



# AVA LABS

## Avalanche Wallet Penetration Test

Prepared by: **Steven Walbroehl**  
Date of Engagement: **09.10.2020**  
Visit: [Halborn.com](https://Halborn.com)

# TABLE OF CONTENTS

|  |    |
|--|----|
| Document Revision History  | 3  |
| Contacts   | 3  |
| Executive Summary  | 4  |
| 1.1 Introduction   | 4  |
| 1.2 Objective  | 4  |
| 1.3 SCOPE  | 5  |
| 1.4 Assessment Summary And Findings Overview                           | 6  |
| 1.5 METHODOLOGY  | 7  |
| Findings & Technical Details   | 8  |
| Ava Labs Wallet - Open Source Dependency Security Checks               | 8  |
| Unused Dependencies - <b>MEDIUM</b>                                    | 8  |
| Ava Labs Wallet - Preventive Security Measures Implemented Into Wallet | 10 |
| Security Exploits - <b>NONE</b>  | 11 |
| Static Code Security Analysis Findings                                 | 13 |
| Insecure Use of Document.Location - <b>LOW</b>                         | 13 |
| Description:   | 13 |
| Code Locations: Src\Public\Serviceworker\Redirect.Html Line 260        | 13 |
| Remediation:   | 14 |
| Insecure Use Of Setattribute() - <b>LOW</b>                            | 14 |
| Description:   | 14 |
| Code Locations: \Src\Store\Index.Ts:474 And 475                        | 15 |
| Remediation:   | 15 |
| Local Storage Disclosure - <b>LOW</b>                                  | 16 |
| Description:   | 16 |
| Code Locations: Src\Src\Store\Index.Ts Line 413                        | 16 |
| Remediation:   | 16 |

## DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION      | DATE      | AUTHOR           |
|---------|-------------------|-----------|------------------|
| 0.1     | Document Creation | 9/10/2020 | Steven Walbroehl |
| 0.2     | Document Update   | 9/13/2020 | Steven Walbroehl |
| 1.0     | Final Draft       | 9/16/2020 | Steven Walbroehl |

## CONTACTS

| CONTACT          | COMPANY | EMAIL  |
|------------------|---------|--|
| STEVEN WALBROEHL | Halborn | <a href="mailto:Steven.Walbroehl@halborn.com">Steven.Walbroehl@halborn.com</a> |
| ROB BEHNKE       | Halborn | <a href="mailto:Rob.Behnke@halborn.com">Rob.Behnke@halborn.com</a>             |

## 1.1 INTRODUCTION

Halborn was engaged to perform penetration testing services for Ava Labs. A Penetration test is an active and hands-on engagement performed using deep security inspection to identify critical vulnerabilities before they are exploited by adversaries in the real world.

The Halborn team began a penetration test for the specified wallet web application, and the underlying systems, protocols, and code on September 8th, 2020, and concluded the testing on September 18th, 2020. To begin the test, Halborn was provided access to the web-application, and the Client's team provided the source code, and a test environment to conduct security testing using tools to detect possible vulnerabilities found in the application, and report the findings at the end of the engagement.

Key contacts at both Ava Labs and Halborn remained in communication each week of the engagement in order to keep in touch with status, progress items, and provide any troubleshooting, access, or material updates as required.

This Report provides the security findings; along with evidence, severity/risk levels (if applicable), vulnerability information, and remediation recommendations.

## 1.2 OBJECTIVE

This penetration test simulates the activities and tactics typically performed by threat actors, and helps to ensure the subject of the test reduces its exposure for the assets in scope. The objective of this assessment was to assess the overall security posture of the wallet. This includes determining the application's ability to resist common attack patterns, and identifying vulnerable areas in client code that may be exploited by a malicious user. These attack vectors may include stealing of private keys, weak cryptographic protocols, or common/advanced web application exploit techniques.

## 1.3 SCOPE

The penetration test included the following target application assets:

- A Web Application Client Wallet developed by AVA Labs  
<https://wallet.avax.network>

For the sprint, Halborn was provided by AVA Labs with several artifacts as required for a proper penetration test, including:

- Faucet to provide tokens for the test net in AVA Labs  
<https://faucet.avax.network/>
- API provided to send jsonrpc methods to the AVA blockchain  
[testapi.avax.network](https://testapi.avax.network)
- NPM package for avalance  
<https://www.npmjs.com/package/avalanche>
- Source code in scope for target assets web components.

## 1.4 ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW |
|----------|------|--------|-----|
| 0        | 0    | 1      | 3   |

| SECURITY ANALYSIS                      | RISK LEVEL |
|--|------------|
| Unused Code Dependencies Used In Build | Medium     |
| Insecure use of document.location      | Low        |
| Insecure use of setAttribute()         | Low        |
| Local storage disclosure               | Low        |

Overall, the application is found to be **very secure**. The several layers of security access controls along with secure configuration and the limited paths to exploit vulnerabilities via client side would give even an attacker with a considerable amount of time and resources a difficult time to circumvent.



## 1.5 METHODOLOGY

Halborn engaged in a time-boxed manual security assessment against the target application. This assessment involved a deep scan using scanning tools to discover common vulnerabilities, as well as manual testing. Manual testing includes validation of all issues found from any tools used, as well as checks for problems not typically found by automated scanners such as authentication issues, encryption issues, deserialization issues, and business logic flaws. The methodology and items that were tested for included but were not limited to:

- Mapping Application Content and Functionality
- Client Side/Browser Based Control Auditing
- Application Logic Flaws
- Access Handling
- Authentication
  - Session Management
  - Input Handling
- Fuzzing of all input parameters
  - Tests for XSS and CSRF
  - Test for Injection (/JSON/HTML/Command)
  - Web server misconfiguration
- Technology stack specific vulnerabilities and Code Audit
- Known vulnerabilities in 3rd party / OSS dependencies.



# FINDINGS & TECH DETAILS





## AVA LABS WALLET - OPEN SOURCE DEPENDENCY SECURITY CHECKS

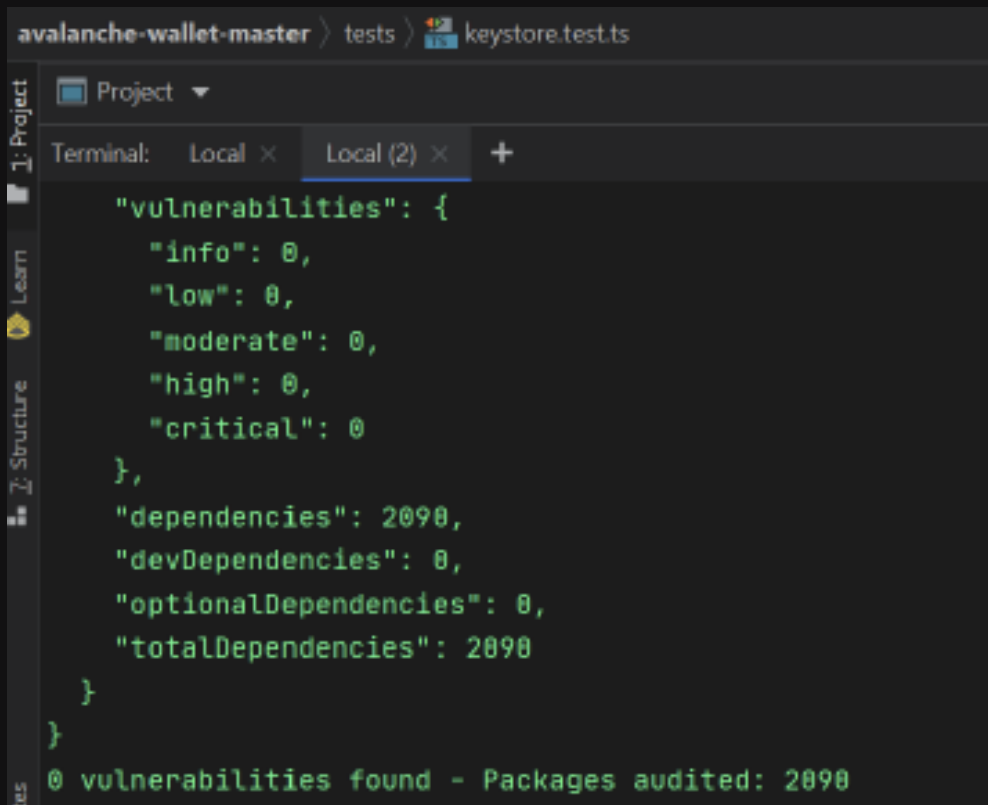
Dependency-Checking is an effort that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries.

The Avalanche Wallet, based on vue.js and yarn builds, has 2090 dependencies included in its web app build. Of all dependencies No vulnerabilities were identified.

However, there is a few risks in deprecated versions that can impact lifecycle management and maintainability. These are noted in Exhibit B, but are unrelated to security exploitation.

### Unused Dependencies - MEDIUM

There is one library found that is a potential vector for Cross-Site scripting, however, this library is not being used in the wallet, jspdf and is a non-referenced package. It is recommended that non-used dependencies in package.json be removed from future builds.

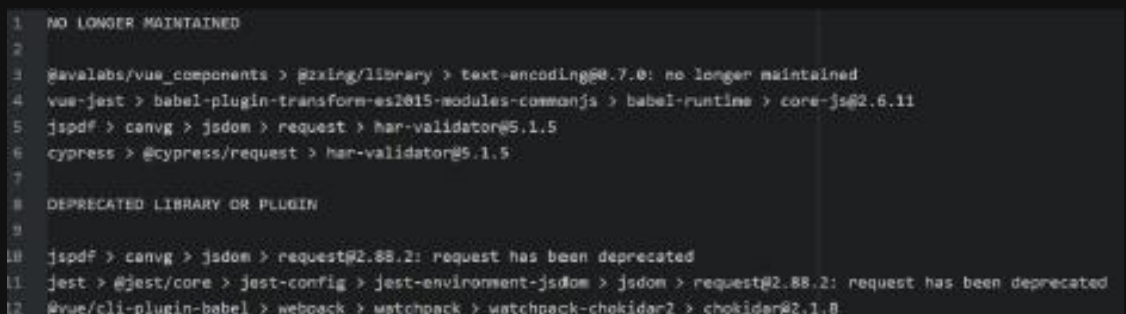


```
avalanche-wallet-master > tests > keystore.test.ts

Project
Terminal: Local x Local (2) x +

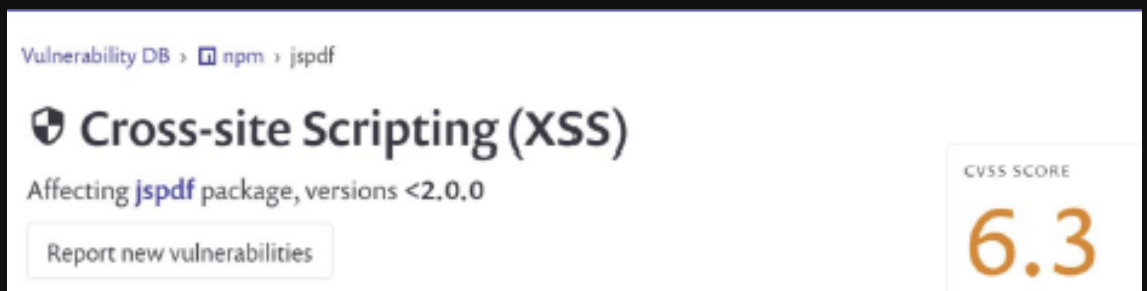
"vulnerabilities": {
  "info": 0,
  "low": 0,
  "moderate": 0,
  "high": 0,
  "critical": 0
},
"dependencies": 2090,
"devDependencies": 0,
"optionalDependencies": 0,
"totalDependencies": 2090
}
}
0 vulnerabilities found - Packages audited: 2090
```

Exhibit A: No vulnerabilities detected in 2090 packages.



```
1 NO LONGER MAINTAINED
2
3 @avalabs/vue_components > @zxing/library > text-encoding@0.7.0: no longer maintained
4 vue-jest > babel-plugin-transform-es2015-modules-commonjs > babel-runtime > core-js@2.6.11
5 jspdf > canvg > jsdom > request > har-validator@5.1.5
6 cypress > @cypress/request > har-validator@5.1.5
7
8 DEPRECATED LIBRARY OR PLUGIN
9
10 jspdf > canvg > jsdom > request@2.88.2: request has been deprecated
11 jest > @jest/core > jest-config > jest-environment-jsdom > jsdom > request@2.88.2: request has been deprecated
12 @vue/cli-plugin-babel > webpack > watchpack > watchpack-chokidar2 > chokidar@2.1.8
```

Exhibit B: Deprecated or unmaintained libraries



Vulnerability DB > npm > jspdf

## Cross-site Scripting (XSS)

Affecting **jspdf** package, versions <2.0.0

Report new vulnerabilities

CVSS SCORE  
**6.3**

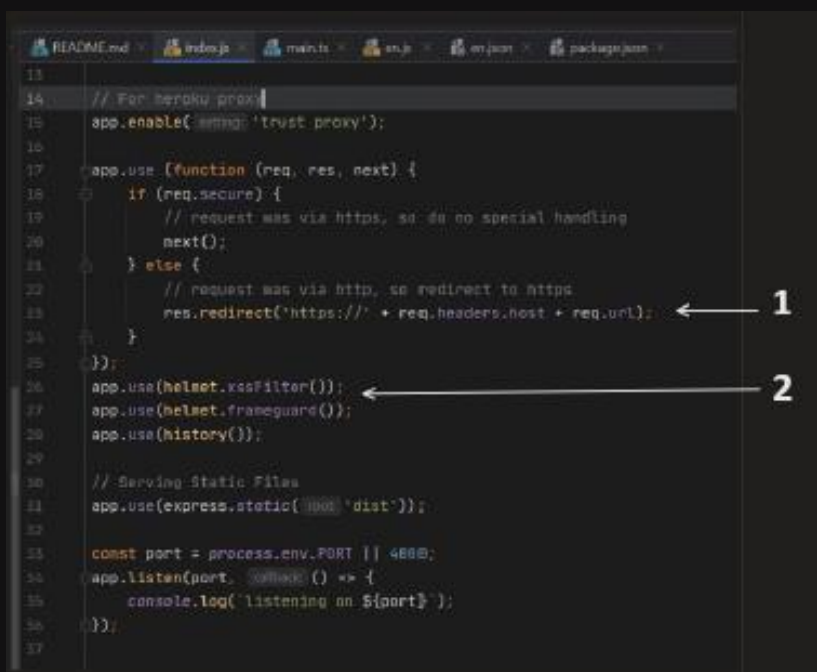
Exhibit C: Vulnerable version of js pdf can lead to XSS

# Ava Labs Wallet - Preventive Security Measures Implemented Into Wallet

The `X-XSS-PROTECTION` header was found to have a multitude of issues, instead of helping the developers protect their application. Following a decision by Google Chrome developers to disable Auditor, developers should be able to disable the auditor for older browsers and `set it to 0`.

`helmet.xssFilter` disables browsers' buggy cross-site scripting filter by setting the X-XSS-Protection header to 0

Also, `helmet.frameguard` sets the `X-Frame-Options` header to help you mitigate clickjacking attacks. This header is superseded by the frame ancestor's Content Security Policy directive but is still useful on old browsers. These are both enabled and protecting the wallet.



```
13
14 // For heroku proxy
15 app.enable('trust proxy');
16
17 app.use(function (req, res, next) {
18   if (req.secure) {
19     // request was via https, so do no special handling
20     next();
21   } else {
22     // request was via http, so redirect to https
23     res.redirect('https://' + req.headers.host + req.url); ← 1
24   }
25 });
26 app.use(helmet.xssFilter()); ← 2
27 app.use(helmet.frameguard());
28 app.use(history());
29
30 // Serving Static Files
31 app.use(express.static('dist'));
32
33 const port = process.env.PORT || 4880;
34 app.listen(port, () => {
35   console.log('listening on ${port}');
36 });
37
```

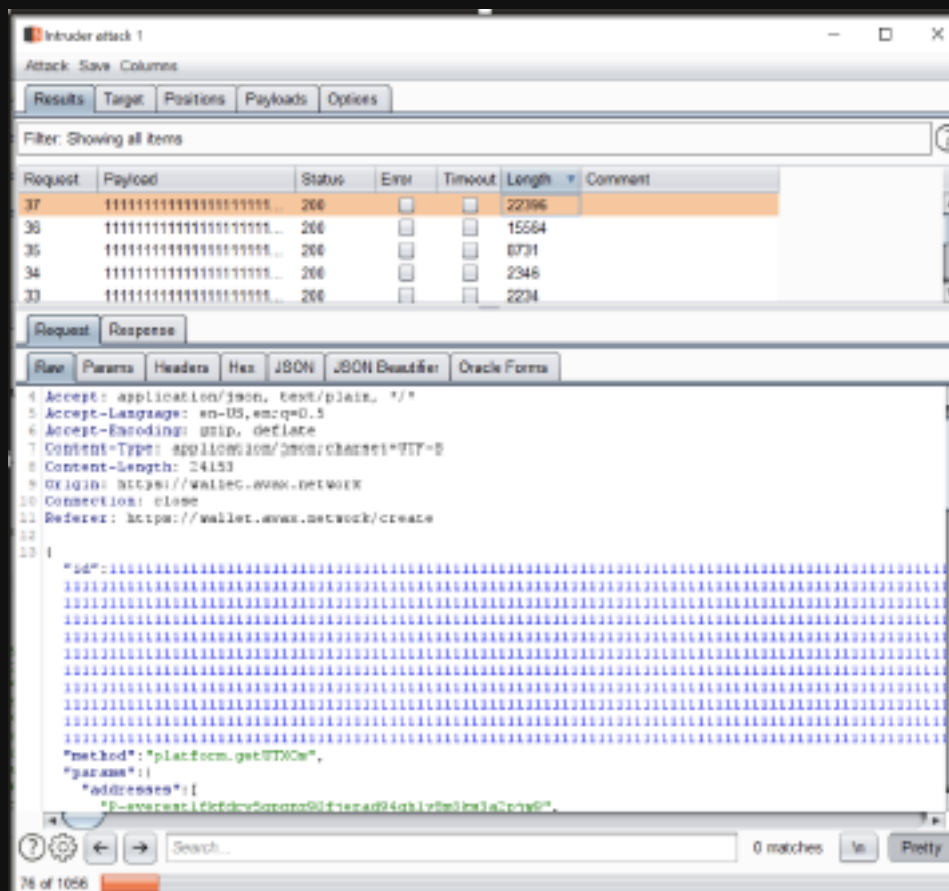
Exhibit D: Security protections for XSS and iFrame

## 1. Large number checks & Buffer Overflow/Bounds Checks:

A buffer overflow occurs when data written to a buffer also corrupts data values in memory addresses adjacent to the destination buffer due to insufficient bounds checking. This can occur when copying data from one buffer to another without first checking that the data fits within the destination buffer.

The security tester attempted multiple efforts to insert overflowing items into various parameters of the wallet, and the call to the network.

Result: No vulnerabilities, and error checks and unexpected values are handled.



## 2. Negative Integer Checks and Conditions

Result: No vulnerabilities, and error checks and unexpected values are handled.

Negative integer checks: Requests:

```
"P-everest12kv9h25g4wj1c737sefvkagh87ky40xw6cp8yc",
"P-everest1dn853t5emwvrxhj8xyykp4p03stkqaphgzc3y4",
"P-everest1vzrvekdarqxh67055rs18axm73sch6e776dt8a",
"P-everest1nxbh7ncwm0nh63kvn94tatk71jrt58rc498cld",
"P-everest1hd3rr83u0nv7seymfryfxvvjgc4cjwv2uexwsr",
"P-everest19cccs3u3hx8kxqk7w39npzhscws82hul323p80",
"P-everest1rxhjvpuhy5shet2ppq6agjpjy7hq7qnuv32gnq",
"P-everest1nh8yr77jxewva35xjfc22pgvfjht66p7p2sqddv",
"P-everest1s0npp93qflrskvkv9j0a7695yurf3na5x1wtc",
"P-everest1ugcjwahuqz33gg05tvly41d0226ceyvconq3598",
"P-everest13sqd0e2vpaxww32f6fghmsnp38uxm02cnvf6u5",
"P-everest172dsr37cxspacc3sm4qqej9hs2j334mtn13dpw",
"P-everest19e7cq2jppnh5tcqlf0qh9na0x81ylcn2x12wkd",
"P-everest1cg5u7r156rdt38hnf5ydqa0lhelaw5vtqqg90e",
"P-everest1cf49khwgt98mdycmh3zaaxsjd2s8x2c4yk0mz0",
"P-everest1jhe7duaexsd41z9v5xx4z915evz5ruvkhzrqgh",
"P-everest1ev8ewi2v69u8hv9hhsj48tcwmr1ezw4aeYvkvq",
"P-everest1ktq5n15hqjtlwu9rhl2uh5far81505kgw2nqj0",
"P-everest1q5xyvg94f0glcd7nudhunwf6try0u329y9aw55",
"P-everest1h48f0tccscuzq8kv4n7ka99r9s017cvt1ww52n",
"P-everest1q6xcgqf3g5aye0dmjxw6myq8p9455qf4gns46n"

1,
"limit":-1
},
"jsonrpc":"2.0"
```

Responses:

```
Set-Cookie: cfduid=deff8e1c593a171179c0b3845b42bc1600030356; expires=Tue, 13-Oct-28 20:52:16 GMT; path=/
Set-Cookie: AWSALBTG=264w1Hc3SDQLYi4HL2HCQ6zqB013UgUImhHw51Sj1CL8qNvbgzApp530EHQVUEX8BdvB+uNHVd8aBc72h/ogf
Set-Cookie: AWSALBTG=264w1Hc3SDQLYi4HL2HCQ6zqB013UgUImhHw51Sj1CL8qNvbgzApp530EHQVUEX8BdvB+uNHVd8aBc72h
Set-Cookie: AWSALB=CBpqp3p8Mfc1t3YDDtqSDqabxgk+dg4Ym3DQx1CP6VScTSQMAN1kdy2m11Cw5OG57TTeVj+9B3E5EePAQC7z
Set-Cookie: AWSALB=CBpqp3p8Mfc1t3YDDtqSDqabxgk+dg4Ym3DQx1CP6VScTSQMAN1kdy2m11Cw5OG57TTeVj+9B3E5EePAQC7z
Access-Control-Allow-Origin: *
Vary: Origin
CF-Cache-Status: DYNAMIC
cf-request-id: 062ad6cb420000ee6a94b44200000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
CF-RAY: 3d14c0bedb41ee6a-MIA
Content-Length: 164

{
  "jsonrpc": "2.0",
  "error": {
    "code": -32000,
    "message": "couldn't unmarshal an argument. Ensure arguments are valid and properly formatted. See document",
    "data": null
  },
  "id": 60605
}
```



# STATIC CODE SECURITY ANALYSIS FINDINGS



## Insecure Use of Document.Location - LOW

### Description:

Content being assigned to the `document.location` variable could be carrying tainted data. Since this data can be further used within the HyperText Markup Language (HTML) of a page, take steps to validate it. The Document Object Model (DOM) as defined by the World Wide Web Consortium (W3C) supports setting the location of the document, assuming it is well formed. Content passed into `document.location` should not be exposed to the user in a way that allows for alteration of that content. Exposing such content to the user can lead to a variety of Cross Site Scripting and HTML Injection attacks.

### Code Locations:

`src\public\serviceworker\redirect.html` Line 260



## Insecure Use of Document.Location - LOW

### Remediation:

In the cases where `document.location` carries input originating from an external location, further validation on the input is recommended. Validation of such content when it is exposed to the user should be performed in the same manner as server-side input validation. Input validation should not be solely performed by client-side JavaScript components exposed to the user. Validation implemented in this manner can be bypassed by means of manipulating the browser and/or the submitted request. Use Server-Side validation or a combination of Client-Side and Server-Side validation. Furthermore, any unnecessary content should not be attached to `document location`. Exposing such content to the user can lead to a variety of Cross Site Scripting and HTML Injection attacks.

## Insecure Use of setAttribute() - LOW

### Description:

Content being passed into the `setAttribute()` methods should be checked for containing tainted data. Since this data is used to add or modify attributes of elements inside the HyperText Markup Language (HTML) of a page, steps should be taken towards validating it. Content inputted to these methods should not be trusted without further validation. Using such content without validation can lead to a variety of Cross Site Scripting and HTML Injection attacks.

### Code Locations:

\src\store\index.ts:474 and 475

```
461         let utcDate = new Date()
462         let dateString = utcDate.toISOString().replace(' ', '_');
463         let filename = `AVAX_${dateString}.json`;
464
465         var blob = new Blob(
466             [ text ],
467             {
468                 type : "application/json"
469             }
470         );
471         let url = URL.createObjectURL( blob );
472         var element = document.createElement('a');
473
474         element.setAttribute('href', url);
475         element.setAttribute('download', filename);
476         element.style.display = 'none';
477         document.body.appendChild(element);
478         element.click();
479         document.body.removeChild(element);
480     },
481 }
```

## Insecure Use of setAttribute() - LOW

### Mitigation:

In the cases where this method carries input originating from an external location, further validation on the input is recommended. Validation of such content when it is exposed to the user should be performed in the same manner as server-side input validation. Input validation should not be solely performed by client-side JavaScript components exposed to the user. Validation implemented in this manner can be bypassed by means of manipulating the browser and/or the submitted request. Use Server-Side validation or a combination of Client-Side and Server-Side validation. Furthermore, any unnecessary content should not be passed to this method. Exposing such content to the user can lead to a variety of Cross Site Scripting and HTML Injection attacks.

## Local Storage Disclosure - LOW

### Description:

LocalStorage injection through the snippets functionality makes DOM (document object model) XSS possible. This can cause malicious user from replacing or using sensitive data in sessionStorage or localStorage with malicious code.

### Code Locations:

\src\src\store\index.ts Line 413

```
487
488
489     let wallets = state.wallets;
490
491     let file = await makeKeyfile(wallets, pass);
492     let fileString = JSON.stringify(file);
493     localStorage.setItem('w', fileString);
494
```

### Mitigation:

Always validate, encode, and escape user input before placing into localStorage or sessionStorage. Always validate, encode, and escape data read from localStorage or sessionStorage before writing onto the page (DOM). Always treat all data read from localStorage or sessionStorage as untrusted user input. In other words, any authentication your application requires can be bypassed by a user with local privileges to the machine on which the data is stored. Therefore, it's recommended not to store any sensitive information in local storage.



THANK YOU FOR CHOOSING

 **HALBORN**

