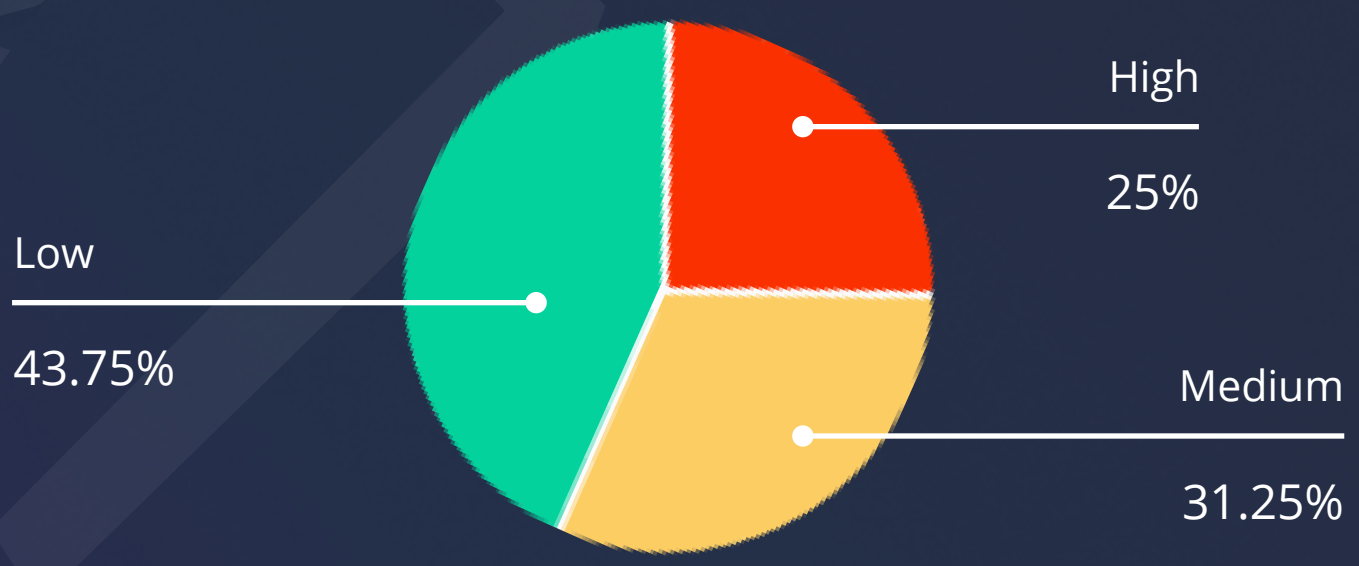
Overview

Scope of Audit	The scope of this pentest was to analyze the OronAndroid App for quality, security, and correctness.
In Scope	com.example.oron
Initial Review	20th February 2024 - 28th February 2024
Updated APK Received	16th April 2024
Final Review	17th April 2024 - 18th April 2024

Number of Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	2	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	4	5	5	0

Security Chart



Checked Vulnerabilities

We scanned the application for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Improper Authentication
- Improper Resource Usage
- Improper Authorization
- Insecure File Uploads
- Insecure Direct Object References
- Client-Side Validation Issues
- Broken Access Controls
- Insecure Cryptographic Storage
- Insufficient Cryptography
- Insufficient Session Expiration
- Information Leakage
- Third-Party Components
- Rate Limit
- Input Validation
- Injection Attacks
- Cross-Site Request Forgery
- Broken Authentication and Session Management
- Insufficient Transport Layer Protection
- Malware
- Denial of Service (DoS) Attacks
- Cross-Site Scripting (XSS)
- Security Misconfiguration
- Unvalidated Redirects and Forwards

And more...

Techniques and Methods

Throughout the pentest of Oron web applications, care was taken to ensure:

- Information gathering – Using OSINT tools information concerning the web architecture, information leakage, web service integration, and gathering other associated information related to web server & web services.
- Using Automated tools approach for Pentest like Nessus, Acunetix etc.
- Platform testing and configuration.
- Error handling and data validation testing.
- Encryption-related protection testing.
- Client-side and business logic testing.

Tools and Platforms used for Pentest:

- Burp Suite
- DNSenum
- Dirbuster
- SQLMap
- Acunetix
- Neucli
- Nabbu
- Turbo Intruder
- Nmap
- Metasploit
- Horusec
- Postman
- Netcat
- Nessus and many more.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your web app can be exploited. Issues on this level are critical to the web app's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the web app code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Issues Found

High Severity Issues

1. Account Takeover Via IDOR

Description

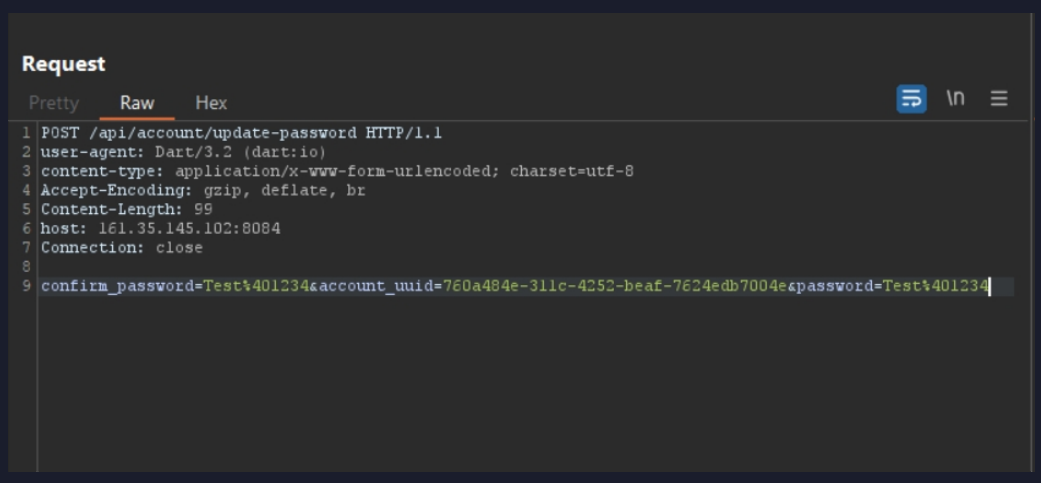
A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as mobile number, email, as a URL or form parameter. An attacker can manipulate mobile numbers to access another user mobile number without authorization unless an access control check is in place.

Vulnerable Endpoint

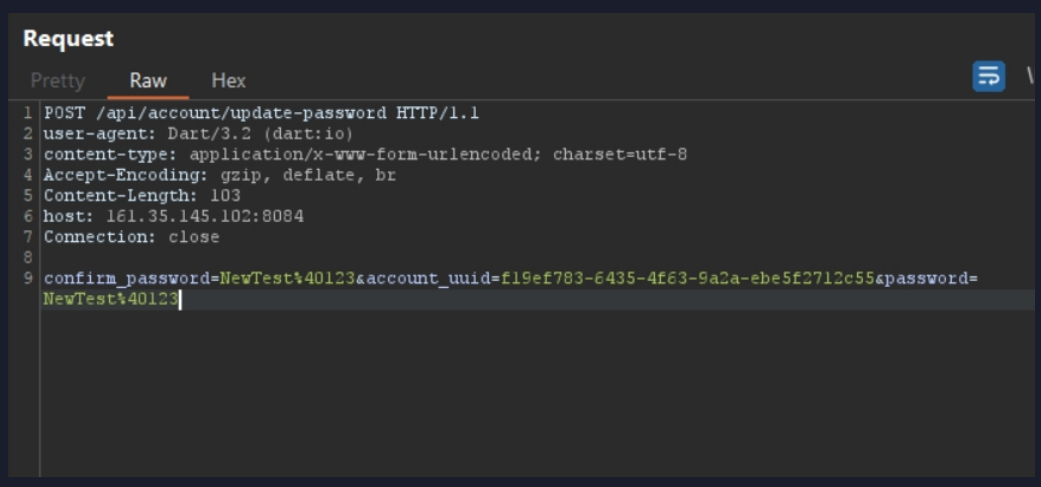
http://161.35.145.102:8084/api/account/update-password [account_uuid]

POC

Step 1: We triggered the change password module and intercepted the request through the application.



Step 2: We then replaced the ‘account_uuid’ parameter value with the victim uuid and forwarded the request.



Step 3: The password change was successfully for the victim account.

Step 4: As can be seen below, we were able to log in into the victim account using the above password.

Request:

Request

Pretty Raw Hex

```
POST /api/account/login HTTP/1.1
user-agent: Dart/3.2 (dart:io)
content-type: application/x-www-form-urlencoded; charset=utf-8
Accept-Encoding: gzip, deflate, br
Content-Length: 130
host: 161.35.145.102:8084
Connection: close

account_uuid=f19ef783-6435-4f63-9a2a-ebe5f2712c55&wallet_address=
0xb406703bb25ea9e9c9447d32d83ab8b6f334385f&password=NewTest%40123
```

Response:

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Content-Type: application/json
Connection: close
Cache-Control: no-cache, private
Date: Tue, 27 Feb 2024 14:32:06 GMT
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 56
Access-Control-Allow-Origin: *
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Content-Length: 531

{
  "status":true,
  "message":"Successfully login",
  "data":{
    "wallet_address":"0xb406703bb25ea9e9c9447d32d83ab8b6f334385f",
    "name":"Account 1",
    "qrcode_image_path":"storage\/wallet\/0xb406703bb25ea9e9c9447d32d83ab8b6f334385f.png",
    "notification":{
      "id":"03040395-fae7-4276-b997-086536bc23b6",
      "options":{
        "type":"basic",
        "iconUrl":".\/assets\/img\/notification.png",
        "title":"Wallet Login",
        "message":"Account 1 Successfully Login!",
        "priority":2,
        "requireInteraction":false
      },
      "action":{
```

Recommendation

To mitigate IDOR, implement access control checks for each object that users try to access. Web frameworks often provide ways to facilitate this. Additionally, use complex identifiers as a defense-in-depth measure, but remember that access control is crucial even with these identifiers.

Impact

The impact of an Account Takeover Via IDOR vulnerability can be severe and wide-ranging, including:

- **Data Manipulation or Deletion:** Attackers can manipulate or delete user data, leading to data loss or corruption. This can have serious consequences, especially if the application is used for critical purposes such as healthcare or emergency services.
- **Unauthorized Access to Sensitive Information:** Attackers can gain unauthorized access to sensitive information such as personal user data, financial records, or private messages by exploiting IDOR vulnerabilities. This can lead to breaches of privacy and confidentiality.
- **Identity Theft:** IDOR vulnerabilities can be exploited to steal user identities, which can then be used for various malicious activities such as fraud, phishing, or accessing other online accounts belonging to the user.

Status

Resolved

2. IDOR Leaking Secret Keys

Description

IDOR (Insecure Direct Object Reference) Leaking Secret Keys occurs when an application fails to properly enforce access controls, allowing an attacker to access resources or sensitive data belonging to other users. In this specific case, the attacker manipulates a parameter (such as 'wallet_address') in a request to access the recovery phrase codes of a victim's wallet. This unauthorized access exposes sensitive information, potentially compromising the security and confidentiality of the victim's wallet.

NOTE: Application-wide IDOR is present.

Vulnerable Endpoint

- http://161.35.145.102:8084/api/wallet/view-private-key [wallet_address]
- http://161.35.145.102:8084/api/account/view-recovery-phrase [wallet_address]
- http://161.35.145.102:8084/api/wallet/import [account_uuid]
- http://161.35.145.102:8084/api/account/check-recovery-phrase [account_uuid and secret_phrase]

- http://161.35.145.102:8084/api/account/update-password [account_uuid]

POC

Step 1: We triggered the view-recovery-phrase module and intercepted the below request.

Pretty Raw Hex

```
POST /api/account/view-recovery-phrase HTTP/1.1
user-agent: Dart/3.2 (dart:io)
content-type: application/x-www-form-urlencoded; charset=utf-8
Accept-Encoding: gzip, deflate, br
Content-Length: 128
host: 161.35.145.102:8084
Connection: close

wallet_address=0x37f72aaaac4ffaf723fa0df3cfee52b6e192f1ca&account_uuid=f19ef783-6435-4f63-9a2a-ebe5f2712c55&password=Test%401234
```

Step 2: We then changed 'wallet_address' parameter value to victim wallet address.

Pretty Raw Hex

```
POST /api/account/view-recovery-phrase HTTP/1.1
user-agent: Dart/3.2 (dart:io)
content-type: application/x-www-form-urlencoded; charset=utf-8
Accept-Encoding: gzip, deflate, br
Content-Length: 128
host: 161.35.145.102:8084
Connection: close

wallet_address=0xa37f1de768c39e2c1ec30ac932736ecbe717070d&account_uuid=f19ef783-6435-4f63-9a2a-ebe5f2712c55&password=Test%401234
```

Step 3: As can be seen below, we are able to view the recovery phrase codes of victim wallet in response.

Response

Pretty Raw Hex Render

```
X-RateLimit-Remaining: 59
Access-Control-Allow-Origin: *
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Content-Length: 269

{
  "status":true,
  "message":"",
  "data":{
    "wallet_address":"0xa37f1de768c39e2c1ec30ac932736ecbe717070d",
    "recovery_phrase":[
      "narren",
      "oleron",
      "dickson",
      "tenais",
      "manquer",
      "russia",
      "aroused",
      "sinope",
      "quavant",
      "mismas",
      "toonde",
      "stuffed",
      "lustre",
      "reussit",
      "grains",
      "lheure"
    ]
  }
}
```

Recommendation

- **Implement Least Privilege Principle:** Limit the privileges granted to each user or entity to the minimum necessary for performing their tasks. This reduces the impact of a potential IDOR vulnerability by restricting access to only authorized resources.
- **Implement Proper Authorization Checks:** Ensure that the application properly verifies the user's authorization to access sensitive data or perform actions. Access controls should be enforced server-side to prevent unauthorized access to sensitive resources.

Impact

The impact of IDOR leaking secret keys can be severe, leading to unauthorized access to sensitive information. In the context of recovery phrase codes, attackers could gain full control over the victim's wallet and cryptocurrency assets. This can result in financial loss, identity theft, and reputational damage for both the victim and the affected organization. It is crucial to address IDOR vulnerabilities promptly and implement strong security measures to protect against such attacks.

Status

Resolved

3. No Rate Limit Implemented

Description

Rate limiting is the process of controlling traffic rate from and to a server or component. It can be implemented on infrastructure as well as on an application level. Rate limiting can be based on (offending) IPs, IP blocklists, geolocation, etc.

Vulnerable Endpoint

- http://161.35.145.102:8084/api/account/login
- http://161.35.145.102:8084/api/account/import
- http://161.35.145.102:8084/api/token/delete
- http://161.35.145.102:8084/api/account/create
- http://161.35.145.102:8084/api/account/check-recovery-phrase

POC

Step 1: We triggered the login request and intercepted the login API endpoint.

```
POST /api/account/login HTTP/1.1
user-agent: Dart/3.2 (dart:io)
content-type: application/x-www-form-urlencoded; charset=utf-8
Accept-Encoding: gzip, deflate, br
Content-Length: 128
host: 161.35.145.102:8084
Connection: close

account_uuid=760a484e-311c-4252-beaf-7624edb7004e&wallet_address=0x73a117914a295ebdcb9385d5fdd07a4293f86a15&password=Test%4012$$
```

Step 2: We send the login API endpoint to intruder and did the password brute force attack. As can be seen below, we were able to successfully brute force the password due to no rate limit.

Request	Payload	Status code	Error	Timeout	Length	Comment
33	32	200			451	
34	33	200			451	
35	34	200			873	

Request Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Content-Type: application/json
Connection: keep-alive
Cache-Control: no-cache, private
Date: Tue, 27 Feb 2024 11:16:32 GMT
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 25
Access-Control-Allow-Origin: *
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Content-Length: 531

{
  "status":true,
  "message":"Successfully login",
  "data":{
    "wallet_address":"0x73a117914a295ebdcb9385d5fdd07a4293f86a15",
    "name":"Account 1",
    "qrcode_image_path":"storage\/wallet\/0x73a117914a295ebdcb9385d5fdd07a4293f86a15.png",
    "notification":{
      "id":"eed1c694-82d9-4ab9-9568-a83f85866e7b",
      "options":{
        "type":"basic",
        "iconUrl":".\/assets\/img\/notification.png",
        "title":"Wallet Login"
```

Search

Recommendation

- Implement a limit on how often a client can call the API within a defined timeframe.
- Notify the client when the limit is exceeded by providing the limit number and the time at which the limit will be reset.
- Add proper server-side validation for query string and request body parameters, specifically the one that controls the number of records to be returned in the response.

Impact

The impact of this vulnerability includes, but is not limited to:

- **Brute Force Attacks:** Without rate limiting, attackers can launch brute force attacks with unlimited attempts to guess user credentials, such as passwords or authentication tokens. This increases the likelihood of successful unauthorized access to user accounts or sensitive data.
- **Resource Exhaustion:** Continuous and unrestricted requests from automated scripts or bots can exhaust server resources such as CPU, memory, and bandwidth, impacting the performance and availability of the application for all users.
- **API Abuse:** In the case of APIs (Application Programming Interfaces), lack of rate limiting can lead to abuse by malicious users or poorly coded scripts, resulting in excessive consumption of API resources, degradation of service quality, and potential financial losses for the organization.

Status

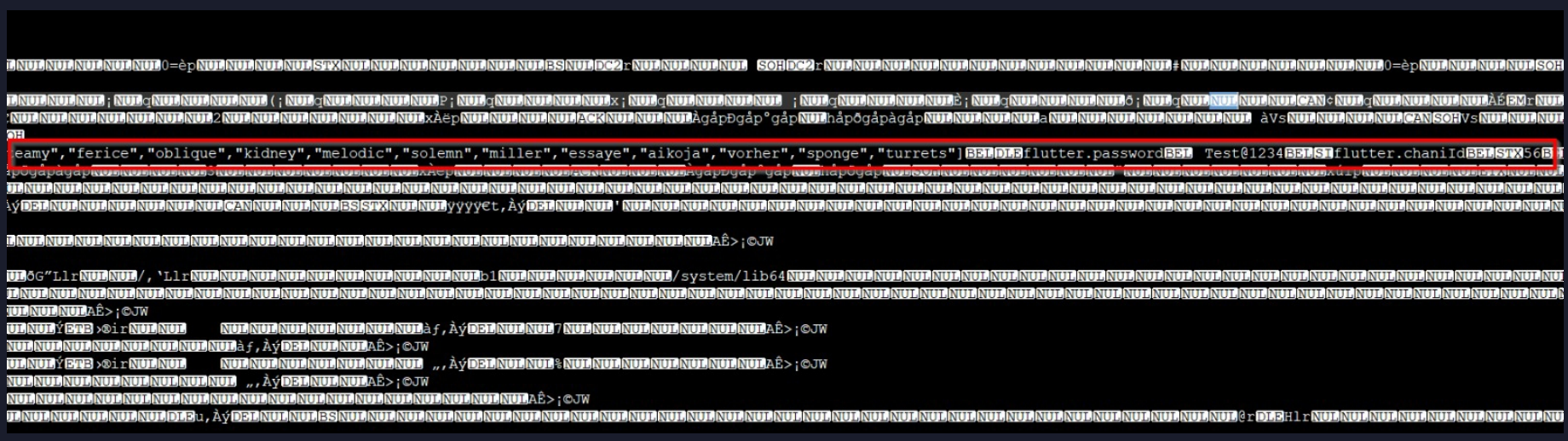
Resolved

4. Sensitive Information Leaked in Memory Dump

Description

Sensitive Information Leaked in Memory Dump is a critical security issue where confidential data, such as passwords or backup codes, is exposed within the memory dump of an application. Memory dumps can occur due to various reasons, including debugging tools, logging mechanisms, or vulnerabilities in the application's memory management.

POC



Recommendation

- **Avoid Storing Sensitive Information in Memory:** Ensure that sensitive data, such as passwords or backup codes, are not stored in plaintext in memory. Implement secure handling mechanisms, such as encryption or hashing, for sensitive data to prevent exposure in memory dumps.
- **Secure Memory Management:** Implement secure coding practices to minimize the risk of memory leaks or buffer overflows, which could lead to memory dumps containing sensitive information. Use memory-safe programming languages or libraries to reduce the likelihood of memory-related vulnerabilities.
- **Memory Protection Mechanisms:** Utilize memory protection mechanisms provided by the operating system or runtime environment, such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP), to mitigate the impact of memory-related attacks.
- **Sensitive Data Handling:** Implement secure storage and retrieval mechanisms for sensitive data, such as using secure key management systems and encryption algorithms. Avoid storing sensitive information in plaintext files or logs that may be accessible to unauthorized users.

Impact

- **Data Exposure:** The exposure of sensitive information, such as passwords and backup codes, can lead to unauthorized access to accounts or systems.
- **Identity Theft:** Attackers can use leaked sensitive information to impersonate users or gain unauthorized access to their accounts.
- **Financial Loss:** Leaked information, if used maliciously, can result in financial loss for individuals or organizations.
- **Reputation Damage:** The discovery of a memory leak exposing sensitive information can damage the reputation of an application or organization, eroding trust among users and stakeholders.
- **Regulatory Compliance Issues:** Failure to protect sensitive information in memory dumps can lead to legal and regulatory consequences for organizations, especially in industries with strict data protection requirements.

Status

Resolved

Medium Severity Issues

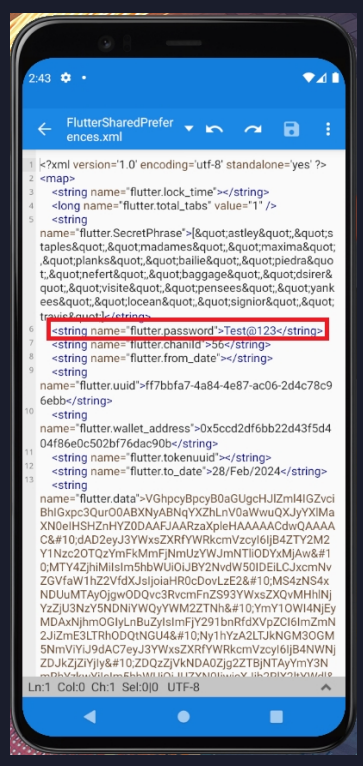
1. Storage of Sensitive Data in Shared Preferences

Description

Shared Preferences in Android is a simple key-value storage system, and if sensitive information like authentication tokens or private keys is stored here without proper encryption or security measures, it becomes vulnerable. Unauthorized access or rooted devices may expose this data, compromising user privacy and potentially leading to unauthorized access or misuse.

POC

As can be seen below, by browsing to /data/data/com.example.oron/shared_prefs/ FlutterSharedPreferences.xml file we were able to view password and other sensitive data.



Recommendation

Implementing secure storage mechanisms, such as encryption or utilizing secure storage APIs, is essential to safeguard sensitive information stored in Shared Preferences.

https://pub.dev/packages/flutter_secure_storage

Impact

Storing sensitive data in Shared Preferences exposes a security risk, as this data can be easily accessed by other applications or malicious entities. Unauthorized access may lead to privacy breaches, exposing sensitive user information.

Status

Resolved

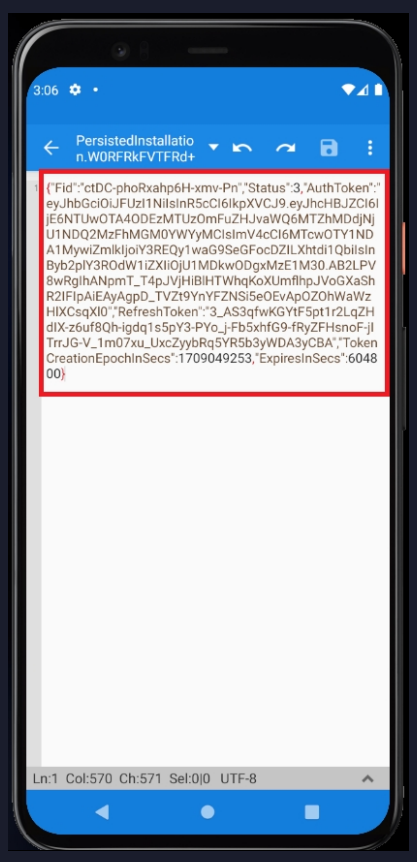
2. Sensitive Data Stored Locally

Description

Storing sensitive data locally on Android devices without proper encryption or protection introduces a notable security risk. This data, such as login credentials or private information, becomes vulnerable to unauthorized access through device compromise or malicious applications.

POC

As can be seen below, by browsing to /data/data/com.example.oron/files/ PersistedInstallation,WORFRK....json file we were able to view JWT token



Recommendation

Employing robust encryption and secure storage practices is essential to mitigate the risks associated with locally stored sensitive data on Android devices.

https://pub.dev/packages/flutter_secure_storage

Impact

In the event of unauthorized access or a device compromise, sensitive information, such as user credentials or personal data, becomes vulnerable to theft or misuse. This can lead to identity theft, unauthorized access to accounts, and potential legal and reputational consequences for both users and the application.

Status

Resolved

3. SSL Pinning not implemented

Description

Pinning effectively removes the "conference of trust". An application that pins a certificate or public key no longer needs to depend on others - such as DNS or CAs - when making security decisions relating to a peer's identity. This application doesn't implement SSL pinning and hence it was possible for the user to intercept the HTTPS request and response and interpret it clearly. Thus, an attacker can tamper meaningfully with the parameters and try to trigger logical flaws in the application.

POC

As can be seen below, we were able to intercept API calls.

Recommendation

SSL pinning should be implemented to prevent user from intercepting the request and response using MITM tool.

https://pub.dev/packages/ssl_pinning_plugin/versions

Impact

- Without SSL pinning, the application is susceptible to man-in-the-middle attacks.
- Attackers may intercept and manipulate communication between the app and the server.
- SSL pinning enhances security by ensuring that the app only trusts specific SSL/TLS certificates, mitigating the risk of unauthorized certificate interception

Status

Resolved

4. Missing Root / Emulator Detection

Description

The root / emulator detection mechanism in the target system fails to properly check for the presence of certain files or system configurations that are commonly associated with a rooted /emulator device. An attacker can exploit this vulnerability by modifying these files or configurations in a way that the root / emulator detection mechanism does not detect, allowing the attacker to gain elevated privileges on the device.

POC

As can be seen below, the application is running on rooted and emulator device.

```
generic_x86_64_arm64:/ # whoami
root
generic_x86_64_arm64:/ # ps -A | grep com.example.oron
u0_a161       11843    303 17297392 195136 do_epoll_wait      0 S com.example.oron
generic_x86_64_arm64:/ #
```

Recommendation

Implement checks for root detection & emulator detection.

Impact

An attacker who successfully exploits this vulnerability can gain elevated privileges on the device, potentially allowing them to access sensitive data, install malicious software, or perform other actions that would normally be restricted to a non-root user.

Status

Resolved

5. Sensitive Information Disclosure in Logs

Description

This vulnerability report identifies a security issue in the application's logging mechanism that results in the leakage of sensitive information including the App Pin, Auth Token, Private Key and potentially other confidential data to the system logcat. This vulnerability poses a significant risk to the confidentiality and integrity of user data and the application's security.

POC

Step 1: Connect the Device using adb.

Step 2: Run adb logcat | findstr “<PID of Oran App here>”

Step 3: As can be seen below, we were able to view the password and backup codes in Logs.

```
)2-27 21:48:38.892  5417  5417 I FLTFireMsgService: FlutterFirebaseMessagingBackgroundService started!
)2-27 21:48:42.053  5417  5483 D ProfileInstaller: Installing profile for com.example.oron
)2-27 21:48:56.180  5417  5461 I flutter : Test@123
)2-27 21:48:56.180  5417  5461 I flutter : Password: <optimized out>#b7037(TextEditingValue(text: ┤Test@123├, selection: TextSelection.invalid, composing: T
extRange(start: -1, end: -1)))
)2-27 21:49:11.585  5417  5461 I flutter : 151
)2-27 21:49:11.585  5417  5461 I flutter : ["astley","staples","madames","maxima","planks","bailie","piedra","nefert","baggage","dsirer","visite","pensees",
yankees","locean","signior","travis"]
)2-27 21:50:11.618  5417  5461 I flutter : hello
)2-27 21:50:13.283  5417  5461 I flutter : dataaaa: {status: true, message: Your wallet is successfully created, data: {wallet_address: 0xe663f5776943bad2ac
e3abf59b861200168f8b2, name: Account 1, qrcode_image_url: http://161.35.145.102:8084/storage/wallet/0xe663f5776943bad2ac6e3abf59b861200168f8b2.png, account
uuid: ff7bbfa7-4a84-4e87-ac06-2d4c78c96ebb}}
)2-27 21:50:13.283  5417  5461 I flutter : ran till here
```

Recommendation

Don’t print or store sensitive logs.

Impact

The impact of this vulnerability includes, but is not limited to:

- **Unauthorized Access:** Attackers with access to the device's logcat logs can potentially gain unauthorized access to user accounts and systems, as they have access to sensitive authentication information.
- **Data Exfiltration:** The leaked information can be used to exfiltrate sensitive data, such as user credentials, private keys, or access tokens, which can be exploited for malicious purposes.
- **Reputation Damage:** Discovery of such vulnerabilities can lead to a loss of trust among users and stakeholders, damaging the application's reputation and potentially resulting in legal consequences.

In this case, we were able view password, backup codes and user information.

Status

Resolved

Low Severity Issues

1. Insecure Random Number Generator

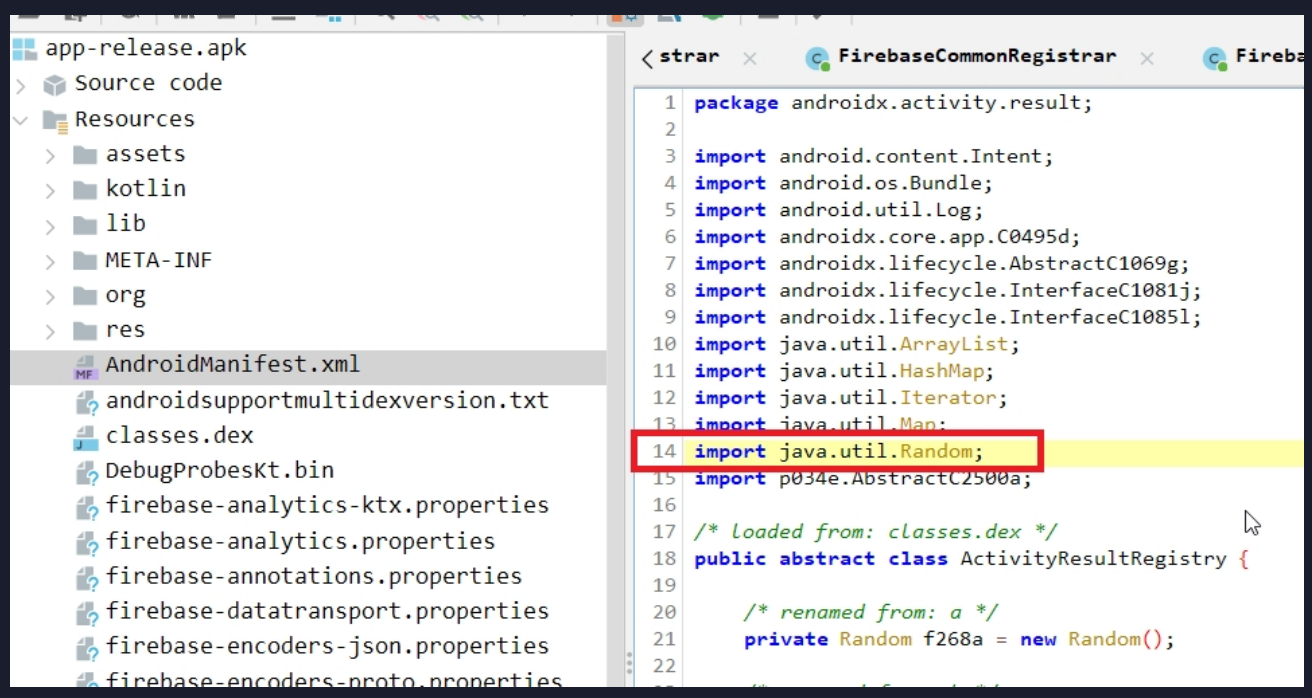
Description

Cryptography requires secure pseudo random number generation (PRNG). Standard Java classes do not provide sufficient randomness and in fact may make it possible for an attacker to guess the next value that will be generated and use this guess to impersonate another user or access sensitive information.
Most developers should instantiate SecureRandom via the default constructor without any arguments. Other constructors are for more advanced uses and, if used incorrectly, can lead to decreased randomness and security.
In general, if a PRNG is not advertised as being cryptographically secure (e.g. java.util.Random), then it is probably a statistical PRNG and should not be used in security-sensitive contexts. Pseudo-random number generators can produce predictable numbers if the generator is known, and the seed can be guessed. A 128-bit seed is a good starting point for producing a "random enough" number.

POC

Step 1: Open the APK in Jadx and search for java.util.Random Library.

Step 2: As can be seen here, java.util.Random is in use for random number generation.



Recommendation

It is recommended to use Java's SecureRandom class to generate a cryptographically strong pseudo-random number (DO THIS):

```
public static int generateRandom(int maximumValue) {
    SecureRandom ranGen = new SecureRandom();
    return ranGen.nextInt(maximumValue);
}
```

Impact

An insecure random number generator can have significant security consequences. It undermines the integrity of cryptographic systems and increases the risk of predictable patterns or vulnerabilities in applications. Security protocols relying on random numbers, such as encryption keys or session tokens, become susceptible to exploitation. This weakness may lead to unauthorized access, data breaches, or compromised system integrity. Developers and security professionals should prioritize using robust and cryptographically secure random number generators to mitigate these risks.

Status

Acknowledged

2. Tapjacking Vulnerability

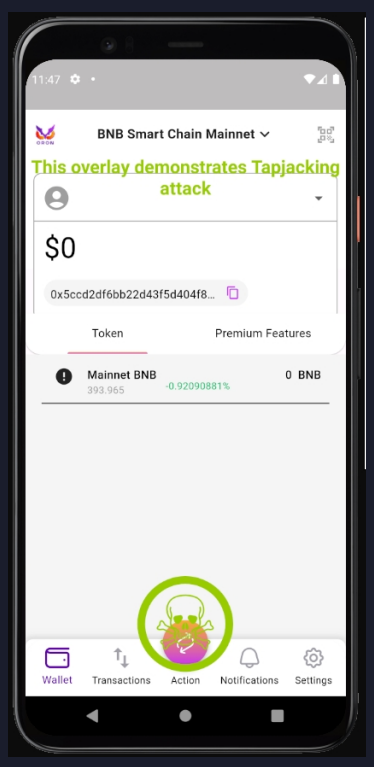
Description

Tapjacking is a security vulnerability in mobile devices where a malicious app overlays a transparent or disguised user interface element on top of legitimate app interfaces. This deceptive overlay can trick users into performing unintended actions, potentially compromising their security or privacy. The attacker takes advantage of the device's multitouch capabilities to capture the user's taps and interactions, which are then unknowingly applied to the obscured app beneath the overlay. This technique can lead to unauthorized actions, data theft, or unintended app behavior.

POC

Download the Tapjacker APK from GitHub and open the Oran application from Tapjacker application.

https://github.com/dzmitry-savitski/tapjacker



Recommendation

- **Avoid Exporting Sensitive Components:** Make sure your sensitive components like Activities, Broadcast Receivers, or Services are not unnecessarily exported. For example, if you have a Broadcast Receiver that is only intended for internal use, set the android:exported attribute to false in your manifest.

```
<receiver
  android:name="com.botree.productsfa.main.LoginActivity"
  android:exported="false">
  <!-- ... -->
</receiver>
```

- **Use Permissions:** For components that need to be exported, require a custom permission that only your app holds. This prevents other apps from sending broadcasts or interacting with your components.

```
<receiver
  android:name="com.botree.productsfa.main.LoginActivity"
  android:exported="true">
  <intent-filter>
     <!-- ... -->
  </intent-filter>
  <permission android:name="com.yourapp.PERMISSION" />
</receiver>
```

- **Secure Your UI:** Implement secure UI design practices to prevent overlays. Avoid drawing UI elements outside of your app's window boundaries. You can use the **FLAG_SECURE** flag to prevent screenshots and screen recording within sensitive activities.

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
                     WindowManager.LayoutParams.FLAG_SECURE);
```

Impact

Attackers can overlay deceptive UI elements on legitimate apps, tricking users into unintended actions. This can lead to unauthorized transactions, granting permissions, or compromising sensitive data, posing a significant risk to user privacy and security. Implementing safeguards, such as UI overlays prevention and user awareness, is crucial to mitigate this threat.

Status

Resolved

3. Cross Origin Resource Sharing Misconfigured

Description

Cross-origin resource sharing (CORS) is a mechanism that allows JavaScript on a web page to make XMLHttpRequests to another domain. CORS defines a way in which the browser and the server can interact to determine whether or not to allow the cross-origin request.

Vulnerable Endpoint

http://161.35.145.102:8084/*

Note: Whole application is vulnerable to Cross Origin Resource Sharing Misconfigured vulnerability.

Recommendation

It is recommended to that an Access-Control-Allow-Origin header should not allow all domains.

Impact

Insecure CORS policies can enable attackers to access sensitive data from other origins. By abusing CORS misconfigurations, attackers can steal user data, session tokens, or other sensitive information from trusted websites, leading to data breaches and privacy violations.

Status

Resolved

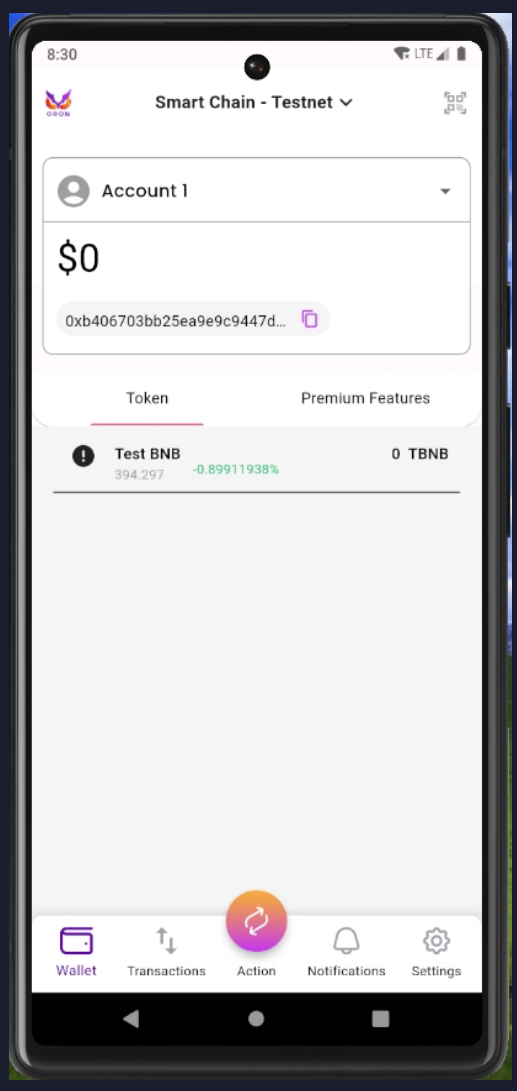
4. Concurrent Login Enabled

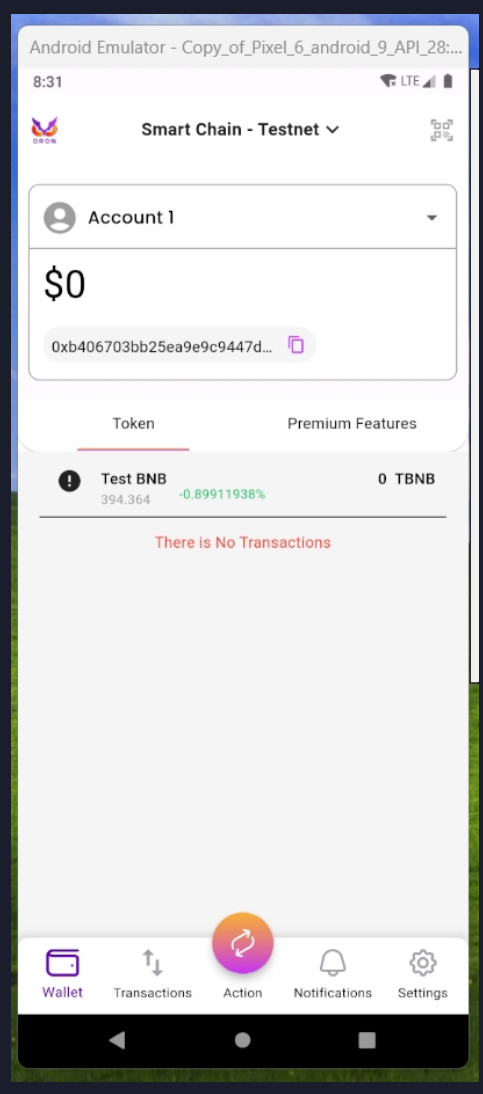
Description

Having concurrent sessions enabled on the application enables the user to have multiple sessions. An attacker who gets access to the login credentials may access the application make changes to the application and cause harm to the organization.

POC

As can be seen below, we were able to login in two different devices with same account at same time.





Recommendation

It's recommended to disable concurrent sessions for the same user.

Impact

Enabling concurrent logins facilitates account takeover attacks. Attackers can leverage stolen credentials to log in concurrently with the legitimate user, effectively taking control of the account and conducting malicious activities such as data theft, fraud, or unauthorized changes to account settings.

Status

Acknowledged

5. Failure to Logout User - On Password Reset

Description

Most application frameworks use sessions to manage the user's identity and access levels. It may be possible for a malicious user to use the existing session and log into the user's account.

Vulnerable Endpoint

http://161.35.145.102:8084/api/account/update-password

POC

Step 1: We logged in with two same accounts on two devices.

Step 2: From the first device we triggered password reset functionality.

Request:

Request

Pretty Raw Hex

```
1 POST /api/account/update-password HTTP/1.1
2 user-agent: Dart/3.2 (dart:io)
3 content-type: application/x-www-form-urlencoded; charset=utf-8
4 Accept-Encoding: gzip, deflate, br
5 Content-Length: 103
6 host: 161.35.145.102:8084
7 Connection: close
8
9 confirm_password=NewTest%40123&account_uuid=f19ef783-6435-4f63-9a2a-ebe5f2712c55&password=
  NewTest%40123
```

Response:

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  Server: nginx/1.18.0 (Ubuntu)
3  Content-Type: application/json
4  Connection: close
5  Cache-Control: no-cache, private
6  Date: Tue, 27 Feb 2024 14:29:49 GMT
7  X-RateLimit-Limit: 60
8  X-RateLimit-Remaining: 58
9  Access-Control-Allow-Origin: *
10 X-Frame-Options: SAMEORIGIN
11 X-Content-Type-Options: nosniff
12 Content-Length: 65
13
14 {
     "status":true,
     "message":"Account Password updated!",
     "data":null
   }
```

Step 3: As can be seen below, In second device the account is not logged out after password reset.

Recommendation

It is recommended that the existing session should be destroyed from server side once the change password or reset password functionality is triggered successfully.

Impact

The impact of this vulnerability can be severe and can lead to the following consequences:

- **Account Compromise:** Failure to invalidate sessions after a password change increases the risk of account compromise. Even if the user changes their password to strengthen security, any active sessions associated with the old password remain valid, allowing attackers to gain unauthorized access to the account and its sensitive information.
- **Data Breaches:** Active sessions associated with old passwords pose a risk of data breaches. Attackers with access to these sessions can exploit them to access sensitive data, manipulate account settings, or perform fraudulent activities, leading to potential data breaches and compromising user privacy.

Status

Resolved

6. Backup Is Not Set To False

Description

The setting for the flag **[android:allowBackup]** is recommended to be configured as "false." By default, it is set to "true," potentially permitting unauthorized access to your application data through the Android Debug Bridge (adb). When this flag is set to "true," users with USB debugging enabled can copy application data from the device, posing a security risk. Therefore, it is advisable to explicitly set **[android:allowBackup]** to "false" to enhance the security of your application data.

POC

Step 1: Decompile the Application

Step 2: Resources AndroidManifest.xml and Search for android:allowBackup="true"

Step 3: As can be seen below, android:allowBackup is missing in Androidmanifest file.

Recommendation

Open the AndroidManifest.xml file in the app's source code.

Set android:allowBackup="true".

Impact

The impact of this vulnerability can be severe and can lead to the following consequences:

- **Data Exposure:** Sensitive user data, including personal information, authentication tokens, and app-specific data, may be accessible to unauthorized parties in the event of a data breach.
- **Privacy Violation:** This vulnerability violates user privacy and may lead to a breach of trust, as users expect their data to be handled securely.

Status

Resolved

7. Information Disclosure by Internal Files

Description

Information Disclosure by Internal Files is a security vulnerability that occurs when an application inadvertently exposes sensitive internal files or directories to unauthorized users or attackers. These internal files may contain critical information such as configuration files, source code, credentials, database backups, or other sensitive data that should not be accessible to external parties.

POC

http://161.35.145.102:8084/web.config

http://161.35.145.102:8084/storage/wallet/*

Recommendation

It is recommended to Implement strong production and development processes to prevent unapproved files from reaching a production environment.

Impact

- **Data Exposure:** The exposure of sensitive information stored in internal files can lead to unauthorized access to personally identifiable information (PII), financial records, intellectual property, or other proprietary data. This can result in identity theft, financial fraud, or misuse of confidential information.

Status

Resolved

Closing Summary

In this report, we have considered the security of the Oron Mobile wallet app. We performed our audit according to the procedure described above.

Some issues of High, medium, low, and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Dapp audit is not a security warranty, investment advice, or an endorsement of the Oron Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multi-step process. One audit cannot be considered enough. We recommend that the Oron Wallet Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com