Auro Wallet

Security Assessment

July 14th, 2024 — Prepared by OtterSec

Bruno Halltari bruno@osec.io

Table of Contents

01 — Executive Summary

Overview

Auro engaged OtterSec to assess the `auro-wallet-browser-extension` program. This assessment was conducted between July 3rd and July 9th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 8 findings throughout this audit engagement.

In particular, we identified a high-risk vulnerability regarding the possibility of performing an origin spoofing attack as a result of sending the origin via post message without performing any validation (OS-ARW-ADV-00). Additionally, the current implementation of PBKDF2 with 10,000 iterations for HMAC-SHA256 is significantly below the recommended 600,000 iterations by OWASP, compromising security against brute-force attacks (OS-ARW-ADV-01). Furthermore, the unrestricted batching in the GraphQL endpoint allows attackers to overload the server with large batched requests (OS-ARW-ADV-02).

We also made recommendations to ensure the security of file downloads by validating the `fileName` parameter in `SaveAs` to prevent path traversal vulnerabilities (OS-ARW-SUG-01) and advised enabling GraphQL introspection on the wallet's production endpoint to allow potential attackers to explore and extract detailed information about the GraphQL schema (OS-ARW-SUG-00). Lastly, we made some suggestions regarding open redirects (OS-ARW-SUG-02) and brute force attacks(OS-ARW-SUG-03).

02 — Scope

The source code was delivered to us in a git repository at https://github.com/aurowallet/auro-wallet-browser-extension. This audit was performed against commit 6999924.

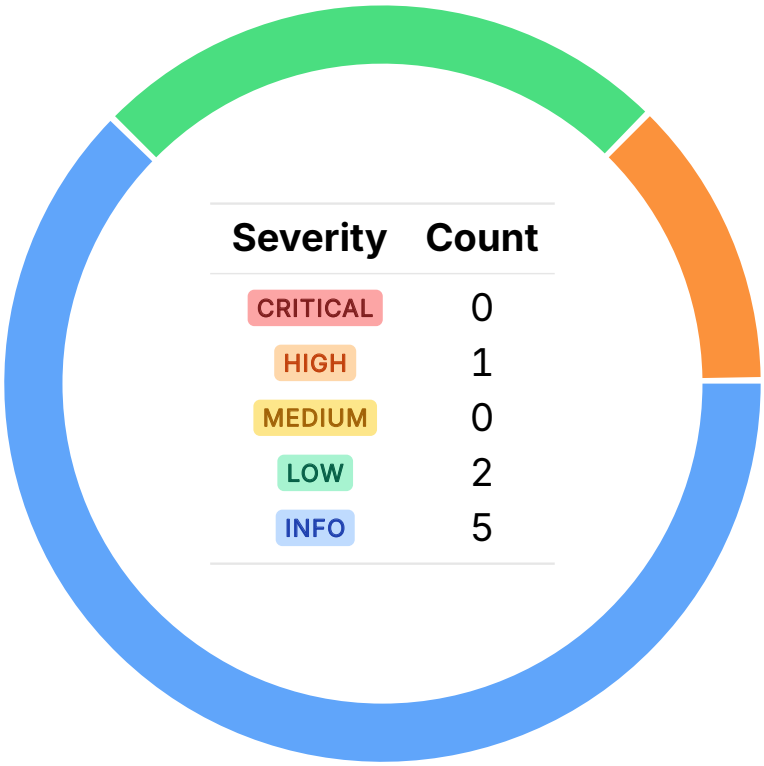
A brief description of the programs is as follows:

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

03 — Findings

Overall, we reported 8 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	0
HIGH	1
MEDIUM	0
LOW	2
INFO	5

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 Appendix A.

ID	Severity	Status	Description
OS-ARW-ADV-00	HIGH	RESOLVED	Possibility of performing an origin spoofing attack as a result of sending the origin via post message without performing any validation.
OS-ARW-ADV-01	LOW	RESOLVED	The current implementation of PBKDF2 with 10,000 iterations for HMAC-SHA256 is significantly below the recommended 600,000 iterations by OWASP, compromising security against brute-force attacks.
OS-ARW-ADV-02	LOW	RESOLVED	The unrestricted batching in the GraphQL endpoint allows attackers to overload the server with large batched requests

Origin Spoofing HIGH OS-ARW-ADV-00

Description

The current implementation of post message inside Auro Wallet is susceptible to origin spoofing because it sends the origin via post message without performing a valid check.

JAVASCRIPT

```
let siteUrl = site.origin
        let openId = id
        let openParams = new URLSearchParams({ siteUrl, siteIcon: site.webIcon, openId
            ↪ }).toString()
        this.popupId = await this.dappOpenPopWindow('./popup.html#/request_sign?' + openParams,
            ↪ windowId.request_sign, "dapp")
```

This vulnerability may potentially enable an attacker to bypass the permission request if an allowed origin is passed and may also allow to perform phishing attacks since every website may perform an action by showing the origin "google.com" instead of the real origin.

Proof of Concept

We have created a proof-of-concept that is able to spoof and take the `origin` of an action, when visited with the wallet installed, notice how the connection request will come from "google.com" instead of real origin.

>_ poc.js JAVASCRIPT

```
window.postMessage({
   message: {
       action: "messageFromWeb",
       data: {
           action: "mina_requestAccounts",
           payload: {
             id: '1',
             site: {
               origin: 'https://google.com',
                 webIcon: 'https://static-00.iconduck.com/assets.00/
                 google-icon-1024x1024-hv1t7wtt.png'
             }
           },

       }
   },
   source: "https://palant.info/articles/",
   isAuro: true
```

```
}, '*');
```

Remediation

It is recommended to avoid forwarding the origin via post message and instead verify the origin via `event.origin` property.

Patch

Fixed in 413e68f.

Insufficient Key Rotations LOW OS-ARW-ADV-01

Description

Password-Based Key Derivation Function 2 (PBKDF2) enhances password security by increasing the computational costs to derive the cryptographic key from the password. The security of PBKDF2 primarily depends on the number of iterations (work factor) along with the hashing algorithm utilized and the salt value. The application utilizes PBKDF2 with 10,000 iterations and the SHA-256 hashing algorithm as shown in the code snippet provided below.

JAVASCRIPT

```
return global.crypto.subtle.importKey(
    'raw',
    passBuffer,
    { name: 'PBKDF2' },
    false,
    ['deriveBits', 'deriveKey']
  ).then(function (key) {

    return global.crypto.subtle.deriveKey(
      { name: 'PBKDF2',
        salt: saltBuffer,
        iterations: 10000,
        hash: 'SHA-256',
      },
      key,
      { name: 'AES-GCM', length: 256 },
      false,
      ['encrypt', 'decrypt']
    )
  })
```

However, it utilizes only 10,000 iterations for PBKDF2-HMAC-SHA256, which is significantly lower than the current OWASP recommendation of 600,000 iterations. This lower iteration count reduces the overall time of the key derivation process but also reduces the security by rendering it vulnerable for attackers to perform brute-force attacks more easily.

Remediation

The number of iterations may be increased to the recommended number of 600,000 for SHA-256. However, this may have a significant impact on performance, thus, it may be increased to 100,000 iterations to balance security and performance based on the application's requirements.

Patch

Fixed in 175d2bd.

Denial Of Service Due To Batching Queries LOW OS-ARW-ADV-02

Description

The GraphQL endpoint for the wallet, accessible here, currently supports batching without any limit. Batching in GraphQL allows multiple queries to be sent in a single request. While this may improve efficiency, it also poses a significant security risk if not properly controlled. Allowing unlimited batching may be exploited to perform Denial of Service (DoS) attacks. An attacker may craft a request that contains an excessively large number of queries, which will overload the server

Remediation

If batching is necessary, implement limits on the number of queries that may be batched in a single request or else disable it entirely.

Patch

Fixed by limiting the number of aliases for all fields.

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-ARW-SUG-00	Enabling GraphQL introspection on the wallet's production endpoint allows potential attackers to explore and extract detailed information about the GraphQL schema.
OS-ARW-SUG-01	Recommendation to ensure the security of file downloads by validating the `fileName` parameter in `SaveAs` to prevent path traversal vulnerabilities.
OS-ARW-SUG-02	Recommendation to ensure the security of the extension against `open redirects`.
OS-ARW-SUG-03	Recommendation to implement a rate limit that restricts the number of attempts allowed, enhancing the security of the extension against brute force attacks.
OS-ARW-SUG-04	Recommendation to use `encodeURIComponent` to prevent any injection via Network ID.

Disable Graphql Introspection OS-ARW-SUG-00

Description

The GraphQL endpoint for the wallet (here) currently has the introspection feature enabled. Introspection allows clients to query the GraphQL schema to discover its structure, including types, queries, mutations, and subscriptions available. While introspection is useful during development for understanding and testing the API, keeping it enabled in a live production environment may pose a security risk. Introspection may be utilized to map out the entire schema, gaining insights into the underlying data structures and operations.

Remediation

Disable the introspection feature in the production environment.

Patch

The introspection feature has been disabled in the production environment.

Filename Validation OS-ARW-SUG-01

Description

Users may manipulate paths in web environments to traverse directories (`../` or `\..\`) and potentially access files outside of the intended directory. In `JsonToCSV::SaveAs` If `fileName` is not properly sanitized, an attacker may craft a malicious `fileName` that includes these sequences to navigate to sensitive files on the server or elsewhere in the filesystem accessible to the server.

>_ JsonToCSV.js JAVASCRIPT

```
SaveAs: function(fileName, csvData) {
    let alink = document.createElement("a")
    let downloadUrl = this.getDownloadUrl(csvData)
    alink.id =  "linkDownloadLink"
    alink.href = downloadUrl
    document.body.appendChild(alink)
    let linkDom = document.getElementById('linkDownloadLink')
    linkDom.setAttribute('download', fileName)
    linkDom.click()
    document.body.removeChild(linkDom)
  }
```

Remediation

implement a simple validation check on the `fileName` before setting it as the download attribute by verifying it does not contain `/../` and `\..\` .

Patch

Fixed in d343a4d.

Implement Chrome Runtime Postmessage OS-ARW-SUG-02

Description

The extension is currently passing certain parameters via URL inside `approve-page` and `request-sign` .

>_ *DappService.js* JAVASCRIPT

```
let openParams = new URLSearchParams({ siteUrl, siteIcon: site.webIcon,id }).toString()
        this.popupId = await this.dappOpenPopWindow('./popup.html#/approve_page?' + openParams,
          windowId.approve_page, "dapp")
```

This approach may allow an attacker to redirect a page to a `chrome-extension://` URL, enabling them to create a malicious payload to phish users into approving a fraudulent action.

Remediation

It is recommended to send every parameter within the URL using `chrome.runtime.postMessage`.

Patch

Fixed in 19fe7f6.

Implement Rate Limit

OS-ARW-SUG-03

Description

The extension lacks a rate limit on sensitive operations such as updating passwords and viewing recovery phrases when a wrong password is inserted. This makes it easier for attackers to perform brute-force attacks by repeatedly guessing passwords until they find the correct one.

Remediation

It is recommended to implement a rate limit that restricts the number of attempts allowed after some failed attempts.

Encode NetworkId

OS-ARW-SUG-04

Description

The method `fetchSupportTokenInfo` has the following code :

```javascript
export async function fetchSupportTokenInfo() {
  let netConfig = await getCurrentNodeConfig();
  const readableNetworkId = getReadableNetworkId(netConfig.networkID);
  const requestUrl =
    BASE_INFO_URL + "/tokenInfo?networkId=" + readableNetworkId;
  const data = await commonFetch(requestUrl).catch(() => []);
  if (data.length > 0) {
    saveLocal(
      SUPPORT_TOKEN_LIST + "_" + readableNetworkId,
      JSON.stringify(data)
    );
  }
```

Since the network ID can be controlled by adding a custom chain network, it is recommended to use `encodeURIComponent` to prevent any injection on the query parameters.

Remediation

it is recommended to use `encodeURIComponent` to prevent any injection on the query parameters.

Patch

Fixed in f32b14d.

A — 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.

B — 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 the 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.