OPEN SECURITY

Community Labs: ArConnect Source Code Audit and Penetration Test

Security Review

Joshua Christman | Chief Operations Officer

Last Revision: 31 July 2023

Powered by CANTINA

1 TABLE OF CONTENTS

2 EXECUTIVE SUMMARY

Many projects in the Web3 ecosystem focus intensely on the security of their blockchain ecosystems. It has become increasingly apparent, however, that further attention needs to be paid to traditional web2 security, as it is foundational to the Web3 ecosystem. Community Labs' ArConnect Wallet provides a Browser Extension that interacts directly with the Arweave protocols and gateways to make a more seamless user experience.

The testing period started on 03 July 2023 and ended on 17 July 2023 and consisted of auditing the Source Code for vulnerabilities, along with testing those potential vulnerabilities through dynamic testing.

Thorough examination of the source code showed strong coding practices, with only one High-severity finding in the extension. While there are some hardening measures that have been identified during testing, the strong technology stack and architecture mitigate several vulnerabilities, allowing for those issues to be mitigated on a longer timeline. Community Labs exhibited strong motivation to address identified issues, patching all 5 findings (including Informational findings) within 2 weeks of initial report delivery.

2.1 RECOMMENDATIONS

The architecture and coding practices leave little room for improvement with the ArConnect browser extension. Implementing a patching cadence where Open Source packages are routinely audited and updated for security vulnerabilities will close the single High-severity finding, which introduced by a third party. Open Security also recommends security testing of new features as they are developed in order to proactively identify any potential regressions in security introduced by these features.

2.1.1 RECOMMENDATIONS FOR SUSTAINMENT

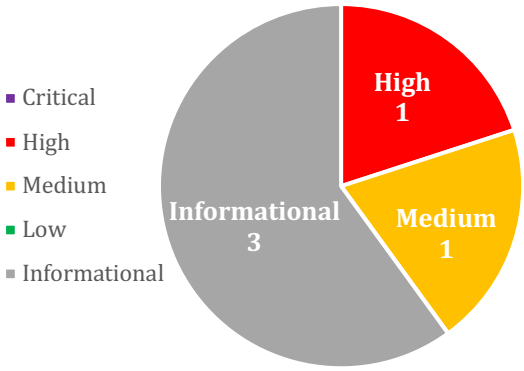
- Continue to architect the application in a secure way, with a strong permissions model.
- Proactive engagement with security vendors allows Community Labs to find vulnerabilities earlier in the development lifecycle. Continuation of this practice can prevent future breaches.

2.1.2 RECOMMENDATIONS FOR REMEDIATION

- Implement a patching cadence for Open Source dependencies.
- Add data validation to all untrusted data sources.
- Consider password complexity and age with regards to the security of the stored data. Implement password policies that address the sensitivity of the data balanced against the security of the password.

2.2 SUMMARY OF FINDINGS

During this Vulnerability Assessment one new **High-** and one new **Medium-severity** vulnerability was identified in the Browser Extension, while three **Informational** findings were reported. All 5 identified vulnerabilities have been addressed and re-tested – they are considered to be fully remediated.



Findings grouped by risk severity:

Critical	*0 → 0*
High	*1 → 0*
Moderate	*1 → 0*
Low	*0 → 0*
Informational	*3 → 0*

2.2.1 KEY FINDINGS

During the Web Application Assessment, 0 critical-risk and 1 high-risk findings were discovered. Findings are listed once even if they pertain to multiple systems across the network and vulnerabilities of common criteria are grouped together.

- The *Vulnerabilities in Open Source Dependencies* finding is based on the presence of vulnerabilities reported by the `npm audit` command, which utilizes public vulnerability databases to identify issues introduced by third-party dependencies. Performing automated patching (which does not update breaking changes) fixes many of the identified vulnerabilities, while leaving a High-severity finding that must be manually addressed. **This finding was remediated and re-tested on 31 July 2023.**

Finding #	Description	Severity	Remediation Status
Finding 1	Vulnerabilities in Open Source Dependencies	~~High~~	**Remediated**

2.2.2 CURRENT RISK ASSESSMENT

Open Security assesses your overall security risk to be: **Low**

3 RISK METHODOLOGY

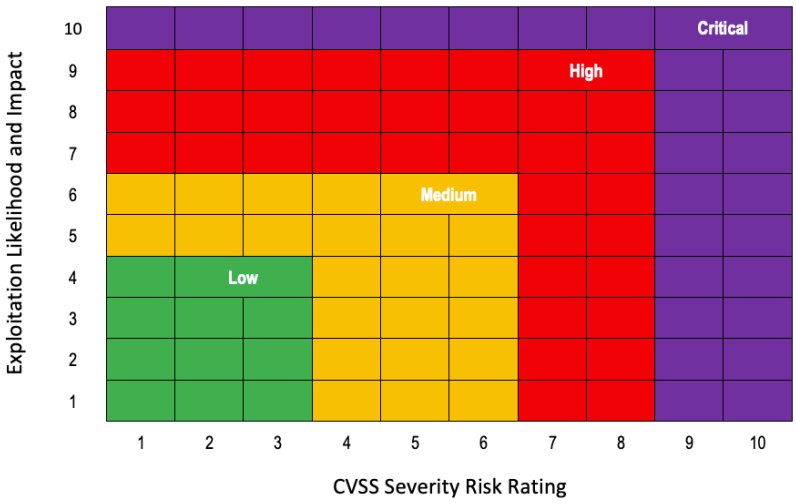
Information security is not about eliminating risk. It is founded upon the science and discipline of risk management. This is an important distinction because computer systems are inherently designed to share information while security strives to guard it. Therefore, it is management's role to weigh the benefits of information sharing with the potential security risks of doing so, all while enabling the organization to achieve its objectives.

3.1 INFORMATION SECURITY RISK RATING SCALE

To effectively evaluate the security posture of a client's network, Open Security uses the Information Security Risk Rating Scale shown below. This scale is based on the open-industry Common Vulnerability Scoring System (CVSS) against the Common Vulnerabilities and Exposures (CVE) Dictionary maintained by the National Cybersecurity Federally Funded Research and Development Center (FFRDC) with funding from the National Cyber Security Division of the US Department of Homeland Security. This base CVSS score, the likelihood of exploitation, and the impact of exploitation are all considered to determine the overall risk presented by the vulnerability.

3.1.1 RISK RATING KEY

When evaluating remediation timelines for your environment, ***Critical*** network and system vulnerabilities should be addressed as quickly as feasible. The bulk of effort will likely involve those rated as ***High*** and ***Medium.*** Open Security recommends that these risks be remediated as soon as possible after report delivery. While it is of vital importance to identify solutions to all risks affecting the network, those rated ***Low*** can be approached methodically, in line with general information security best practices without accepting significant risk of severe financial or data loss. ***Informational*** vulnerabilities are meant to point out accepted best practices but are not included on the chart below because they are either unexploitable in the environment or an exploitation would have no impact on the environment.



Critical risks: very high likelihood of exploitation and possibility of ***catastrophic*** financial losses.

High risks: high likelihood of exploitation with the possibility of ***significant*** financial losses.

Medium risks: average likelihood of exploitation with the possibility of ***material*** financial losses.

Low risks: below average likelihood of exploitation with the possibility of ***limited*** financial losses.

Informational risks: below average likelihood of exploitation with ***little to no impact*** as a result.

4 SECURITY ROADMAP

To strengthen overall information security, Open Security has provided a prospective security roadmap below. The timeline is broken into short-, mid-, and long-term remediation efforts to help security teams prioritize their work. Recommendations are based only on the information gained from this engagement and may not work for all security programs – though they may be a good starting point for planning discussions.

The roadmap takes the overall severity of each finding into account, alongside an estimate of the resources required to address each finding, in order to recommend short-, mid-, and long-term remediation efforts. In other words, a low-risk finding may be recommended for short-term remediation if minimal effort is required to generate a fix, while high or medium risk findings may be prioritized lower if substantial resources must be committed to address a vulnerability. Critical findings should almost always be addressed in the short term in some way, even if only a temporary stopgap is used to reduce risk while a more permanent solution is employed in the long term.

4.1 FINDINGS

Finding #	Description	Severity	Remediation Status
Finding 1	Vulnerabilities in Open Source Dependencies	~~High~~	**Remediated**
Finding 2	Weak Password Policy	~~Medium~~	**Remediated**
Finding 3	Decrypted Wallet Details Held In-Memory	~~Informational~~	**Remediated**
Finding 4	Potential DOM-Based Cross-Site Scripting Vulnerability	~~Informational~~	**Remediated**
Finding 5	No Input Validation	~~Informational~~	**Remediated**

4.1.1 SHORT-TERM REMEDIATION

Short-term remediations should be prioritized for implementation in the next 21 days. These findings typically rank higher in severity and will address the most dangerous vulnerabilities to an organization. They also may be included if there appear to be risks related to maintaining mandatory compliance or other regulatory requirements, as failing those audits may impact continued business operations.

No findings were discovered that were of Critical Severity, which is an outstanding result. This indicates an excellent short-term security posture for Community Labs.

4.1.2 MID-TERM REMEDIATION

Mid-term remediations should be prioritized for implementation in 21 - 45 days. These findings are usually categorized by a cost-benefit analysis of security impact and effort to implement. A high risk finding with significant resource and planning investment may be included here – though every effort to speed up remediation should be made if technical or procedural circumstances allow.

Finding #	Description	Severity	Remediation Status
Finding 1	Vulnerabilities in Open Source Dependencies	~~High~~	**Remediated**
Finding 2	Weak Password Policy	~~Medium~~	**Remediated**

Finding 1: Vulnerabilities in Open Source Dependencies (Remediated)

Vulnerability Rating: ~~High~~

Discovery Method: Manual testing

Remediation Status:

This finding was re-tested on 31 July 2023, with 0 vulnerabilities reported by the `yarn audit` command. This finding is considered to be **fully remediated**.

Description:

When developing software, the security of its dependencies (i.e. the Software Supply Chain) is a significant attack surface. The Supply Chain can affect a product's security at any time in the development process by attacking the developers' tools themselves with malware or by simply introducing vulnerabilities into the software. NPM dependencies in particular, can be challenging to keep up-to-date due to the complex dependency relationships that develop in the NodeJS ecosystem.

Affected Assets:

- `package.json`

Analysis:

The ArConnect Wallet Browser Extension has many out-of-date dependencies, with 6 Critical-, 22 High-, 31 Moderate-, and 7 Low-severity vulnerabilities being reported by the `yarn audit` command. The impact of these vulnerabilities is highly variable, thought the Critical vulnerabilities include dangerous issues such as JavaScript VM sandbox escapes with published Proofs-of-Concept on Github (see Github Advisory in References).

Reproduction Steps:

Run the command `yarn audit`, noting the response:

```
66 vulnerabilities found, 7 Low | 31 Moderate | 22 High | 6 Critical
```

Recommendation:

Follow the steps below to automatically apply non-breaking changes/updates from the NPM registry. For any vulnerabilities that are not fixed automatically, replacement packages or manual updates may be required in order to deal with breaking changes.

1. Run the command `yarn audit`, noting the response (`66 vulnerabilities found, 7 Low | 31 Moderate | 22 High | 6 Critical`)
2. As the `yarn` command does not have a `fix` subcommand like the `npm` CLI, utilize `npm` to apply fixes automatically. Run the following commands:

- `npm i --package-lock-only --legacy-peer-deps`
- `npm audit fix --legacy-peer-deps`

3. After executing these commands, the vulnerabilities are reduced to 7 vulnerabilities (2 low, 4 moderate, 1 high), which will need to be dealt with manually due to breaking changes and/or vulnerabilities without a fix available.

References:

- https://www.dni.gov/files/NCSC/documents/supplychain/Software_Supply_Chain_Attacks.pdf
- https://github.com/advisories/GHSA-whpj-8f3w-67p5

Finding 2: Weak Password Policy (Remediated)

Vulnerability Rating: ~~Medium~~

Discovery Method: Manual testing

Remediation Status:

This finding was re-tested on 31 July 2023. All recommendations were implemented, with a password expiration of 6 months and a minimum strength of “3” for the `check-password-strength` package. This finding is considered to be **fully remediated**.

Description:

When encrypting/decrypting the ArConnect wallet, an encryption key is derived from cryptographically secure, randomly generated values and a user-provided password. This is then utilized to encrypt the wallet. When decrypting the same wallet, only the user-provided password is required. This effectively makes the strength of the user-provided password be the strength of the encryption key.

The password requirement is set by a function in `generator.ts` (Figure 1), where it checks the password strength utilizing an NPM package called `check-password-strength` (see References). The check is set to only require a "Weak" password, which requires a minimum length of 6 characters and 2 different sets of characters (lowercase and special characters, for example).

```
/**
 * Check password strength
 *
 * @param password Password to check
 */
export function checkPasswordValid(password: string) {
  const strength = passwordStrength(password);

  return strength.id >= 1;
}
```

Figure 1 – checkPasswordValid Function Source Code

In addition to this, no password expiration policy is enforced. While not strictly required, a password expiration on a yearly basis may strengthen a user's security stance and prevent an attacker from performing passwords stuffing attacks when trying to decrypt a user's keyfile.

Affected Assets:

- `ArConnect/src/wallets/generator.ts`

Analysis:

With a weak password requirement, it may be possible to brute force decrypt a keyfile that is captured via other means, such as malware or local access to a computer.

Recommendation:

- Enforce a strong password requirement by updating the `checkPasswordValid` function to require a `strength.id === 3`
- Consider implementing a password expiration mechanism wherein a user is required to rotate their password on a periodic basis

References:

- `https://github.com/deanilvincent/check-password-strength#object-result`

4.1.3 LONG-TERM REMEDIATION

Long-term remediations are reserved for low impact vulnerabilities that should be prioritized for remediation after all other vulnerabilities are addressed – usually around 45 days from the delivery of this report. These finding are either very hard to exploit or will have minimal impact to users and business operations. Many of the findings in this section will become the responsibility of an ongoing vulnerability management program and will be addressed as software updates are released or organizations grow.

Finding #	Description	Severity	Remediation Status
Finding 3	Decrypted Wallet Details Held In-Memory	~~Informational~~	**Remediated**
Finding 4	Potential DOM-Based Cross-Site Scripting Vulnerability	~~Informational~~	**Remediated**
Finding 5	No Input Validation	~~Informational~~	**Remediated**

Finding 3: Decrypted Wallet Details Held In-Memory (Remediated)

Vulnerability Rating: ~~Informational~~

Discovery Method: Manual testing

Remediation Status:

This finding was re-tested on 31 July 2023. Community Labs developers implemented a `freeDecryptedWallet` function that overwrites the sensitive values in memory when they are no longer needed in order to prevent third-party malware from reading the values from memory. The function is appropriately utilized and implemented, and this finding is considered to be **fully remediated**.

Description:

JavaScript engines have unpredictable memory-management practices, with unpredictable garbage collection practices. Accordingly, any time a sensitive value is moved into memory, it will stay there until the JavaScript engines determine that it is no longer in use and frees the memory for re-use (see References). Even at this point, the memory is not overwritten and will retain the value until it is overwritten by other data. Accordingly, sensitive data held in-memory by JavaScript is at-risk if process memory is dumped.

Affected Assets:

`decryptWallet` calls:

- `ArConnect/src/components/dashboard/subsettings/WalletSettings.tsx`
- `ArConnect/src/routes/popup/send/auth.tsx`
- `ArConnect/src/wallets/auth.ts`
- `ArConnect/src/wallets/index.ts`

`getActiveKeyfile` calls:

- `ArConnect/src/api/modules/decrypt/decrypt.background.ts`
- `ArConnect/src/api/modules/dispatch/dispatch.background.ts`
- `ArConnect/src/api/modules/encrypt/encrypt.background.ts`
- `ArConnect/src/api/modules/public_key/public_key.background.ts`
- `ArConnect/src/api/modules/sign/fee.ts`
- `ArConnect/src/api/modules/sign/sign.background.ts`
- `ArConnect/src/api/modules/signature/signature.background.ts`
- `ArConnect/src/components/arlocal/Transaction.tsx`

Analysis:

For the ArConnect browser extension, it was determined during scoping that a user is responsible for the security of their own machine. That is, if malware dumps memory for the browser, ArConnect cannot be expected to protect the user's private key and wallet information, which is held in-memory during various function calls (as enumerated in the Affected Assets). Due to this predetermined Risk Management decision, this finding is marked as Informational, as it is not considered to be a risk by Community Labs. Open Security,

however, felt it was prudent to establish that there is a potential attack vector for ArConnect users that can be mitigated with some effort.

Recommendation:

The main root of this finding is that decrypted, sensitive wallet information is held in memory for indeterminate amounts of time, due to the unpredictable nature of the JavaScript engine garbage collectors. Starting with the `decryptWallet` function in `source/ArConnect/src/wallets/encryption.ts`, it is possible to trace the references path through various functions and determine when the need for that data begins and ends. For example, line 53 of `source/ArConnect/src/wallets/auth.ts` utilizes the `decryptWallet` function in the `checkPassword` function as a way of verifying the password, but does not actually utilize the wallet information (Figure 2). This causes the decrypted JWK to be stored in memory for an indeterminate amount of time.

```
try {
  await decryptWallet(localWallets[0].keyfile, password);

  return true;
} catch {
  return false;
}
```

Figure 2 – decryptWallet Function Call in auth.ts

A solution to this would be to make a new function that overwrites the in-memory wallet information that can be called any time the wallet information is no longer needed. While there is no way to manually free the memory reference and cause it to be freed, the in-memory data can be overwritten when a reference to the data is still in-use. A potential implementation is described below; note that the data being overwritten needs to be overwritten with data of the same length in order to be guaranteed that the memory is overwritten fully.

```
// While the "free" language here is a misnomer due to the lack of memory being freed,
// it is overwriting the sensitive data and gives a connotation of freeing due to the data
// no longer being needed.
function freeDecryptedWallet(jwk: JWKInterface) {
  jwk.kty = "0000000000000000000000";
  jwk.e = "0000000000000000000000";
  jwk.n = "0000000000000000000000";
  jwk.d = "0000000000000000000000";
  jwk.p = "0000000000000000000000";
  jwk.q = "0000000000000000000000";
  jwk.dp = "0000000000000000000000";
  jwk.dq = "0000000000000000000000";
  jwk.qi = "0000000000000000000000";
}
```

With such an implementation, once the decrypted wallet information is no longer needed, a call to freeDecryptedWallet and passing the information by reference can allow for that memory to be overwritten.

References:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_management

Finding 4: Potential DOM-Based Cross-Site Scripting Vulnerability (Remediated)

Vulnerability Rating: ~~Informational~~

Discovery Method: Manual testing

Remediation Status:

This finding was re-tested on 31 July 2023. The file with the vulnerable code was removed from the project in order to prevent future introduction of the vulnerability. Accordingly, this finding is considered to be **fully remediated**.

Description:

A DOM-based Cross-Site Scripting (XSS) attack occurs when uncontrolled input is inserted into the DOM via JavaScript methods, such as via a `document.createElement` function call. If an attacker can provide HTML to vulnerable code, a `<script>` tag could be provided that results in client-side code execution, allowing exfiltration of secrets.

Affected Assets:

- `ArConnect/src/api/modules/connect/overlay.ts`

Analysis:

The `createOverlay` function creates a div and sets its `innerHTML` property to a string, which contains a variable that is not validated to prevent XSS attacks (Figure 3). This vulnerability, if provided with attacker-controlled data, could lead to client-side code execution, allowing an attacker to execute arbitrary JavaScript within the browser.

```
/**
 * Create an overlay for pending actions
 *
 * @param text Text to display in the overlay
 * @returns overlay html
 */
export default function createOverlay(text: string) {
  const container = document.createElement("div");

  container.classList.add(OVERLAY_CLASS);
  container.innerHTML = `
    <div style="position: fixed; top: 0; bottom: 0; left: 0; right: 0; z-index: 1000000000000; background-color: r
      <div style="position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); color: #fff;">
        <h1 style="text-align: center; margin: 0; font-size: 3em; font-weight: 600; margin-bottom: .35em; line-hei
        <p style="text-align: center; font-size: 1.2em; font-weight: 500;">${text}</p>
      </div>
    </div>
  `;

  return container;
}
```

Figure 3 – createOverlay Function Source Code

This function is not currently called anywhere within the application, making this vulnerability hypothetical in nature at this time.

Recommendation:

Sanitize data being passed to this function to ensure no DOM elements are provided or remove the function from the codebase.

Finding 5: No Input Validation (Remediated)

Vulnerability Rating: ~~Informational~~

Discovery Method: Manual testing

Remediation Status:

This finding was re-tested on 31 July 2023. Community Labs developers implemented input validation utilizing the `typed-assert` package in alignment with best practices. The code is implemented cleanly with DRY principles by including common assertions within a utils file, and this finding is considered to be **fully remediated**.

Description:

Input validation is crucial in preventing an attacker from performing injection-based attacks against any web application. When a certain type of data is expected but an attacker provides a specially crafted, alternatively formatted piece of data, unexpected behaviors can occur. Any time untrusted data is processed by any application, it should first validate that the data matches the expected format in order to protect against these attacks.

Affected Assets:

- ArConnect Wallet

Analysis:

No successful attacks were discovered during this engagement, largely due to the protections provided by React's technology stack. As the entire application exists in the client-side, no databases exist for attacks such as NoSQL injections. Due to the modern technology stack, there is no immediate risk which is why this is labeled as an Informational finding.

Recommendation:

Validate all untrusted data prior to processing or displaying the information, especially on API calls that require no permissions (such as the `AddToken` API).

5 METHODOLOGY

Open Security utilizes a source-code audit methodology adapted from the OWASP Code Review Guide (https://owasp.org/www-project-code-review-guide/). Accordingly, a detailed checklist is provided in *Appendix B – ArConnect – Secure Code Review Checklist.xlsx*.

6 CONTACT INFORMATION

This report represents a "snapshot" of the security environment assessed at a specific point in time. Conditions may have improved, deteriorated, or remained the same since this assessment was completed. Open Security cannot guarantee the discovery of all system vulnerabilities, breaches, or attempted breaches. Should there be any questions regarding the contents of this report, please don't hesitate to contact us.

6.1 SECURITY TEAM SNAPSHOT

Passionate and forward-thinking, our team possesses decades of combined technical experience as top-tier researchers, penetration testers, application security experts, and more. Drawing from experience in the US military and leading technology firms, we pride ourselves on the capabilities we make available to our clients. Open Security understands the importance of information security and appreciates the opportunity to have worked on this engagement.

JOSHUA CHRISTMAN – CHIEF OPERATIONS OFFICER | JOSH@OPENSECURITY.IO

Josh Christman graduated from the Air Force Academy in 2013 with a BS in Computer Engineering and Computer Science with a focus on Cyberwarfare. Following USAFA, he proceeded to get his Master of Science in Computer Engineering from the Air Force Institute of Technology, focusing on Artificial Intelligence and algorithms and publishing/presenting a paper at the International Conference for Machine Learning and Applications. His Air Force career then continued into Offensive Cyber Operations, working for the premier offensive cyber unit in the Air Force, spending time as an operator at the NSA and US Cyber Command in Fort Meade, MD. Since transitioning out of the Air Force in 2020, he has focused on the private sector where he has quickly transitioned from a Red Team Operator to running Red Teams, Application Security Teams, and Vulnerability Management Programs in the fintech industry. He now runs all engineering efforts for Open Security as the Chief Operations Officer.

7 APPENDIX A – SCOPE

The scope for this engagement included a public Github repository branch for the source code review and a beta Google Chrome application. It was predetermined that a user installing malware and dumping sensitive information from memory was not an attack scenario that Community Labs was focused on addressing in this iteration of the application, so focus was primarily given to Web and Browser Extension vulnerability classes.

Original Source Code:
https://github.com/arconnectio/ArConnect/tree/fffd1bdbdffca74021b7c386a8d65845d8bc59ca

Remediated Source Code:
https://github.com/arconnectio/ArConnect/tree/df999c5b965818e9984a9750b58da94964dc2fae

Google Chrome Extension: https://chrome.google.com/webstore/detail/arconnect-beta/ekmpjilfjeghbjgddfgfbakkjmobfhhm

8 APPENDIX B – SECURE CODE CHECKLIST

See attached *Appendix B – ArConnect – Secure Code Checklist.xlsx*