

# OPEN SECURITY RISK ASSESSMENT REPORT

ARCONNECT SOURCE CODE AUDIT

---

## COMMUNITY LABS

---

PREPARED BY

**Vance Walsh**  
Security Engineer

EDITED BY

**Joshua Christman**  
Chief Operations Officer | OSCP, OSCE

LAST REVISION

**May 6, 2024**

CONTACT

**737.270.9486**  
contact@opensecurity.com

## TABLE OF CONTENTS

<b>EXECUTIVE SUMMARY</b>	<b>3</b>
<b>RISK METHODOLOGY</b>	<b>5</b>
<b>ENGAGEMENT OVERVIEW</b>	<b>6</b>
<b>SECURITY ROADMAP</b>	<b>7</b>
Short-Term Remediation	8
Mid-Term Remediation	9
ARCO-2024-01: Vulnerabilities in Open Source Dependencies (Remediated)	10
Long-Term Remediation	12
ARCO-2024-02: Unexploitable DOM-based XSS	13
ARCO-2024-03: getActiveKeyfile and freeDecryptedWallet function usage	16
ARCO-2024-04: React State kept in-memory after use	19
ARCO-2024-05: HTML a tag with REL properties misspelled (Remediated)	21
ARCO-2024-06: Suggested Hardening Changes	23
<b>METHODOLOGY</b>	<b>24</b>
Phase 1 – Overall Secure Code Review	24
Phase 2 – Code review of changes since last review	24
Phase 3 – Attempted Exploitation and Creation of POC's for Identified Vulnerabilities	25
POC 1 – Explore Page PermaWeb News RSS Feed XSS Injection Attempt	27
POC 2 – XSS Attempts Within a Malicious Subscription	29
<b>APPENDIX A: SECURITY TEAM SNAPSHOT</b>	<b>31</b>
<b>APPENDIX B: DETAILED SCOPE</b>	<b>32</b>

## EXECUTIVE SUMMARY

### OVERVIEW

The ArConnect Source Code Audit provided to Community Labs is intended to provide awareness of any potential methods of attack that could be leveraged by an external party against the ArConnect browser Extension. Testing started on 15 Apr 2024 and ended on 26 Apr 2024 and consisted of a secure code review for the ArConnect browser extension. For more details see [Appendix B: Detailed Scope](#).

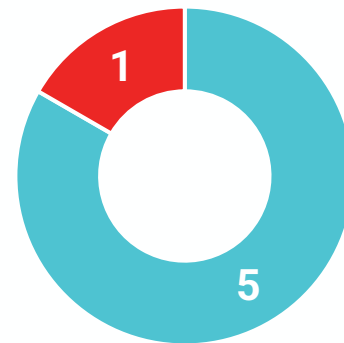
### DESCRIPTION

The secure source code audit was conducted using methodologies developed by the Open Web Application Security Project (OWASP). These methodologies were utilized to ensure standardization and rigor throughout the assessment. The entirety of the engagement was conducted using the Google Chrome browser and installing the development build of the ArConnect browser extension.

Custom malicious proof-of-concepts (aka POCs) were developed to test cases where vulnerabilities were suspected. This included modifying and testing development builds of the ArConnect browser extension with maliciously modified source code, hosting web servers with malicious payloads and debugging the browser extension. **In all test cases, the ArConnect application showed a strong defensive posture and the attacks were unsuccessful.**

### FINDINGS BREAKDOWN

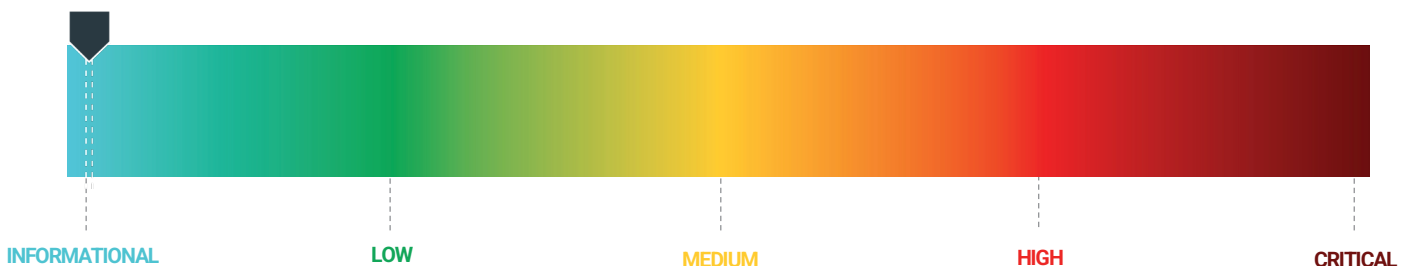
A breakdown of findings by severity is provided in the chart below, which informs the overall risk rating for the ArConnect Application. **Since delivery of this report, all findings have been remediated to be no more than Informational-severity.**



- Informational
- Low
- Moderate
- High
- Critical

### CURRENT RISK RATING

Informational



## KEY FINDINGS

During this Source Code Audit zero **Critical**-risk, one **High**-risk and five **Informational** findings were identified in the source code of the ArConnect browser extension.

- ArConnect's primary source of risk originates from vulnerabilities in open-source dependencies. Community Labs must consistently update and apply upstream patches for open-source dependencies in order to protect from Supply-chain attacks.
- Several code changes have been recommended to harden the ArConnect browser extension, including addressing a currently unexploitable DOM-based XSS

## IDENTIFIED TRENDS

The following trends were identified that are present in many of this report's findings. By addressing these trends, Community Labs can make proactive strides towards preventing security problems from arising.

- Open-source dependencies will consistently have new vulnerabilities found. Keeping these updated on a consistent basis will greatly benefit the security posture of ArConnect
- Memory safety continues to be an important consideration for the ArConnect browser extension development lifecycle. Hardening the browser extension against memory dumps can further solidify the secure posture of ArConnect

## RECOMMENDED ACTION ITEMS

- **Improve Package Management Update Schedule:**  
All high-risk findings stem from out-of-date open-source packages. By utilizing a consistent schedule to update the open-source packages, Community Labs can ensure that the overall risk posed to the ArConnect browser extension project is reduced.
- **Implement Additional Hardening Code Changes:**  
Improving overall code security by implementing as many of the suggested changes as possible. These hardening measures could help to mitigate the potential for higher severity findings in the future.

## RISK METHODOLOGY

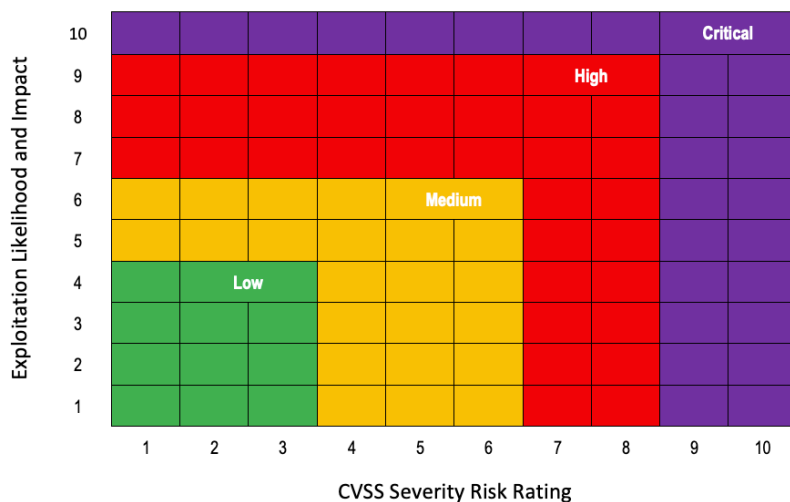
Information security is not about eliminating risk. It is founded upon the science and discipline of risk management. This is an important distinction because computer systems are inherently designed to share information while security strives to guard it. Therefore, it is management’s role to weigh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

### 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.

### 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.

## ENGAGEMENT OVERVIEW

### SCOPE

Community Labs and Open Security collaboratively defined the scope of this project to include a secure source code review of the ArConnect browser extension. Additional information on scope can be found in [Appendix B: Detailed Scope](#).

The focus was on vulnerabilities within the browser extension only. As such, all domains within the ArConnect source code were considered out-of-scope as well as the open-source libraries utilized within the source code. Lastly, any vulnerability tests were conducted in a local environment, separated from the view of the public (i.e. no data was written to a public blockchain or uploaded to a Third-Party website). There were no additional restrictions on the scope of this penetration test.

Open Security engineers were in real-time communications with key stakeholders throughout the engagement and reported all **Critical** findings as soon as they were identified.

### RULES OF ENGAGEMENT

Open Security did not use any cyberspace methodologies which could hinder real-world operations. This assessment consisted of a phase of *secure source code review* for vulnerability identification and evaluation of the ArConnect browser extension.

An active exploitation phase was then performed using a modified development version of the browser extension where exploitation of all non-denial of service vulnerabilities was considered in-scope.

### SECURITY SNAPSHOT

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 do not hesitate to contact Open Security.

## SECURITY ROADMAP

### OVERVIEW

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.

### SUMMARY OF FINDINGS

During the 2024 Q2 ArConnect Source Code Audit, Open Security discovered 0 **Critical**-, 1 **High**-, and 0 **Medium**-severity findings. An additional two vulnerabilities were rated as presenting 0 **Low** risk. Findings are listed once even if they pertain to multiple systems across the network and vulnerabilities of common criteria are grouped together.

<b>Finding ID</b>	<b>Description</b>	<b>Severity</b>	<b>Remediation</b>
ARCO-2024-01	Vulnerabilities in Open Source Dependencies	High	Yes
ARCO-2024-02	Unexploitable DOM-based XSS	Informational	N/A
ARCO-2024-03	getActiveKeyfile and freeDecryptedWallet function usage	Informational	N/A
ARCO-2024-04	React State kept in-memory after use	Informational	N/A
ARCO-2024-05	HTML a tag with REL properties misspelled	Informational	Yes
ARCO-2024-06	Suggested Hardening Changes	Informational	N/A

## 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.***



## 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.

<b>Finding ID</b>	<b>Description</b>	<b>Severity</b>	<b>Remediation</b>
ARCO-2024-01	Vulnerabilities in Open Source Dependencies	High	Yes

## ARCO-2024-01: VULNERABILITIES IN OPEN SOURCE DEPENDENCIES (REMIEDIATED)

**VULNERABILITY RATING:** High  
**CVE/CWE:** CWE-1395: Dependency on Vulnerable Third-Party Component  
**DISCOVERY METHOD:** Yarn Audit Command

### REMIEDIATION STATUS:

Remediation testing was performed on May 3, 2024, utilizing the source code from commit <https://github.com/arconnectio/ArConnect/pull/272/commits/ea46e34460aff8f2ddd4145f4a6f4e78dc39e0c8>. As shown in Figure 1, no vulnerabilities were found running a `yarn audit` command, fully remediating this finding.

```
> yarn audit
yarn audit v1.22.19
0 vulnerabilities found - Packages audited: 1707
🌟 Done in 0.71s.
```

Figure 1 - Yarn Audit Output After Remediation

### 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.

### ANALYSIS:

The ArConnect Browser Extension has many out-of-date dependencies, with 4 Critical-, 15 High-, 36 Moderate-, and 14 Low-severity vulnerabilities being reported by the `yarn audit` command. The impact of these vulnerabilities is highly variable, though the Critical vulnerabilities include dangerous issues such as 'Babel vulnerable to arbitrary code execution when compiling specifically crafted malicious code' (see NPM Advisory in References).

### REPRODUCTION STEPS:

Run the command `yarn audit`, noting the response. A large amount of output will be shown, similar to below.

```
... SNIPPED for brevity
69 vulnerabilities found - Packages audited: 1759
Severity: 14 Low | 36 Moderate | 15 High | 4 Critical
🌟 Done in 4.73s.
```

### 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 (69 vulnerabilities found, 14 Low | 36 Moderate | 15 High | 4 Critical)
2. `yarn upgrade`
3. After executing these commands, the vulnerabilities are reduced to 13 vulnerabilities (0 Low, 4 Moderate, 9 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://www.dni.gov/files/NCSC/documents/supplychain/Software_Supply_Chain_Attacks.pdf)
- <https://www.npmjs.com/advisories/1096886>
- <https://cwe.mitre.org/data/definitions/1395.html>

## 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.

<b>Finding ID</b>	<b>Description</b>	<b>Severity</b>	<b>Remediation</b>
ARCO-2024-02	Unexploitable DOM-based XSS	Informational	N/A
ARCO-2024-03	getActiveKeyfile and freeDecryptedWallet function usage	Informational	N/A
ARCO-2024-04	React State kept in-memory after use	Informational	N/A
ARCO-2024-05	HTML a tag with REL properties misspelled	Informational	Yes
ARCO-2024-06	Suggested Hardening Changes	Informational	N/A

## ARCO-2024-02: UNEXPLOITABLE DOM-BASED XSS

**VULNERABILITY RATING:** Informational  
**CVE/CWE:** CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')  
**DISCOVERY METHOD:** Manual Testing

### REMEDIATION STATUS:

*This section of the report is reserved for future use to document remediation steps and status.*

### DESCRIPTION:

In DOM-based XSS, the client performs the injection of XSS into the page (vs server-side rendering performing the injection). DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as JavaScript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

The ArConnect browser extension codebase includes the user of `elem.innerHTML` to extract an `img` tag from a Third-Party XML RSS feed (Permaweb News Feed - see Reference link). If the Permaweb News Feed website were compromised an attacker could change the XML RSS feed to include potentially malicious HTML tags to perform an XSS attack. This would affect all ArConnect browser extension users that click on the Explore button within the browser extension.

### ANALYSIS:

Due to the XSS being blocked by the Content Security Policy set within the ArConnect browser extension, this attack vector was not possible to exploit. If this XSS attack vector were not blocked, it may be possible to trick the user into giving the malicious actor their private key and password, or abuse other functionality of the browser extension.

### REPRODUCTION STEPS:

Verify within the `build/chrome-mv3-dev/manifest.json` file the following entry exists:

NOTE: in the dev build `http://localhost` will also be included. Even with this present the XSS POC failed

```
...SNIPPET...  
  
"content_security_policy": {  
  "extension_pages": "script-src 'self'; object-src 'self';"  
}  
  
...SNIPPET...
```

### RECOMMENDATION:

Currently no code changes are required, however there is a suggestion which would remove the `innerHTML` usage altogether. This is an informational finding to highlight the importance of the Content

Security Policy setting within the `build/chrome-mv3-dev/manifest.json` file within the ArConnect browser extension. Consider adding developer documentation and/or comments where appropriate to ensure the Content Security Policy is not modified to allow an XSS attack such as this.

The following example shows how to potentially avoid using the `innerHTML` function to extract the same information from the RSS feed. This would harden the code further by using the `DOMParser` web api.

```
function loadRSSFeed(url) {
  fetch(url)
    .then(response => response.text()) // Get the response and convert it
to text
    .then(str => {
      // Parse the XML string
      const parser = new DOMParser();
      const xmlDoc = parser.parseFromString(str, "text/xml");

      // Navigate through the XML document
      const items = xmlDoc.getElementsByTagName("item");
      if (items.length > 1) {
        // Get the second item
        const secondItem = items[1];

        // Extract the <description> element
        const description =
secondItem.getElementsByTagName("description")[0];
        if (description) {
          const descHtml = description.textContent;

          // Further parse the description to extract the <img> tag
          const htmlDoc = parser.parseFromString(descHtml,
"text/html");

          const img = htmlDoc.getElementsByTagName("img")[0];

          // Use the `img` value
          if (img) {
            console.log("Image src:", img.src); // Output the src
of the <img> tag
          } else {
            console.log("No image found in the description.");
          }
        } else {
          console.log("No description tag found.");
        }
      } else {
        console.log("No items found in the feed.");
      }
    })
    .catch(error => {
      console.error("Failed to load RSS feed:", error);
    });
}

// Example usage
```

```
loadRSSFeed('https://permaweb.news/feed');
```

#### REFERENCES:

- <https://cwe.mitre.org/data/definitions/79.html>
- <https://permaweb.news/feed>
- <https://developer.chrome.com/docs/extensions/reference/manifest/content-security-policy>
- <https://developer.mozilla.org/en-US/docs/Web/API/DOMParser/parseFromString>

## ARCO-2024-03: GETACTIVEKEYFILE AND FREEDECRYPTEDWALLET FUNCTION USAGE

**VULNERABILITY RATING:** Informational  
**CVE/CWE:** CWE-316: Cleartext Storage of Sensitive Information in Memory  
**DISCOVERY METHOD:** Manual Testing

### REMEDIATION STATUS:

*This section of the report is reserved for future use to document remediation steps and status.*

### 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.

`getActiveKeyfile` and `freeDecryptedWallet` are JavaScript functions within the ArConnect browser extension. These functions are used to get the current private key and then free the memory to reduce the chance of leaking the users private key during a memory dump. This was added in the last security review to reduce the amount of time the sensitive information was kept in memory. The suggestion here is to continue to use these functions but consider switching to a `try/catch/finally` code pattern when these functions are called. In some cases there is potential for a JavaScript `await` call to throw an error which could prevent `freeDecryptedWallet` from being executed. Switching to a `try/catch/finally` pattern would fit in most use-cases and ensure that the `freeDecryptedWallet` function is called irrespective of any errors.

### ANALYSIS:

The cleartext storage of sensitive information in the browser extension's memory can lead to the disclosure of sensitive information such as the private key of the wallet. Attackers who gain access to a user's system, through malware or other means, could extract this sensitive data directly from memory eg, by creating a memory dump.

### REPRODUCTION STEPS:

The following are commands useful for finding all references in the codebase to the two functions:

```
# grep -rli "freedecryptedwallet" src
src/subscriptions/payments.ts
src/components/dashboard/subsettings/WalletSettings.tsx
src/components/arlocal/Transaction.tsx
src/lib/avatar.ts
src/api/modules/signature/signature.background.ts
src/api/modules/dispatch/dispatch.background.ts
src/api/modules/dispatch/allowance.ts
src/api/modules/private_hash/private_hash.background.ts
```



```
src/api/modules/sign_data_item/sign_data_item.background.ts
src/api/modules/public_key/public_key.background.ts
src/api/modules/verify_message/verify_message.background.ts
src/api/modules/encrypt/encrypt.background.ts
src/api/modules/sign_message/sign_message.background.ts
src/api/modules/sign/fee.ts
src/api/modules/sign/sign.background.ts
src/api/modules/decrypt/decrypt.background.ts
src/wallets/index.ts
src/wallets/encryption.ts
src/wallets/auth.ts
src/routes/popup/send/auth.tsx
src/routes/popup/send/confirm.tsx

---

# NOTE: this list has parity with the above freedecryptedwallet list, with the
# noted exception of the aoToken/ao.ts file
# grep -rli "getActiveKeyfile" src
src/subscriptions/payments.ts
src/components/arlocal/Transaction.tsx
src/lib/avatar.ts
src/api/modules/signature/signature.background.ts
src/api/modules/dispatch/dispatch.background.ts
src/api/modules/private_hash/private_hash.background.ts
src/api/modules/sign_data_item/sign_data_item.background.ts
src/api/modules/public_key/public_key.background.ts
src/api/modules/verify_message/verify_message.background.ts
src/api/modules/encrypt/encrypt.background.ts
src/api/modules/sign_message/sign_message.background.ts
src/api/modules/sign/fee.ts
src/api/modules/sign/sign.background.ts
src/api/modules/decrypt/decrypt.background.ts
src/wallets/index.ts
src/tokens/aoTokens/ao.ts
src/routes/popup/send/auth.tsx
src/routes/popup/send/confirm.tsx
```

#### RECOMMENDATION:

Consider changing to a pattern such as the following:

```
try{
  const decryptedWallet = await getActiveKeyfile().catch((e) => {
    isNotCancelError(e);

    // if there are no wallets added, open the welcome page
    browser.tabs.create({ url: browser.runtime.getURL("tabs/welcome.html") });

    throw new Error("No wallets added");
  });

  // ... do things with the decrypted wallet here
  // eg:
  await submitTx(convertedTransaction, arweave, type);
```

```
}catch(e){  
  // ... console.log out the error or other error handling  
}finally{  
  // finally, remove wallet from memory  
  freeDecryptedWallet(decryptedWallet);  
}
```

Consider the try/catch/finally pattern to the following files:

- `src/api/modules/sign/sign.background.ts`
- `src/lib/avatar.ts` line 56
- `src/routes/popup/send/auth.tsx` lines 253, 319
- `src/subscriptions/payments.ts`
- `src/routes/popup/send/confirm.tsx`

Consider adding the following comments, for future reviews:

- `src/tokens/aoTokens/ao.ts` consider adding comment as to this not needing the `freeDecryptedWallet`, but would be required after `sendAoTransfer` is used (related to `src/routes/popup/send/confirm.tsx` usage)

#### REFERENCES:

- <https://cwe.mitre.org/data/definitions/316.html>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory\\_management](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_management)

## ARCO-2024-04: REACT STATE KEPT IN-MEMORY AFTER USE

**VULNERABILITY RATING:** Informational  
**CVE/CWE:** CWE-316: Cleartext Storage of Sensitive Information in Memory  
**DISCOVERY METHOD:** Manual Testing

### REMEDIATION STATUS:

*This section of the report is reserved for future use to document remediation steps and status.*

### 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.

The React `useState` hook is used throughout the browser extension for storing various user input and other values. This finding considers that those values may stay in memory longer than necessary, which could create a situation where a memory dump would reveal sensitive information.

### ANALYSIS:

The cleartext storage of sensitive information in the browser extension's memory can lead to the disclosure of sensitive information such as the private key of the wallet. Attackers who gain access to a user's system, through malware or other means, could extract this sensitive data directly from memory (e.g. by creating a memory dump).

### REPRODUCTION STEPS:

The following list highlights the files with React `state` which stores sensitive information:

- `src/components/SeedInput.tsx` consider clearing `words` state after use
- `src/routes/auth/connect.tsx` line 114 reset the `passwordInput.state` after use
- `src/components/dashboard/subsettings/AddWallet.tsx` line 151 reset the `passwordInput.state` after use
- `src/routes/auth/allowance.tsx` line 80 reset the `passwordInput.state` after use
- `src/routes/auth/unlock.tsx` line 30 reset the `passwordInput.state` after use
- `src/routes/popup/unlock.tsx` line 44 reset the `passwordInput.state` after use
- `src/routes/popup/send/auth.tsx` line 275 reset the `passwordInput.state` after use
- `src/routes/popup/send/confirm.tsx` line 420 reset the `passwordInput.state` after use
- `src/routes/welcome/load/password.tsx` line 36 reset the `passwordInput.state` and `validPasswordInput.state` after use

## RECOMMENDATION:

Consider using the `useState` function declared for the state variables to a new default value such as `AAAAAAAAAAAAAAAAAAAAAAAAAAAA`.

An example React component which would allow the user to enter a Password and then overwrite the password after the user pressed a Submit button is shown below. Once the password value is finished being used, the state is overwritten with a default value.

```
import React, { useState } from 'react';

function PasswordForm() {
  // Create a state variable 'password' and a function to update it
  const [password, setPassword] = useState('');

  // Function to handle the input change
  const handleInputChange = (event) => {
    setPassword(event.target.value);
  };

  // Function to handle the button click for form submission
  const handleSubmit = () => {
    alert(`Password Submitted: ${password}`); // Alert the submitted password
    (or handle as needed)

    // Suggested Mitigation:
    // once finished using the password, overwrite it's value to all 'A'
    // characters, with the same length as before
    setPassword(prev => {
      return 'A'.repeat(prev.length)
    })
  };

  return (
    <div>
      <input
        type="password"
        value={password}
        onChange={handleInputChange} // Update state on input change
        placeholder="Enter your password"
      />
      <button onClick={handleSubmit}>Submit</button> // Button to submit the
      password
    </div>
  );
}

export default PasswordForm;
```

## REFERENCES:

- <https://cwe.mitre.org/data/definitions/316.html>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory\\_management](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_management)

## ARCO-2024-05: HTML A TAG WITH REL PROPERTIES MISPELLED (REMIEDIATED)

VULNERABILITY RATING:	Informational
CVE/CWE:	CWE-1022: Use of Web Link to Untrusted Target with window.opener Access
DISCOVERY METHOD:	Manual Testing

### REMIEDIATION STATUS:

Remediation testing was performed on May 3, 2024, utilizing the source code from commit <https://github.com/arconnectio/ArConnect/pull/272/commits/9e34137462bad82d95f5692fbbdc4734b8a3b6>. All instances within the source code where the `rel` properties were misspelled have been addressed, fully remediating this finding.

### DESCRIPTION:

An HTML `<a>` tag is used to create hyperlinks in web documents, allowing users to navigate from one web page or resource to another. When the `<a>` tag includes the `rel` attribute with the values `noopener` and `noreferrer`, it enhances security and privacy. The `noopener` value prevents the newly opened page from being able to access the originating page's window object via `window.opener`, thereby safeguarding against malicious scripts. The `noreferrer` value stops the browser from sending the HTTP referrer header to the new page, ensuring that the destination page does not receive any information about the source page. This combination is particularly useful for links to external sites, improving security and user privacy.

### ANALYSIS:

When a user clicks a link to an external site ("target"), the `target="_blank"` attribute causes the target site's contents to be opened in a new window or tab, which runs in the same process as the original page. The `window.opener` object records information about the original page that offered the link. If an attacker can run a script on the target page, then they could read or modify certain properties of the `window.opener` object, including the location property - even if the original and target site are not the same origin. An attacker could modify the location property to automatically redirect the user to a malicious site, e.g. as part of a phishing attack. Since this redirect happens in the original window/tab (which is not necessarily visible, since the browser is focusing the display on the new target page) the user might not notice any suspicious redirection.

### REPRODUCTION STEPS:

- `src/routes/popup/token/[id].tsx` line 523 appears to have a typo for the `rel` properties of the `a` tag with `rel="noopen noreferrer"` which should be `rel="noopener noreferrer"`.

### RECOMMENDATION:

- Set the `rel` attribute to `rel="noopener noreferrer"` for all HTML `a` tags

## REFERENCES:

- <https://github.com/arconnectio/ArConnect/blob/35cce3b9312915bac93fda570115ad7386bbdbfe/src/routes/popup/token/%5Bid%5D.tsx#L523>
- <https://cwe.mitre.org/data/definitions/1022.html>

## ARCO-2024-06: SUGGESTED HARDENING CHANGES

**VULNERABILITY RATING:** Informational  
**CVE/CWE:** N/A  
**DISCOVERY METHOD:** Manual Testing

### REMEDIATION STATUS:

*This section of the report is reserved for future use to document remediation steps and status.*

### DESCRIPTION:

This informational finding aims to highlight some potential changes that could harden the ArConnect browser extension further. These changes may impact functionality of the browser extension in some circumstances.

### AFFECTED ASSETS:

### ANALYSIS:

The `externally_connectable` property within the `manifest.json` of the browser extension declares which extensions and web pages can connect to your extension using `runtime.connect()` and `runtime.sendMessage()`. The default value is to allow all extensions can connect, but no web pages can connect. Disabling this functionality would disabled Third-Party browser extensions to directly utilize functionality of the ArConnect browser extension.

### REPRODUCTION STEPS:

1. Run the `yarn dev:chrome` command
2. View the contents of the `build/chrome-mv3-dev/manifest.json` file. Note that no `externally_connectable` property is declared, which falls back to a default of allowing connections from other browser extensions..

### RECOMMENDATION:

Consider setting `"externally_connectable": {}` within the `build/chrome-mv3-dev/manifest.json`

### REFERENCES:

- [https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/externally\\_connectable](https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/externally_connectable)
- <https://developer.chrome.com/docs/extensions/reference/manifest/externally-connectable>

## METHODOLOGY

The Secure Code Audit was conducted in three overall phases. The first and second phase were each performed purely from a manual secure source-code review, with in-depth analysis of discovered vulnerabilities. The third phase consisted of attempts to actively exploit previously identified vulnerabilities.

### PHASE 1 – OVERALL SECURE CODE REVIEW

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 C – ArConnect – Secure Code Review Checklist.xlsx*.

The first step taken is to review the code for familiarization of the components involved, the general software patterns and technologies involved. This includes looking over every line of code within the ArConnect GitHub repository, noting any complicated or difficult to understand code for further review. This initial review aids in the direction of any POCs attempted (see [Phase 3 – Attempted Exploitation and Creation of POC's for Identified](#)). Identification of commonly used files for tasks such as decrypting the users private key are also noted.

Once the overall familiarization is complete, the review shifted into a targeted search approach whereby the identified important functions were further scrutinized, reviewing the definition of these functions and their usage throughout the codebase. Additional searches for commonly vulnerable and/or insecure Web APIs were also conducted. VsCode was utilized as the IDE for these code reviews as it helps to apply more context to the strongly typed code and makes code navigation efficient. Common Linux utilities such as `grep` were also used for finding information within the source code.

### PHASE 2 – CODE REVIEW OF CHANGES SINCE LAST REVIEW

GitHub provides comparison functionality within their website wherein a user can compare code changes between two branches. This was utilized to review the code changes since the last secure code review and subsequent remediation of the previous findings. See [Appendix B: Detailed Scope](#) for the GitHub compare URL used.

Utilizing this compare functionality helped narrow down specific changes and areas to further investigate from the previous review. During this phase more notes were taken, ensuring focus on new areas of code development – which are more likely to have security vulnerabilities than previously reviewed code.



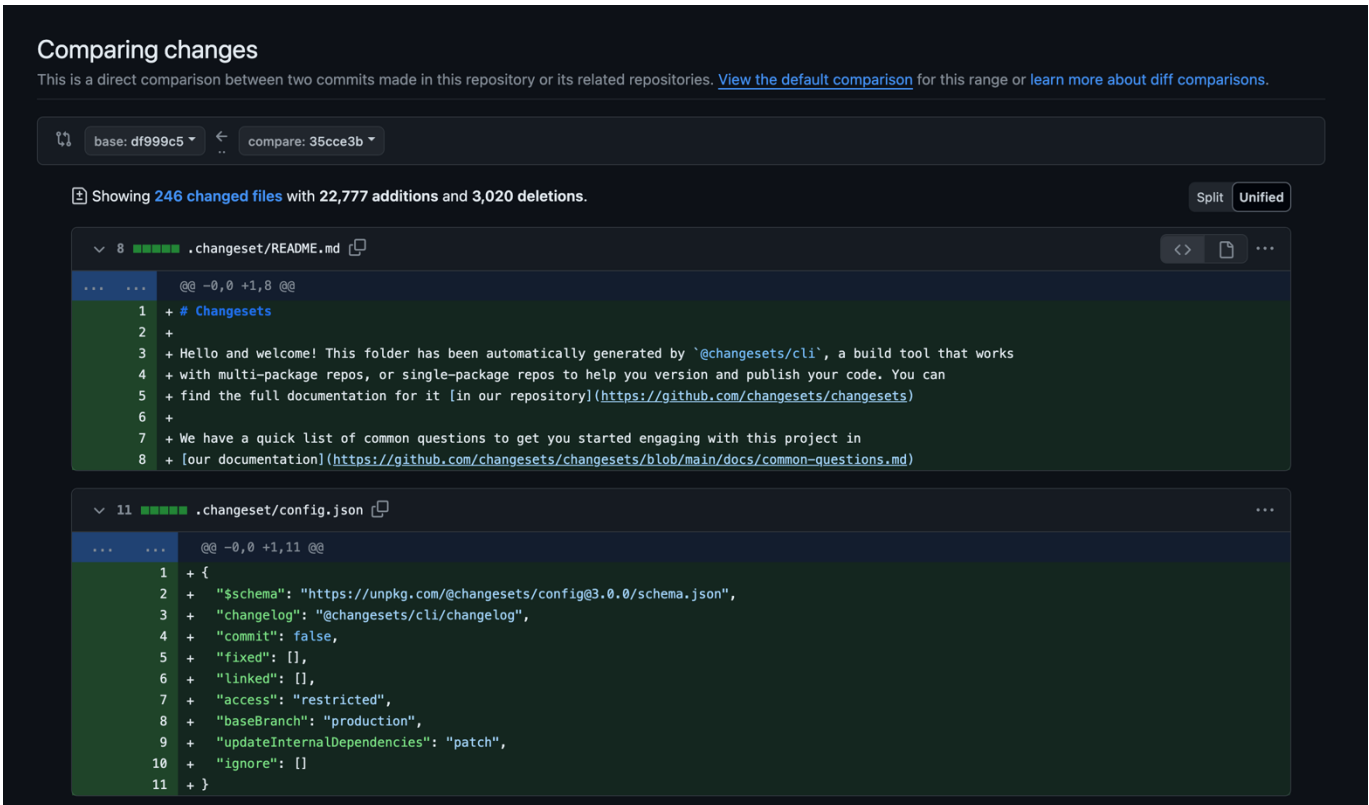


Figure 2 – GitHub compare feature showing some of the changes from the last remediation commit

### PHASE 3 – ATTEMPTED EXPLOITATION AND CREATION OF POC’S FOR IDENTIFIED VULNERABILITIES

This phase consists of attempting to exploit some potentially vulnerable code paths by building and installing a development version of the ArConnect browser extension as well as making modifications to prove any vulnerabilities.

1. An exploit was attempted with injecting malicious payloads into an SVG loaded on the **Explore** page, attempting to provide a XSS attack vector. The exploit attempt failed due to Content Security Policy settings.
2. An exploit was attempted with making a malicious Subscription via both a malicious JavaScript URL for the Management URL, as well as a malicious payload in an SVG file as an XSS attack vector. This exploit attempt also failed.

```

> yarn install
yarn install v1.22.19
[1/4] 🔍 Resolving packages...
warning Resolution field "@parcel/runtime-js@2.8.3" is incompatible with requested version "@parcel/runtime-js@2.9.3"
warning Resolution field "@parcel/runtime-js@2.8.3" is incompatible with requested version "@parcel/runtime-js@2.9.3"
[2/4] 📦 Fetching packages...
warning node-object-hash@3.0.0: The engine "pnpm" appears to be invalid.
[3/4] 🔗 Linking dependencies...
warning "@parcel/runtime-js > @parcel/plugin > @parcel/types > @parcel/cache@2.8.3" has unmet peer dependency "@parcel/core@^2.8.3".
warning "@parcel/runtime-js > @parcel/plugin > @parcel/types > @parcel/fs@2.8.3" has unmet peer dependency "@parcel/core@^2.8.3".
warning "@parcel/runtime-js > @parcel/plugin > @parcel/types > @parcel/package-manager@2.8.3" has unmet peer dependency "@parcel/core@^2.8.3".
warning "@parcel/runtime-js > @parcel/plugin > @parcel/types > @parcel/workers@2.8.3" has unmet peer dependency "@parcel/core@^2.8.3".
warning " > react-qr-reader@2.2.1" has incorrect peer dependency "react@~16".
warning " > react-qr-reader@2.2.1" has incorrect peer dependency "react-dom@~16".
warning "@keystonehq/arweave-keyring > @keystonehq/sdk > qrcode.react@1.0.1" has incorrect peer dependency "react@^15.5.3 || ^16.0.0 || ^17.0.0".
warning " > styled-components@5.3.6" has unmet peer dependency "react-is@>= 16.8.0".
warning Workspaces can only be enabled in private projects.
[4/4] ⚙️ Building fresh packages...
$ husky install
husky - Git hooks installed
🚀 Done in 13.00s.

```

Figure 3 – Yarn install command is used to prepare all the necessary open-source packages used for building the ArConnect browser extension

```

> yarn dev:chrome
yarn run v1.22.19
$ plasmio dev
● Plasmio v0.82.0
● The Browser Extension Framework
● WARN | A new version of plasmio is available: v0.86.1
| Run "yarn add plasmio@0.86.1" to update
● INFO | Starting the extension development server...
● INFO | Loaded environment variables from: {}
● DONE | Extension re-packaged in 3149ms! 🚀

```

Figure 4 – Yarn dev:chrome is the command used to create a development build for Google Chrome, shown here is the successful execution of this command.

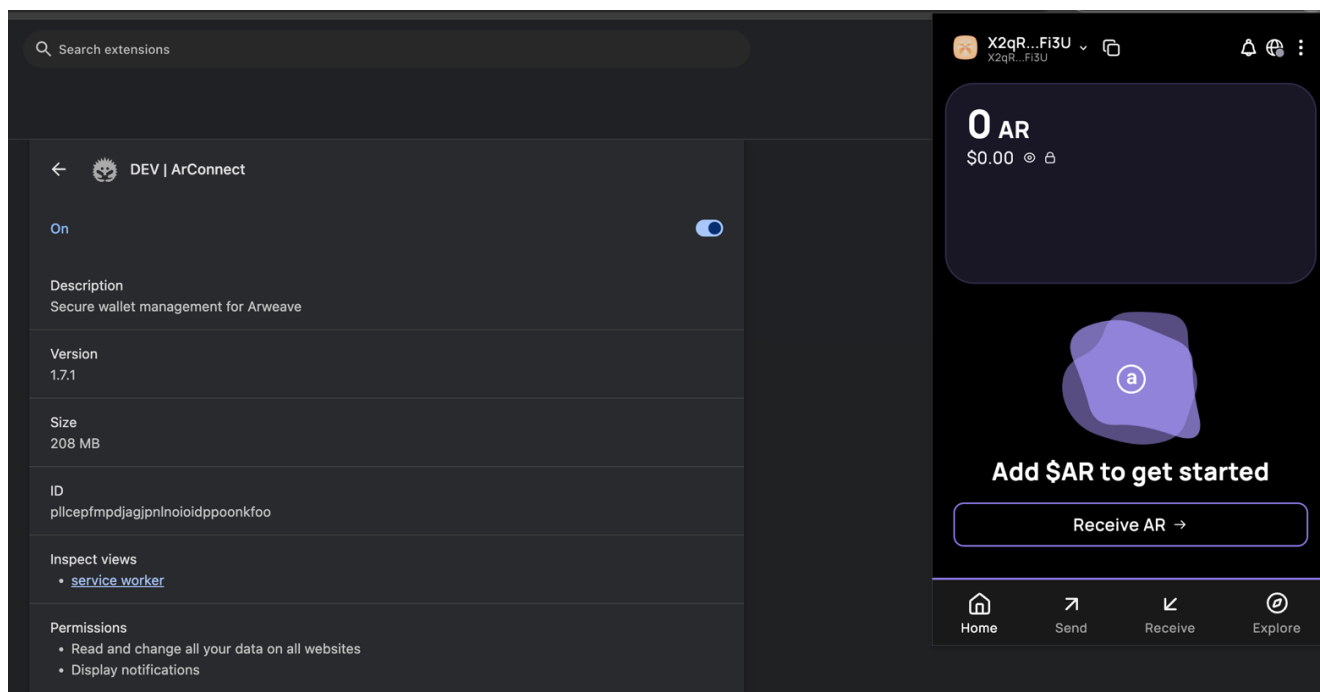
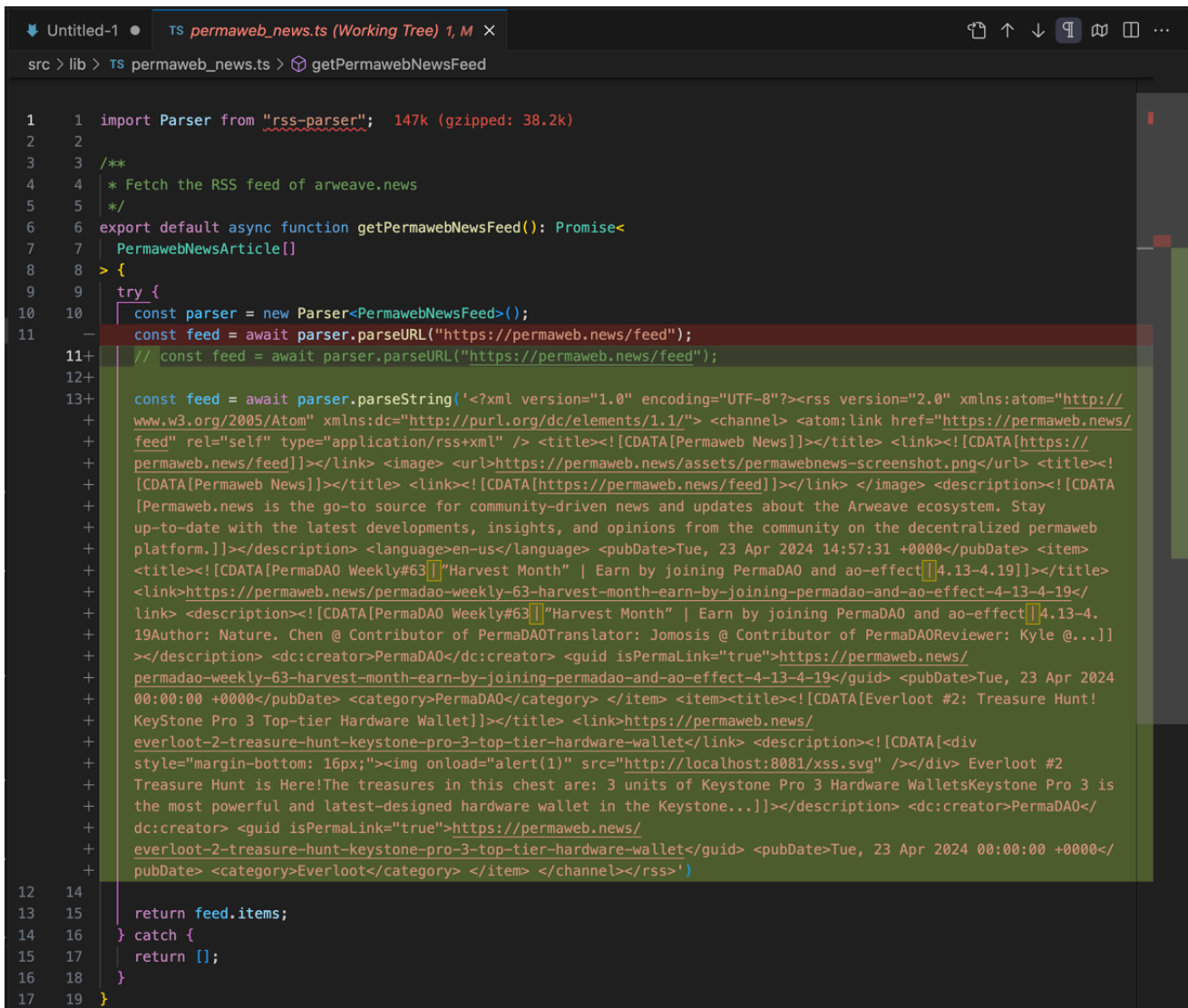


Figure 5 – Once the ArConnect browser extension is installed and running, after setting up the wallet and password this is the main view. This image shows the successful operation of the browser extension as used for the security review.

## POC 1 – Explore Page PermaWeb News RSS Feed XSS Injection Attempt

Modifying the source code to inject a custom response (to simulate the PermaWeb News feed Third-Party website being modified by malicious actors) was tested to attempt to perform a DOM-based XSS via the `innerHTML` usage within the ArConnect browser extension. While the SVG was loaded, the XSS attempt was not executed due to the Content Security Policy setting within the `manifest.json`. This was the basis of *ARCO-2024-02 - Unexploitable DOM-based XSS* suggestion. The following screenshots highlight the code changes and POC tested.



```

1 1 import Parser from "rss-parser"; 147k (gzipped: 38.2k)
2 2
3 3 /**
4 4  * Fetch the RSS feed of arweave.news
5 5  */
6 6 export default async function getPermaWebNewsFeed(): Promise<
7 7   PermaWebNewsArticle[]
8 8 > {
9 9   try {
10 10    const parser = new Parser<PermaWebNewsFeed>();
11 11    const feed = await parser.parseURL("https://permaweb.news/feed");
12 12    // const feed = await parser.parseURL("https://permaweb.news/feed");
13 13    const feed = await parser.parseString('<?xml version="1.0" encoding="UTF-8"?><rss version="2.0" xmlns:atom="http://
+ www.w3.org/2005/Atom" xmlns:dc="http://purl.org/dc/elements/1.1/"> <channel> <atom:link href="https://permaweb.news/
+ feed" rel="self" type="application/rss+xml" /> <title><![CDATA[PermaWeb News]]></title> <link><![CDATA[https://
+ permaweb.news/feed]]></link> <image> <url>https://permaweb.news/assets/permawebnews-screenshot.png</url> <title><![
+ [CDATA[PermaWeb News]]></title> <link><![CDATA[https://permaweb.news/feed]]></link> </image> <description><![CDATA
+ [PermaWeb News is the go-to source for community-driven news and updates about the Arweave ecosystem. Stay
+ up-to-date with the latest developments, insights, and opinions from the community on the decentralized permaWeb
+ platform.]]></description> <language>en-us</language> <pubDate>Tue, 23 Apr 2024 14:57:31 +0000</pubDate> <item>
+ <title><![CDATA[PermaDAO Weekly#63 | Harvest Month | Earn by joining PermaDAO and ao-effect]]></title>
+ <link>https://permaweb.news/permaDAO-weekly-63-harvest-month-earn-by-joining-permaDAO-and-ao-effect-4-13-4-19</
+ link> <description><![CDATA[PermaDAO Weekly#63 | Harvest Month | Earn by joining PermaDAO and ao-effect]]></description>
+ <dc:creator>PermaDAO</dc:creator> <guid isPermaLink="true">https://permaweb.news/
+ permaDAO-weekly-63-harvest-month-earn-by-joining-permaDAO-and-ao-effect-4-13-4-19</guid> <pubDate>Tue, 23 Apr 2024
+ 00:00:00 +0000</pubDate> <category>PermaDAO</category> </item> <item><title><![CDATA[Everloot #2: Treasure Hunt!
+ Keystone Pro 3 Top-tier Hardware Wallet]]></title> <link>https://permaweb.news/
+ everloot-2-treasure-hunt-keystone-pro-3-top-tier-hardware-wallet</link> <description><![CDATA[<div
+ style="margin-bottom: 16px;"></div> Everloot #2
+ Treasure Hunt is Here! The treasures in this chest are; 3 units of Keystone Pro 3 Hardware Wallets Keystone Pro 3 is
+ the most powerful and latest-designed hardware wallet in the Keystone...]]></description> <dc:creator>PermaDAO</
+ dc:creator> <guid isPermaLink="true">https://permaweb.news/
+ everloot-2-treasure-hunt-keystone-pro-3-top-tier-hardware-wallet</guid> <pubDate>Tue, 23 Apr 2024 00:00:00 +0000</
+ pubDate> <category>Everloot</category> </item> </channel></rss>')
12 14
13 15   return feed.items;
14 16 } catch {
15 17   return [];
16 18 }
17 19 }

```

Figure 6 – VsCode git diff of the `permaweb_news.ts` file which was modified with a POC of a malicious payload which was attempted for the XSS attack. The main part to focus on is the `<img onload="alert(1)" ... />` tag that was inserted.

```

1  [?]xml version="1.0" standalone="no"?
2  DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3
4  <svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
5    <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />
6    <script type="text/javascript">
7      alert("xss");
8      console.log('svgxss');
9      fetch("http://localhost:8081/svgxssproof");
10   </script>
11 </svg>

```

Figure 7 – SVG files can contain malicious payloads which attempt to execute Javascript. This is a POC that was used during the testing. This SVG draws a rectangular blue box when rendered.

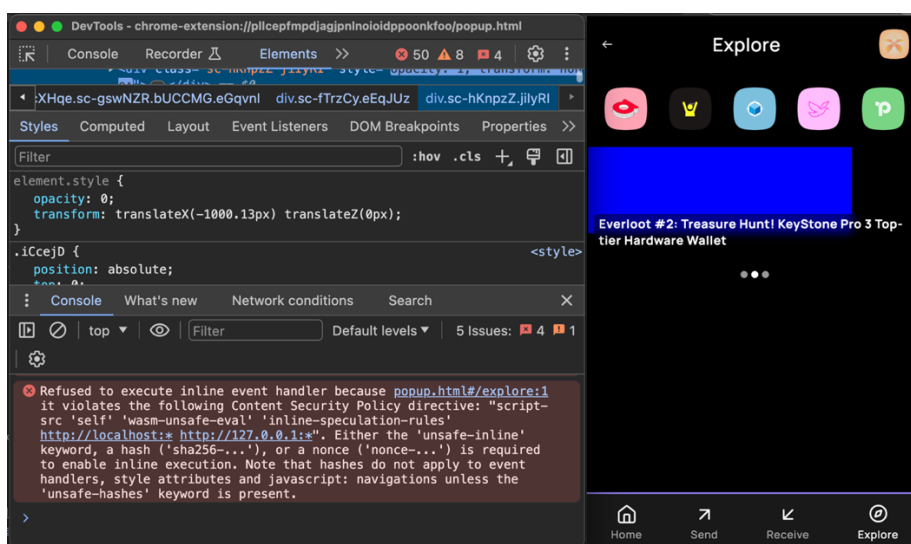


Figure 8 – Google Chrome’s Content Security Policy (applied by the *manifest.json*) is shown here blocking the XSS POC attempt. This screenshot also shows the SVG was loaded and rendered.

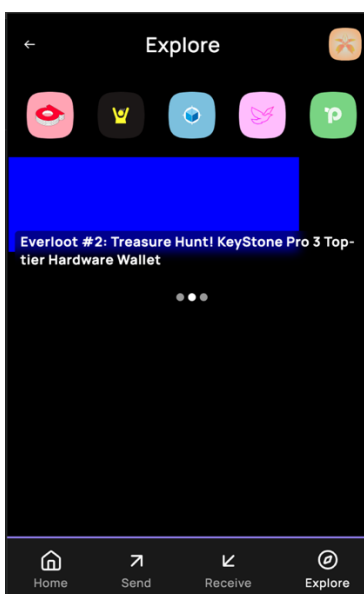
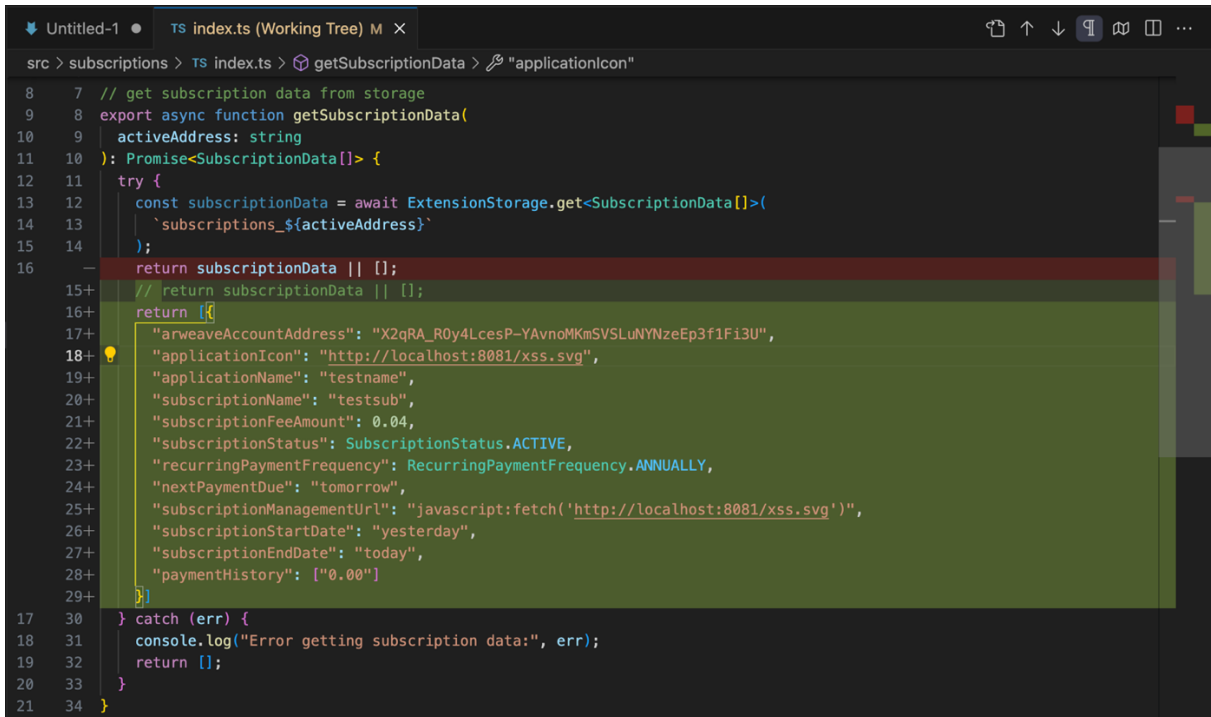


Figure 9 – The ArConnect browser extension is shown here after the User has clicked on the Explore tab. The SVG was rendered, proving the malicious SVG file was loaded properly however the XSS payload was blocked.

## POC 2 – XSS Attempts Within a Malicious Subscription

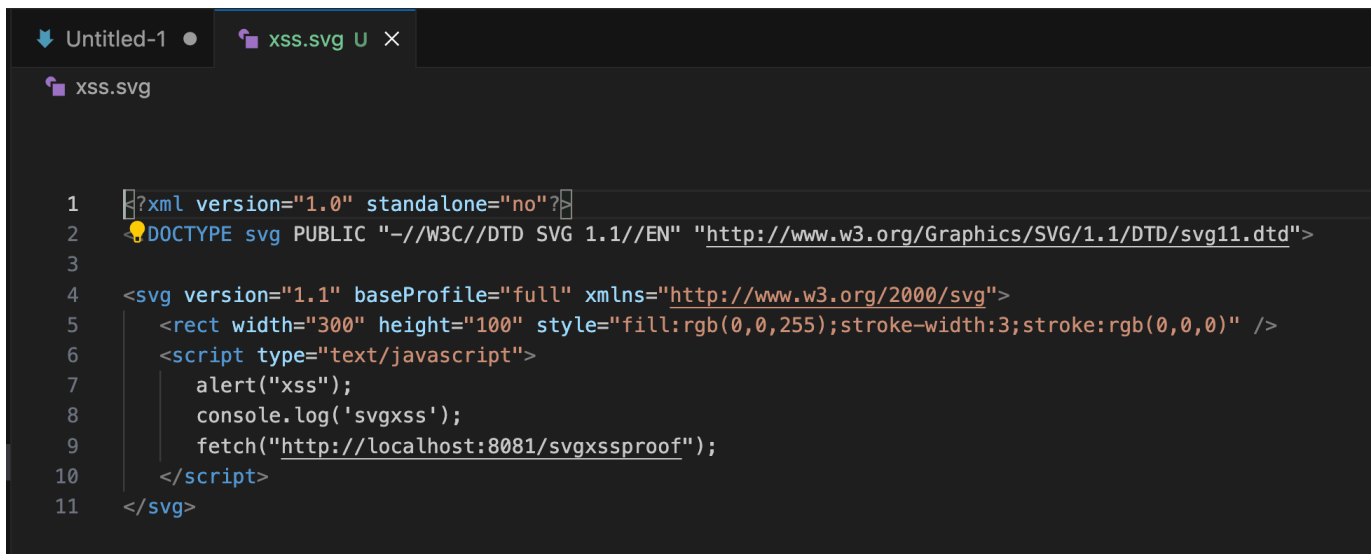
With the new Subscription feature, a POC was tested to ensure that an XSS attack was not possible using either an SVG with malicious payload for the Icon or an XSS attack via the `subscriptionManagementUrl`. All attempted XSS attacks were unsuccessful due to the security boundary enforcement within the browser extension sandbox. The following screenshots highlight the code changes and POC tested.



```

8 7 // get subscription data from storage
9 8 export async function getSubscriptionData(
10 9   activeAddress: string
11 10 ): Promise<SubscriptionData[]> {
12 11   try {
13 12     const subscriptionData = await ExtensionStorage.get<SubscriptionData[]>(
14 13       `subscriptions_${activeAddress}`
15 14     );
16 -   return subscriptionData || [];
16+  // return subscriptionData || [];
16+  return [
17+    "arweaveAccountAddress": "X2qRA_R0y4LcesP-YAvnoMKmSVSLuNYNzeEp3f1Fi3U",
18+    "applicationIcon": "http://localhost:8081/xss.svg",
19+    "applicationName": "testname",
20+    "subscriptionName": "testsub",
21+    "subscriptionFeeAmount": 0.04,
22+    "subscriptionStatus": SubscriptionStatus.ACTIVE,
23+    "recurringPaymentFrequency": RecurringPaymentFrequency.ANNUALLY,
24+    "nextPaymentDue": "tomorrow",
25+    "subscriptionManagementUrl": "javascript:fetch('http://localhost:8081/xss.svg')",
26+    "subscriptionStartDate": "yesterday",
27+    "subscriptionEndDate": "today",
28+    "paymentHistory": ["0.00"]
29+  ];
17 30 } catch (err) {
18 31   console.log("Error getting subscription data:", err);
19 32   return [];
20 33 }
21 34 }
  
```

Figure 10 – VsCode git diff showing the changes to the subscriptions/index.ts file that were used as a potentially malicious payload for the applicationIcon and subscriptionManagementUrl properties of a subscription.



```

1  <?xml version="1.0" standalone="no"?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3
4  <svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
5     <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />
6     <script type="text/javascript">
7         alert("xss");
8         console.log('svgxss');
9         fetch("http://localhost:8081/svgxssproof");
10    </script>
11 </svg>
  
```

Figure 11 - SVG files can contain malicious payloads which attempt to execute Javascript. This is a POC that was used during the testing. This SVG draws a rectangular blue box when rendered.

```
> python3 -m http.server 8081
Serving HTTP on :: port 8081 (http://[::]:8081/) ...
:::1 - - [26/Apr/2024 09:38:09] "GET /xss.svg HTTP/1.1" 304 -
```

Figure 12 – A Python3 http server console output is shown here, which was used to host the xss.svg image used in the various locations within the XSS tests.

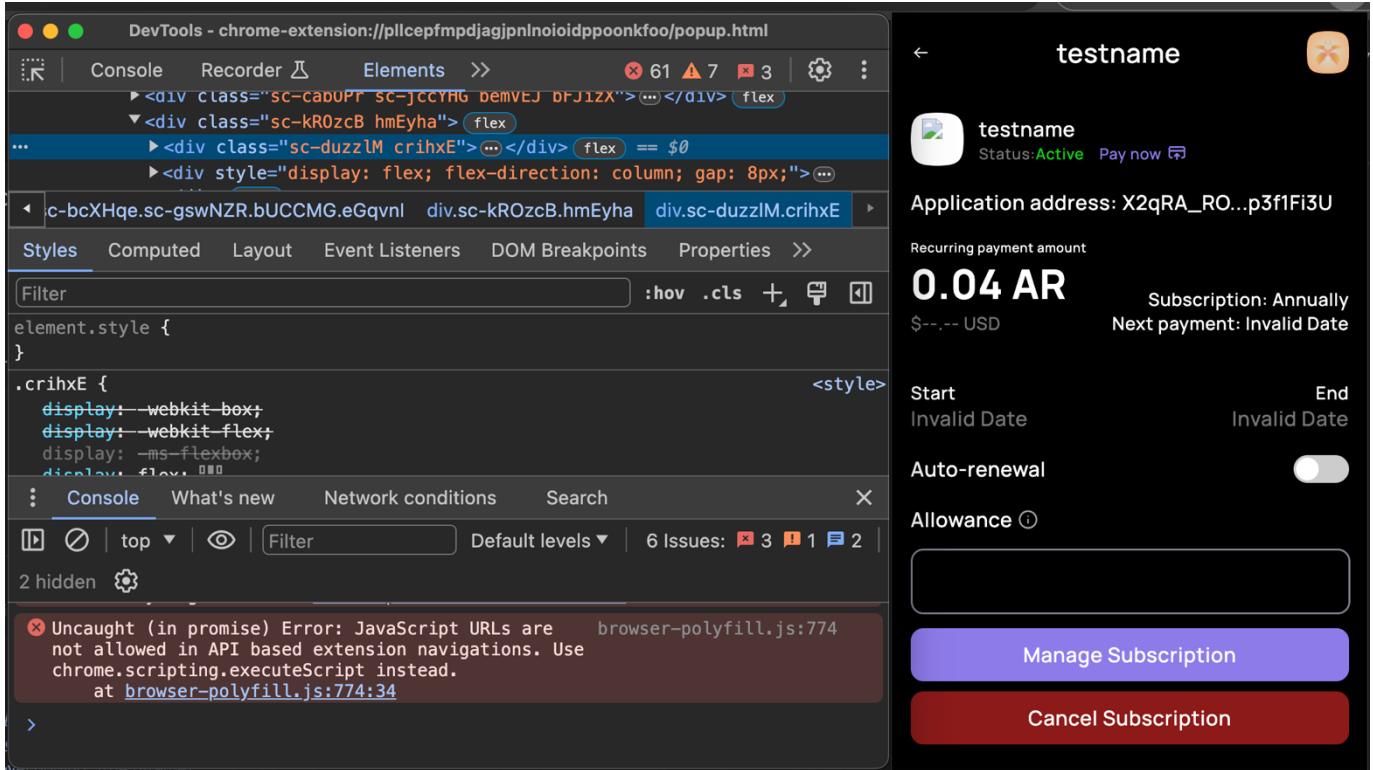


Figure 13 – Google Chrome developer console is shown rejecting the execution of the Javascript XSS payload due to the security restrictions placed on the extension.

## APPENDIX A: SECURITY TEAM SNAPSHOT

Passionate and forward-thinking, our team brings 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, understands the importance of information security and appreciates the opportunity to have worked on this engagement.



### VANCE WALSH | SECURITY ENGINEER

Leveraging a long-time software engineer career, Vance has applied his vast knowledge of various software technologies into a successful cybersecurity career. He's worked in various software projects ranging from traditional website development, blockchain dApp development, embedded systems as well as implementing AI technologies.

[Vance.Walsh@OpenSecurity.com](mailto:Vance.Walsh@OpenSecurity.com)



### JOSHUA CHRISTMAN | OSCP, OSCE | CHIEF OPERATIONS OFFICER

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. His Air Force career then continued into Offensive Cyber Operations, working for the premier offensive cyber unit in the Air Force. Since transitioning out of the Air Force he has run Red Teams, Application Security Teams, and Vulnerability Management Programs in the fintech industry. He is now responsible for all security engineering efforts at Open Security.

[Joshua.Christman@OpenSecurity.com](mailto:Joshua.Christman@OpenSecurity.com) | 719.375.9355

## APPENDIX B: DETAILED SCOPE

### URLS:

The following are the URLs of interest throughout this quarter's assessment.

- <https://github.com/arconnectio/ArConnect/tree/feat/arc-281-subscription-api>
- <https://github.com/arconnectio/ArConnect/compare/df999c5..35cce3b>
- <https://permaweb.news/feed>
- <https://chromewebstore.google.com/detail/arconnect/einnioafmpimabjcdiinlhmijaionap>