OtterSec

Auro Mobile Wallet

Security Assessment

October 21st, 2024 — Prepared by OtterSec

Caue Obici caue@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Auro engaged OtterSec to assess the **`auro-wallet-mobile-app`**. This assessment was conducted between October 9th and October 16th, 2024. For more information on our auditing methodology, refer to chapter 07.

Key Findings

We produced 5 findings throughout this audit engagement.

In particular, we discovered a method to exfiltrate private keys and mnemonics by exploiting and hijacking the local web server created by the app, with the only prerequisite being the installation of a malicious app on the mobile device (OS-AMW-ADV-00). Additionally, we reported a phishing technique that leverages an origin spoofing vulnerability, enabling malicious zkApps to impersonate legitimate ones and request signing actions on their behalf (OS-AMW-ADV-01).

We also made recommendations regarding the implementation of SSL Pinning to prevent network attacks (OS-AMW-SUG-01), and suggested increasing the computational rounds of the **`cryptoPwhash`** function to strengthen user defenses against brute-force password attacks (OS-AMW-SUG-00).

02 — Scope

The source code was delivered to us in a git repository at https://github.com/aurowallet/auro-wallet-mobile-app. This audit was performed against commit eb41ec2.

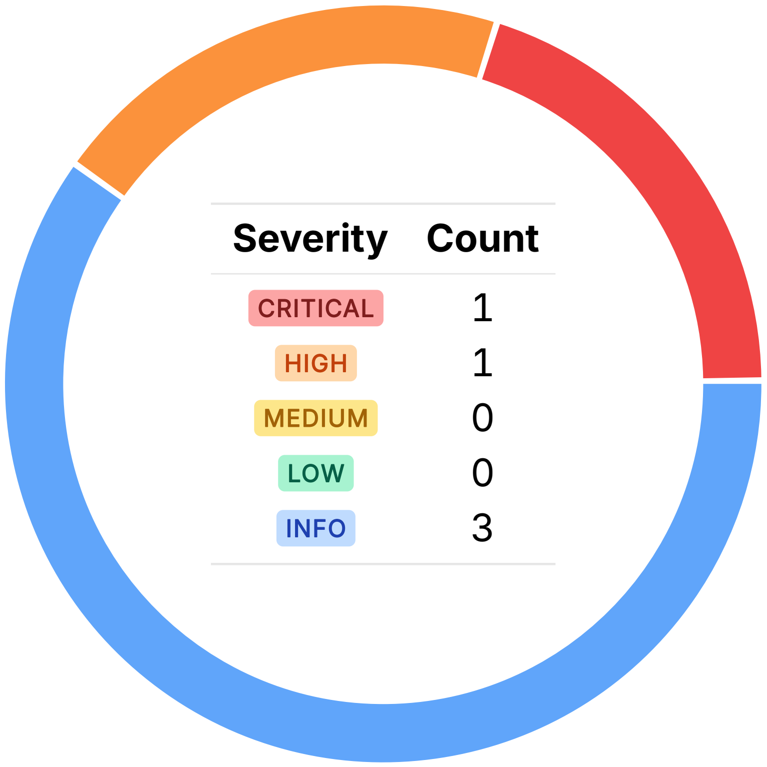
A brief description of the programs is as follows:

Name	Description
auro-wallet-mobile-app	An open-source mobile app wallet designed for the Mina Protocol. It allows you to easily send, receive, and stake MINA and other tokens.

03 — Findings

Overall, we reported 5 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	1
HIGH	1
MEDIUM	0
LOW	0
INFO	3

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in chapter 06.

ID	Severity	Status	Description
OS-AMW-ADV-00	CRITICAL	RESOLVED ✓	It is possible to change the code of the bridge service by hijacking the local web server default port.
OS-AMW-ADV-01	HIGH	RESOLVED ✓	It is possible to spoof the origin of the request since it is passed in the payload.

Bridge Service Hijacking Leaks Private Keys CRITICAL OS-AMW-ADV-00

Description

The bridge service is a javascript bundle executed inside a headless webview. To pull the javascript file, the mobile app creates a local web server by trying to locate an available port from range 8000 to 8200, as demonstrated below:

>_ *LocalWebviewServer.dart* DART

```dart
  int _currentPort = 8080;
  final int _portRangeStart = 8000;
  final int _portRangeEnd = 8200;

Future<String> startLocalServer() async {
    if (await _isPortAvailable(_currentPort)) {
      await _startServerOnPort(_currentPort);
    } else {
      for (int port = _portRangeStart; port <= _portRangeEnd; port++) {
        if (await _isPortAvailable(port)) {
          _currentPort = port;
          await _startServerOnPort(port);
          break;
        }
      }
    }
    return serverUrl;
  }
```

The issue with this algorithm is that if no port is available, the function will still succeed by returning the `serverUrl` with the default port (8080). If a malicious app is present on the device, it could create TCP listeners on all those ports and host an attacker-controlled web server on port 8080.

This setup allows the attacker to serve a malicious JavaScript bundle to the bridge service's webview. Since the webview is used to derive private keys from mnemonics and generate signatures, the malicious bundle could expose this sensitive information, leading to its leakage.

Remediation

Remove the usage of `LocalWebServer` and use static asset loading instead. Here is the official documentation of WebViewAssetLoader

Patch

Fixed in e27f14c by removing the usage of the local webserver and loading the asset statically from the assets directory.

>_ *lib/service/webview/bridgeWebView.dart* diff

```diff
-           controller.loadUrl(
-               urlRequest: URLRequest(
-                   url: WebUri(localServerUrl + "assets/webview/bridge.html")));
+           // https://github.com/pichillilorenzo/flutter_inappwebview/issues/586
+           await controller.loadFile(
+               assetFilePath: "assets/webview/bridge.html");
```

Origin Spoofing HIGH OS-AMW-ADV-01

Description

Messages can be sent on behalf of any origin because the origin is extracted from the message payload, which an attacker can fully control.

>_ *injectedBrowser.dart* DART

```dart
[...]
    Map? params = payload['params'];
    Map? siteInfo = payload['site'];

    Object message = params?["message"];

    await UI.showSignatureAction(
      method: method,
      context: context,
      content: message,
      url: siteInfo?['origin'],
[...]
```

By spoofing the origin, any zkApp can request actions without establishing a valid connection. This technique can be used to phish users by impersonating a legitimate zkApp that the victim has previously connected with and trusted.

Remediation

Consider retrieving the origin from `sourceOrigin` parameter sent to `onPostMessage` handler. Additionally, ensure that the `isMainFrame` parameter is set to true to prevent iframes from initiating unauthorized actions within the mobile app.

Patch

Fixed in 2370f3c by retrieving the source from the `originSource` parameter sent by the `InAppWebView` itself.

>_ *lib/page/browser/injectedBrowser.dart* diff

```diff
- Future<dynamic> _msgHandler(Map msg) async {
+ Future<dynamic> _msgHandler(Map msg, String origin) async {
    final String method = msg['action'];
    Map payload = msg['payload'];
    Map? siteInfo = payload['site'];
+     if (siteInfo != null) {
+       siteInfo['origin'] = origin;
+       payload['site'] = siteInfo;
+     } else {
+       payload['site'] = {
+         "origin": origin
+       };
+     }
[...]

- String? origin = payload?["site"]?['origin'];
+ String origin = sourceOrigin.toString();

+ if (origin.isNotEmpty) {
+     if (id != null) {
+         _msgHandler(msg, origin);
```

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-AMW-SUG-00	Increase computation rounds to make harder brute-force attacks against users' passwords.
OS-AMW-SUG-01	Implement SSL pinning to ensure secure communication by validating the server's certificate against a pre-defined trusted certificate.
OS-AMW-SUG-02	Validate the URL schema in appLink openUrl action.

Increase Opslimit OS-AMW-SUG-00

Description

The Libsodium documentation states that 3 is the minimum value for the `opslimit` parameter; however, it can be higher if it does not significantly impact performance.

>_ *encryption.dart* DART

```dart
static Uint8List password2Hash(String pwd, Uint8List salt) {
    if (!_inited) {
      Sodium.init();
      _inited = true;
    }
    Uint8List key = Sodium.cryptoPwhash(32,
        Uint8List.fromList(pwd.codeUnits),
        salt,
        3,
        Sodium.cryptoPwhashMemlimitInteractive,
        Sodium.cryptoPwhashAlgArgon2id13);
    return key;
  }
```

Remediation

It is recommended to test how long the computation takes and increase the parameter without compromising the user experience with excessive delays. Increasing the value makes the hash computation more time-consuming, thereby reducing the effectiveness of brute-force attacks against private key encryption, potentially rendering them impractical.

Implement SSL Pinning OS-AMW-SUG-01

Description

The application does not implement SSL pinning, leaving it vulnerable to certain types of network routing manipulation, such as BGP hijacking.

With SSL pinning, the client (e.g., a mobile app) is configured to trust only a specific certificate or public key, rather than relying solely on the general list of trusted Certificate Authorities (CAs). This adds an extra layer of security, ensuring the app will not communicate with any server that fails to present the pinned certificate.

Remediation

Implement SSL pinning to the critical servers such as the default MINA nodes and base information server.

Validate URL Scheme OS-AMW-SUG-02

Description

The app has a deeplink handler that allows other apps to open URLs inside the app WebView. The `extractParameters` function extracts the URL to be opened from the deeplink, however, it doesn't properly check if the URL is a valid HTTP URL as demonstrated below:

>_ *lib/utils/index.dart* FLUTTER

```
```

The `isValidHttpUrl` checks only if a domain is valid, however, it should check if the URL has the http schema.

Remediation

The `isValidHttpUrl` should check if the URL starts with `http://` or https:// schemas.

06 — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
HIGH	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
MEDIUM	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
LOW	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
INFO	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

07 — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.