Least Authority
PRIVACY MATTERS

Avalanche Browser Extension + SDK

Security Audit Report

Ava Labs

Final Audit Report: 15 December 2023

Table of Contents

Overview

Background

Ava Labs has requested that Least Authority perform a security audit of their SDK and Avalanche Browser Extension Wallet – a non-custodial wallet for storing, sending, receiving, and swapping Avalanche assets.

Project Dates

- **August 8, 2023 - September 22, 2023:** Initial Code Review *(Completed)*
- **September 26, 2023:** Delivery of Initial Audit Report *(Completed)*
- **15 December, 2023:** Verification Review *(Completed)*
- **15 December, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Nicole Ernst, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Avalanche Browser Extension Wallet and SDK followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Avalanche Browser Extension Wallet: https://github.com/ava-labs/extension-avalanche
- Avalanche Wallet SDK: https://github.com/ava-labs/avalanche-sdks/tree/alpha-release/packages/wallets-sdk

Specifically, we examined the Git revisions for our initial review:

- Avalanche Browser Extension Wallet: `cd49a47ea4453295e0e23205ce406fbb341d05b5`
- Avalanche Wallet SDK: `aacd8ed1753a23accc75eef986ae6355a6a04234`

For the verification, we examined the Git revision:

- `a288e2eeb43ea69592bcd596e355d3c013df65b6`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Avalanche Browser Extension Wallet: https://github.com/LeastAuthority/ava-labs-avalanche-browser-extension-audit
- Avalanche Wallet SDK: https://github.com/LeastAuthority/ava-labs-avalanche-browser-extension-sdk-audit/tree/alpha-release

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Avalanche Browser Extension Wallet `README.md`:
  https://github.com/ava-labs/extension-avalanche
- Avalanche Wallet SDK `README.md`:
  https://github.com/ava-labs/avalanche-sdks/wallets-sdk

In addition, this audit report references the following documents:

- P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, et al., "Digital Identity Guidelines: Authentication and lifecycle management." *NIST Special Publication 800-63B*, 2017, [GFN+17]
- The `scrypt` parameters:
  https://words.filippo.io/the-scrypt-parameters
- MDN documentation:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random
- Windows opened via <a target=_blank> should not have an opener by default:
  https://github.com/whatwg/html/issues/4078
- `dropbox/zxcvbn`:
  https://github.com/dropbox/zxcvbn
- RFC 9106:
  https://datatracker.ietf.org/doc/html/rfc9106#name-parameter-choice

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and adherence to best practices;
- Common and case-specific implementation errors;
- Exposure of any critical information during user interactions with the blockchain and external libraries, including authentication mechanisms;
- Adversarial actions and other attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Vulnerabilities in the code and whether the interaction between the related and network components is secure;
- Proper management of encryption and storage of private keys, including the key derivation process;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team conducted a comprehensive review of the Avalanche browser wallet extension, which allows its users to interact with the Avalanche blockchain, its dApps, and the Avalanche Bridge that connects the Avalanche and Ethereum blockchains. In addition to conducting a full audit of the current design and implementation of the browser extension, we also examined the implementation of the Avalanche wallet SDK, a toolkit designed to streamline the development of wallets within the Avalanche ecosystem. As this is the third review of the Browser extension by our team, we also checked if the security issues and suggestions that were identified in the previous review have been addressed in the current implementation.

System Design

Overall, our team found that the Ava Labs team has taken security into consideration in the design and development of the Avalanche Browser Extension Wallet and the Wallet SDK. We examined the locking mechanism, in addition to reviewing the processing of secret keys in the Wallet SDK and could not identify any issues.

Our team also examined proper management of encryption and identified several areas of improvement. Although these Issues are very local and would not affect the overall design, we found that functions and parameters used for encryption can be improved to add an extra layer of security (Issue A, Issue B).

Additionally, we found that the function generating a random sampling of mnemonics can produce an invalid index, which can lead to users being unable to complete the onboarding process (Issue C).

Furthermore, the wallet uses the `window.open` method for redirecting users to external pages. However, the `noopener` parameter, which adds a layer of security to the `window.open` method by preventing newly opened windows from having unauthorized access to the object of the parent window, is not utilized. We recommend explicitly setting the `noopener` attribute whenever the `window.open` method is utilized (Issue D).

Finally, our team found that several Issues that were reported in previous audits remain unresolved. More specifically, our team found that the recovery phrase is still stored in clipboard, the user still does not have control over a wallet's session management, and low entropy passwords continue to be allowed upon new wallet creation. We acknowledge that unresolved Issues involve a potential trade-off between security and ease of use. However, we believe in prioritizing security and transparently communicating any associated risks to users when mitigation is challenging or not possible. Thus, our team is reporting these Issues again (Issue E, Issue F, Issue G).

Code Quality

We performed a manual review of the repositories in scope and found the codebases to be organized and well-written. Additionally, the code is easy to read and reason about.

Tests

The Ava Labs team has implemented sufficient unit tests, which showed high coverage of important logic.

Documentation

The project documentation provided for this review provides a generally sufficient overview of the system and its intended behavior.

Code Comments

We found that code comments sufficiently describe the intended behavior of security-critical components and functions.

Scope

The scope of this review was sufficient and included all security-critical components.

Dependencies

Our team did not identify any security concerns resulting from the unsafe use of dependencies. However, we found that dependencies have occasionally been updated and recommend that the Ava Labs team continue to regularly review and assess currently used dependencies.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Insufficient Argument Validation in Encryption Functions	Resolved
Issue B: Insufficient Key Derivation Strength	Resolved
Issue C: Mnemonic Sampling May Fail	Resolved
Issue D: Insecure Calls to window.open	Resolved
Issue E: Recovery Phrase Is Stored in Clipboard	Unresolved
Issue F: No User Controls Over Wallet’s Session Management	Unresolved
Issue G: Low Entropy Passwords Are Allowed Upon New Wallet Creation	Partially Resolved

Issue A: Insufficient Argument Validation in Encryption Functions

Location

`storage/utils/crypto.ts`

Synopsis

The function takes a boolean flag `derive`, a key, and a value. The key may be a string or a `Uint8Array`. If `derive` is set or the key does not have a length of 32, the `scrypt` password hashing function is used to derive a strong key.

It can be assumed that the caller only expects the function to derive a key when `derive` is set. This means that the function is too liberal in its inputs: If the caller mistakenly assumes that they pass a 32-byte `Uint8Array`, but accidentally pass, for example, an uninitialized array, and set `derive` to `false`, the function will use an encryption key derived from an empty array.

Impact

This Issue could result in keys being stolen, or the loss of funds.

Preconditions

An attacker would need access to encrypted data, for example, by stealing the browser profile.

Feasibility

Our team did not find a way to exploit the attack, given how the function is used in the rest of the codebase.

Remediation

The Issue occurs as a result of:

1. Being liberal in the inputs in the face of conflicting inputs; and
2. One function, which is controlled by a Boolean flag argument, being used to perform two separate tasks.

The first can be resolved by returning an error when `derive == false` and `key.length != 32`. However, this approach results in code that is disorganized and difficult to read. Instead, we recommend splitting the function into two parts: One function `encryptWithKey` that takes a `Uint8Array`, which is checked to have a length of 32, and another function `encryptWithPassword` that takes a string, which is used to derive a key and then calls `encryptWithKey`. The caller can then call the appropriate function depending on the context.

Status

The Ava Labs team has implemented the remediation as recommended.

Verification

Resolved.

Issue B: Insufficient Key Derivation Strength

Location

`storage/utils/crypto.ts`

Synopsis

The browser extension uses the `scrypt` password hashing function with parameters `N = 1<<15`, `r = 8`, `p = 1`. This matches the recommendations for interactive logins, but since the situation requires file encryption, this is insufficient.

Impact

This Issue could result in keys being stolen, or the loss of funds.

Preconditions

An attacker would need access to encrypted data, for example, by stealing the browser profile.

Feasibility

This type of exploit requires significant but realistic computation power.

Remediation

We recommend either increasing N to `1<<20` or using `Argon2id` with parameters t=3, p=4, `m=64MiB`, and 16-byte salt, as recommended in [section 4 of RFC 9106](#).

Status

The Ava Labs team has resolved this Issue by implementing a parallelism factor parameter of 1, which does not affect security.

Verification

Resolved.

Issue C: Mnemonic Sampling May Fail

Location

[`Onboarding/CreateWallet/ConfirmMnemonic.tsx#L34`](#)

[`Onboarding/CreateWallet/ConfirmMnemonic.tsx#L52`](#)

Synopsis

The code written for randomly sampling mnemonics may yield an array index out of range error.

Impact

This Issue could result in users being unable to complete the onboarding process.

Technical Details

The function `Math.random` generates a random decimal number between 0 (inclusive) and 1 (exclusive), as per the [MDN documentation](#):

```
Math.ceil(Math.random() * wordCount - 1);
```

Consequently, the aforementioned function can potentially produce an invalid index when `Math.random` returns zero or a value very close to zero. This is because subtracting 1 from a small decimal number can result in a -1, and `Math.ceil` will round this up to -1, thus leading to an incorrect array index.

Remediation

We recommend using `Math.floor` to round the randomly generated number as follows:

```
Math.floor(Math.random() * wordCount);
```

In this corrected code, `Math.random() * wordCount` produces a floating-point number in the range `[0, wordCount)`. Then, `Math.floor` rounds this number down to the nearest whole number, resulting in an integer in the range `[0, wordCount - 1]`, which is the correct range of valid indices for an array of length `wordCount`.

Status

The Ava Labs team has resolved this Issue by implementing the remediation as recommended.

Verification

Resolved.

Issue D: Insecure Calls to window.open

Location

Examples (non-exhaustive):

`pages/Wallet/WalletRecentTxs.tsx#L352`

`components/common/CameraAccessPromptDialog.tsx#L55`

`settings/pages/Legal.tsx#L46`

`pages/Bridge/BridgeTransactionStatus.tsx#L267`

Synopsis

The code uses the `window.open` method in multiple places to link and redirect users to external pages without explicitly setting the `noopener` attribute. The `noopener` attribute, when used with the `window.open` method, serves as a security feature to prevent newly opened windows or tabs from accessing the `window.opener` object of the parent window. This feature is crucial in safeguarding against potential security vulnerabilities, particularly cross-site scripting (XSS) attacks.

Impact

If an attacker manages to exploit this omission, they can gain access to the `window.opener` object of the parent window from a child window or tab. This access allows malicious scripts running in the child window to manipulate or interact with the parent window, potentially leading to unauthorized actions, data theft, or other security breaches.

Preconditions

This Issue can be exploited if:

1. A new window or tab is opened using the `window.open` method, establishing a parent-child relationship between them;
2. The `noopener` attribute is not included as part of the `window.open` method when opening the new window or tab;
3. The attacker has the ability to execute malicious scripts within the child window or tab; and
4. Either the use case has an explicit `noopener` attribute explicitly set or users use browser versions that do not implement this spec change.

Remediation

We recommend reviewing the existing usages of the `window.open` method across the codebase and always using the `noopener` attribute explicitly when utilizing the `window.open` method to open new windows or tabs by including the `noopener` attribute as an argument. This attribute prevents the child window or tab from accessing the `window.opener` object of the parent window.

Status

The Ava Labs team has resolved this Issue by adding the `noreferrer` flag, which is even more restrictive than `noopener`, to all instances of `window.open`.

Verification

Resolved.

Issue E: Recovery Phrase Is Stored in Clipboard

Location

`Onboarding/CreateWallet/CopyPhrase.tsx`

Synopsis

The mnemonic phrase used to derive the Avalanche browser extension wallet keys is accessible to all applications that have access to Clipboard. Clipboard is a global object that is accessible across application security boundaries. Applications watching Clipboard will be able to see the mnemonic when the user copies it to Clipboard.

Impact

This Issue could result in the loss of funds or a complete wallet takeover.

Preconditions

An attacker would need to infect the victim's PC with a Clipboard hijacker malware or any other malicious Clipboard monitoring process. This can also occur if a user installs a malicious browser extension with the required permissions, or in the event that the browser was previously compromised and an extension was installed without the user's knowledge.

Technical Details

The mnemonic phrase may be copied to Clipboard upon finishing the registration process. The Clipboard is considered a public medium, which can be accessed by almost any program or process running on the user's device, without any prior permission or notification.

Remediation

We recommend preventing the mnemonic phrase from being saved to Clipboard. Instead, we recommend requiring users to write down the mnemonic phrase in a place off of the filesystem of their device and continuing to require the user to confirm the mnemonic phrase, which helps to confirm that the user has actually written it down.

Note that this Issue was reported in our previous review, and the Ava Labs team decided not to address it, stating that they want to continue providing a copy button for a smoother user experience. Hence, we are reporting it again.

Status

The Ava Labs team decided to not resolve this Issue, as they found it would have a negative impact on user experience.

Verification

Unresolved.

Issue F: No User Controls Over Wallet's Session Management

Location

`services/lock/LockService.ts#L81`

services/lock/models.ts#L7

Synopsis

The Avalanche browser extension wallet implements an auto-locking mechanism, which is hard coded to a 12-hour interval. As a result, the wallet remains unlocked for an extended period of time unless a user manually activates the locking mechanism or closes the browser running the extension. Users have no control over auto-locking intervals. A user who leaves their wallet-hosting device unattended or gets infected with a Remote Access Trojan (RAT) is vulnerable to losing funds. An unlocked wallet gives an attacker an additional advantage, particularly because signing a transaction or removing the wallet account does not require password authentication if the wallet is unlocked. This is considered an insecure practice, as users have a propensity to forget locking their wallets.

Impact

Once a user has logged into their wallet by providing a password, the password is stored in Chrome's session storage with a key named `SESSION_AUTH_DATA_KEY`.

For example, the plaintext value of the password can be extracted through Chrome's Developer Tools as follows:

```
chrome.storage.session.get("SESSION_AUTH_DATA_KEY", result=>
console.log(result))
```

Persisting the wallet access sessions unnecessarily for an extended period of time increases the attack window in which an attacker can take advantage of the system.

Remediation

We recommend that the Ava Labs team implement a user-controllable auto-lock feature and activate it by default for all users. Additionally, we recommend adding a warning of possible hazards to users who try to disable it. Furthermore, we recommend that users be required to authenticate their identity by inputting the password to sign transactions and remove a wallet account.

Note that this Issue was reported in our previous review, and the Ava Labs team decided not to address it, stating that implementing the auto-lock feature can have implications for user experience with regards to dApp connections. Hence, we are reporting it again.

Status

The Ava Labs team decided to not resolve this Issue, as they found it would have a negative impact on user experience.

Verification

Unresolved.

Issue G: Low Entropy Passwords Are Allowed Upon New Wallet Creation

Location

pages/Onboarding/CreatePassword.tsx#L51

Synopsis

In the current implementation, the user password strength checking mechanism is only utilized for password changes and not during the initial onboarding process. As a result, a user can select an insufficiently secure password during the initial onboarding to the Avalanche browser extension wallet.

Impact

Weak passwords leave the wallet susceptible to dictionary attacks. An attacker who is able to guess the user password can take complete control of the wallet.

Preconditions

The attacker would need to have access to either the encrypted mnemonic or the extension.

Remediation

The strength check that is currently performed when the user changes their password should also be performed during the initial onboarding to the Avalanche browser extension wallet. We recommend using the dropbox/zxcvbn library for a password composition policy intended to prevent the use of passwords obtained from previous breaches, common passwords, and passwords containing repetitive or sequential characters, as recommended by the NIST guidelines noted in [GFN+17]. Finally, we recommend requiring that all passwords meet `zxcvbn`'s strength rating of 4.

Note that this Issue was reported in our previous review, and the Ava Labs team decided not to address it, as they considered it to be a low threat vector. Hence, we are reporting it again.

Status

The Ava Labs team has implemented the `zxcvbn` library as recommended; however, users are required to choose a password strength of at least 2, which is below the "[safely unguessable] standard of 3 .

Verification

Partially Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.