Least Authority

PRIVACY MATTERS

Taho Wallet: Private Key Import + Recovery Phrase Reveal

Security Audit Report

Thesis

Final Audit Report: 7 July 2023

Table of Contents

Overview

Background

Thesis has requested that Least Authority perform a security audit of their Taho Wallet Extension.

Project Dates

- **May 11 - June 2, 2023:** Initial Code Review *(Completed)*
- **June 6, 2023:** Delivery of Initial Audit Report *(Completed)*
- **July 6 2023:** Verification Review *(Completed)*
- **July 7, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Nicole Ernst, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Taho Wallet: Private Key Import + Recovery Phrase Reveal 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:
- `Keyring-with-pk`:
    - https://github.com/tahowallet/extension/blob/keyring-with-pk/background/
    - https://github.com/tahowallet/extension/tree/keyring-with-pk/ui/
    - https://github.com/tahowallet/extension/pull/3089/files
    - https://github.com/tahowallet/extension/pull/3119
- `Hd-Keyring`:
    - https://github.com/tahowallet/hd-keyring/pull/13

Code segments relative to the execution paths related to the list of directories deemed in scope were also analyzed in order to achieve comprehensive understanding and coverage.

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

- `Keyring-with-pk`: `98ea9e7ead335ae6bc33527d815f985f3ad5c063`
- `Hd-Keyring`: `c5175d50793643905742f7947958385fb20445e7`

For the verification, we examined the Git revisions:

- `4fc5d6ab5e66e0d26d10a333a7e4a4e2926cf93b`
- `af9527aab99b50ed6395c94a47e904b35a0d5ce2`
- `2c1816b6515d5b16615f9e9339e9a146151aafb7`

For the review, this repository was cloned for use during the audit and for reference in this report:

- Taho Wallet Extension:
  https://github.com/LeastAuthority/Taho-wallet-extension

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:

- Recovery Phrase Reveal - design and files impacted:
  https://gist.github.com/0xDaedalus/f3d25a6d64c90c2410fa8ec330f5c3e7
- Design document for the current iteration:
  https://github.com/tahowallet/extension/blob/98ea9e7ead335ae6bc33527d815f985f3ad5c063/rfb/rfb-4-one-off-keyring-design.adoc
- Milestones:
  https://github.com/tahowallet/extension/milestone/29
- README:
  https://github.com/tahowallet/hd-keyring/blob/main/README.md
- Website:
  https://taho.xyz/

In addition, this audit report references the following documents and links:

- A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications." *IRTF*, 2021, [BDK+21]
- D. L. Wheeler, "`zxcvbn`: Low-Budget Password Strength Estimation." *USENIX*, 2016, [Wheeler16]
- Project `libpwquality`:
  https://github.com/libpwquality/libpwquality
- Embarrassingly parallel:
  https://en.wikipedia.org/wiki/Embarrassingly_parallel
- `argon2-browser` package:
  https://github.com/antelle/argon2-browser
- Validator.js:
  https://github.com/validatorjs/validator.js
- JSDoc:
  https://jsdoc.app

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the wallet;
- Attacks that impacts funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Malicious attacks and security exploits that would impact the wallet;
- Vulnerabilities in the wallet code and whether the interaction between the related and network components is secure;
- Exposure of any critical or sensitive information during user interactions with the wallet, and use of external libraries and dependencies;
- Proper management of encryption and storage of private keys;
- 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 performed a security audit of the Taho Wallet Extension. The specific focus of this audit was the user mnemonic and secret key generation, handling, and storage within the implementation. Our team investigated the process for new user onboarding and the importing of a wallet using a mnemonic phrase and the validations performed on them. In addition, our team examined the `internal.signer` functions, which enable the user to sign using the private key.

We examined the wallet's communications with external APIs and did not identify any security vulnerabilities. Furthermore, our team reviewed the encryption of data and parameters used, and generated logs to check for leaks of sensitive data. The logs collect metrics in a collection pool. We reviewed these logs and did not find any personally identifying information (PII) or sensitive data logged.

We found the components to be generally well-designed and implemented. Our team reviewed the Taho wallet key handling previously. The team from this current review noted that issues identified in the previous audit, which can make users vulnerable to critical vulnerabilities, continue to be unresolved. We recommend that the use of cryptography be improved and that security be prioritized over user experience considerations in implementing a wallet.

System Design

Our team found that the wallet does not enforce any constraints for user-defined passwords in the onboarding process. Weak passwords can be easily guessed by dictionary attacks, which can lead to a complete takeover of the user's wallet. We recommend that constraints for password strength be implemented to prevent the selection of insufficiently secure passwords (Issue A). In addition, we found that the wallet extension currently does not allow users to change their passwords (Issue C).

We examined the key derivation function (KDF) used for encrypting the mnemonic, and found that the wallet utilizes the key derivation algorithm PBKDF2, which is CPU-bound and therefore can be efficiently parallelized. We recommend the use of KDF based on memory-hard functions such as `Argon2` (Issue D). Furthermore, the mnemonic phrase used to derive private keys can be copied to the clipboard and is hence accessible to all applications maliciously monitoring the clipboard. We recommend preventing seed phrases from being saved to the clipboard (Issue B).

In the current implementation, the wallet will automatically lock itself after a period of 60 minutes. Although this functionality reduces the chances of unauthorized access to the unlocked wallet, the fixed time interval may not be suitable for all users. In an unlocked state, an attacker can execute transactions or remove the wallet without authentication. We recommend giving users the ability to control the auto-lock interval (Suggestion 2).

Code Quality

Our team performed a manual review of the in-scope implementation, and found the code to be well-organized and adhering to best practices.

Tests
The Taho Wallet Extension has an extensive test suite, including unit and end-to-end UI tests that cover most user scenarios.

Documentation
The project documentation provided for this review was generally sufficient, describing the general architecture and the components that compose the system.

Code Comments
In our review, we found some code comments explaining the intended behavior of functions and components. We recommend that code comments be improved to comprehensively describe the intended behavior of all security-critical functions and components (Suggestion 3).

Scope
The scope of this security review was not clearly defined from the beginning of the audit. As a result, our team had to ask for clarification during the review regarding what parts of the implementation were in scope. Once clarified, our team was able to comprehensively review the in-scope code. However, we recommend that a comprehensive audit of the entire implementation be performed.

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: Weak Passwords Allowed	Unresolved
Issue B: Seed Phrase and Private Key Can Be Copied to Clipboard	Partially Resolved
Issue C: No Option To Change User Passwords	Unresolved
Issue D: Insufficiently Secure Key Derivation Algorithm Used	Resolved
Suggestion 1: Suppress Backtracking To Prevent Potential ReDoS Attack	Unresolved
Suggestion 2: Implement a User-Controllable Auto-Lock Feature With a Shorter Default Time	Resolved
Suggestion 3: Improve Code Comments	Unresolved

Issue A: Weak Passwords Allowed

Location
`components/InternalSigner/InternalSignerSetPassword.tsx#L34`

`Onboarding/Tabbed/SetPassword.tsx#L41`

components/Password/PasswordStrengthBar.tsx

Synopsis

In the current implementation, although the password should be at least 8 characters long, there are no restrictions on how weak a password can be. This could result in a user selecting an insufficiently secure password, which would enable an attacker with access to the target's machine to decrypt secrets or unlock the wallet.

Impact

A decrypted seed phrase gives full control over the wallet. Similarly, an attacker with access to the browser of the target can unlock the wallet and gain full access.

Preconditions

This Issue is likely to occur if the user selects an insufficiently secure password, and the attacker is either in possession of the encrypted secrets or in control of the target's browser.

Feasibility

Attackers with encrypted secrets can easily guess the password using a brute-force algorithm.

Remediation

We recommend using a library such as `zxcvbn`, as noted in [Wheeler16], or `libpwquality` to estimate the strength of user-selected passwords, and requiring passwords to have the maximum strength of 4.

Status

The Taho team stated that they do not consider it appropriate to dictate minimum strength to users without a better understanding of their intended use of the wallet. However, they are considering adding minimum strength suggestions based on asset value for an unscheduled future release.

Verification

Unresolved.

Issue B: Seed Phrase and Private Key Can Be Copied to Clipboard

Location

components/AccountsBackup/RevealMnemonic.tsx#L93

components/AccountsBackup/RevealPrivateKey.tsx#L44

Synopsis

The wallet currently features a button that allows users to copy the seed phrase to the system clipboard. Since all processes running on the system can read the clipboard, this presents a security risk when malicious programs are present on the system.

Impact

The seed phrase allows the attacker control over the wallet, which can result in a complete loss of funds.

Preconditions

The attacker needs to have a malicious program or browser extension running on the target's system.

Feasibility
The attack itself is trivial. For example, scripts that scan for seed phrases are available online, and restoring a wallet from a seed phrase is possible using the Namada Interface Extension.

Remediation
We recommend removing the function from the extension and warning users against screenshotting the seed phrase.

Status
The Taho team has added a warning to alert users about the risks of copying their secrets to the clipboard.

Verification
Partially Resolved.

Issue C: No Option To Change User Passwords

Synopsis
Once a user selects a password when creating an account, they are indefinitely unable to change their password. This presents an issue when a user's password gets compromised.

Impact
See Issue A.

Preconditions
As in Issue A, the attacker would need access to the target's browser or to have their encrypted seed phrase in their possession. Additionally, in this case, they would also need to know the target's compromised password.

Feasibility
If the preconditions are met, the attack is trivial.

Remediation
We recommend adding an option allowing users to change their password, and re-encrypt the secrets, using the new password.

Status
The Taho development team has acknowledged this suggestion and noted that, while the recommended mitigation will not be implemented at this time, it will be taken into consideration for future releases.

Verification
Unresolved.

Issue D: Insufficiently Secure Key Derivation Algorithm Used

Location
`extension/background/services/internal-signer/encryption.ts`

Synopsis

The Taho Wallet Extension currently makes use of PBKDF2, which is a purely CPU-bound key derivation function. This class of algorithms has been considered insufficiently secure for several years because it is embarrassingly parallelizable.

Impact

The limitations of PBKDF2 may result in leakage of vault contents (i.e. mnemonics and secret keys), which can then be compromised by an attacker and result in the loss of user funds.

Preconditions

The attack requires access to the encrypted vaults, and this could happen through two vectors:

1. a malicious website extension could circumvent browser security measures; and
2. a regular program running with user permissions can usually access the browser profile and, thus, all extension data.

Feasibility

The feasibility of the attack depends on the strength of the password. `Argon2` provides security benefits particularly for weak passwords, which could realistically be guessed by brute force.

Technical Details

PBKDF2 is a function that derives keys from passwords, and the function iteratively calls a normal hash function (in this case SHA256). Therefore, it has a tight loop and only requires a small amount of memory. This can be efficiently parallelized and even implemented on a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC) with relatively minimal effort.

The newer class of memory-hard hash functions avoids this problem by accessing a large amount of memory, which is always cost prohibitive to implement in hardware.

Remediation

We recommend using the memory-hard `Argon2id` function instead of the currently implemented PBKDF2. In the `Argon2` RFC, as noted in [BDK+21], guidance is provided for the choice of parameters. We suggest selecting `t=3` iterations, `p=4` lanes, `m=2^(16)` (64 MiB of RAM), 128-bit salt, and 256-bit tag size (i.e. the second recommended option).

In addition, we suggest performing the computation in WebAssembly (e.g. using the `argon2-browser` package).

Status

The Taho team has migrated to using the `argon2-browser` package to encrypt secret vaults and has also implemented functionality to re-encrypt existing vaults upon the first time that the respective user unlocks their vaults.

Verification

Resolved.

Suggestions

Suggestion 1: Suppress Backtracking To Prevent Potential ReDoS Attack

Location

`background/lib/fixed-point.ts#L144`

`background/lib/fixed-point.ts#L206`

`background/lib/logger.ts#L86`

Synopsis

The libraries' regular expressions use nested or in-sequence quantifiers. If a string is passed that is sufficiently large enough, and that partially matches the regular expression but is not accepted, it would fail causing catastrophic backtracking by recalculating all the possible combinations. A sufficiently lengthy string that has a very large amount of combinations to be checked against could cause a denial of service.

Mitigation

While there are no instances of these libraries being used unsafely, we recommend using `validator.js` for input validation. Otherwise, we recommend suppressing backtracking by implementing atomic groups.

Status

The Taho development team has acknowledged this suggestion and noted that, while the recommended mitigation will not be implemented at this time, it will be taken into consideration for future releases.

Verification

Unresolved.

Suggestion 2: Implement a User-Controllable Auto-Lock Feature With a Shorter Default Time

Synopsis

The Taho Wallet Extension currently implements a timeout feature that is set for a period of 30/60 minutes. This provides a baseline level of security in case a user is separated from an unlocked device.

Mitigation

We recommend making the timeout length configurable by the user, and setting the default to a period shorter than ~five minutes, after which the user would have to authenticate.

Status

The Taho team has implemented a function allowing users to set a custom auto-lock timer, with a default time of one hour.

Verification

Resolved.

Suggestion 3: Improve Code Comments

Synopsis

There are currently very few comments throughout the codebase, and the comments that are present repeat the function and method names. The documentation contained within the code should be comprehensive and document every function and entry point, in addition to explaining the intended functionality of each of the components. This allows both maintainers and reviewers of the codebase to comprehensively understand the intended functionality of the implementation and system design, which increases the likelihood for identifying potential errors that may lead to security vulnerabilities.

Mitigation

We recommend creating code comments that explain each variable, function, and entry point.

Status

The Taho team stated that they will continue improving code comments in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

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.