Least Authority
PRIVACY MATTERS

Tally Browser Extension Wallet: Key Handling
Security Audit Report

YLVIS, LLC

Final Audit Report: 18 March 2022

Table of Contents

Overview

Background

YLVIS, LLC has requested that Least Authority perform a security audit of the Tally browser extension wallet key handling code. Tally is a community owned and operated Web3 browser extension wallet.

Project Dates

- **November 1 - 19**: Code review (*Completed*)
- **November 24**: Delivery of Initial Audit Report (*Completed*)
- **March 16 - 17**: Verification Review (*Completed*)
- **March 18**: Delivery of Final Audit Report (*Completed*)

Review Team

- Ann-Christine Kycler, Security Researcher
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Tally browser extension wallet key handling code, followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

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

- `services/keyring`: https://github.com/tallycash/tally-extension/tree/main/background/services/keyring
- `hd-keyring`: https://github.com/tallycash/hd-keyring

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

`services/keyring`: `cc22b1bbdec37611511978c7f91373a4de7cc4fd`

`hd-keyring`: `5b042957f7ac670b7d5e90794f94bb17167f76e2`

For the verification, we examined the Git revision:

`services/keyring`: `a522d0d3b7f81e34f3827224c54481f997ad686e`

`hd-keyring`: `acd2879616d671afcf6efc75cc68ca57b08eb8ed`

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

`services/keyring`: https://github.com/LeastAuthority/tally-extension/tree/tally-key-handling-audit/background/services/keyring

`hd-keyring`: https://github.com/LeastAuthority/tally-hd-keyring/tree/tally-key-handling-audit

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

In addition, the `background/tests` repository, along with 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:

- `README.md`: https://github.com/LeastAuthority/thesis-tally-extension/blob/thesis-tally-key-handling-audit/README.md
- Keyring-design-doc.pdf (*shared with Least Authority via Slack on 1 November 2021*)
- Extension Architecture.pdf (*shared with Least Authority via Slack on 1 November 2021*)
- Extension and Contract Interaction.pdf (*shared with Least Authority via Slack on 1 November 2021*)
- Wallet Taxonomy.pdf (*shared with Least Authority via Slack on 1 November 2021*)

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Key management, encryption, and storage, including the key derivation process; and
- Anything else as identified as critical to these areas during the initial analysis phase.

This is a limited scope review and not intended to be a comprehensive audit of the Tally extension.

Findings

General Comments

Our team performed a security audit of the Tally browser extension wallet's key handling code, which is found primarily in the `hd-keyring` repository and the `services/keyring` component of the `tally-extension` repository. The `hd-keyring` repository implements hierarchic deterministic key derivation according to BIP-44. The `services/keyring` handles encryption and storage of keys, in addition to providing this functionality to other components of the Tally browser extension wallet. All other components that make up the browser extension wallet were considered out of scope for this security audit.

In this report, we provide several recommendations for improving the overall security of the in-scope components. However, we found that security has been considered in the system design and implementation of the Tally browser extension wallet's key handling code and did not identify critical security vulnerabilities.

System Design

The design of the key handling components of the Tally browser extension wallet demonstrate security considerations for the way in which access to secrets is managed. We investigated the keyring service and attempted to extract keys or insert wrong keyring data into the system and did not identify any issues with the implementation. Furthermore, we were unable to break any of the security expectations of the keyring service. However, we identified several instances where data consistency is not guaranteed in the codebase, but could not find ways to exploit these inconsistencies to create unintended or unexpected behavior.

Use of Cryptography

In investigating the use of cryptography in the implementation, we found that PBKDF2 is used as a key derivation function, which accepts a user selected password as an input. This practice is generally considered to be insufficiently secure due to vulnerability to enumeration attacks, leading to the potential leakage of the vault contents. We recommend using a memory-hard key derivation algorithm, such as Argon2id (Issue A).

In addition, data is encrypted using AES-GCM, which is configured to use a nonce parameter that could lead to a collision which would break the security of the encryption. As a result, we recommend configuring the nonce parameters to protect against collisions (Suggestion 6).

Vaults

The keyring service handles encryption and keys, and its vaults are encrypted containers that hold the secret wallet information, such as keys and mnemonics. Our team investigated the design decision to keep copies of previous vaults in storage in order to prevent accidental data loss. We found that this design choice creates a potential vector for inadvertent recovery of deleted accounts, and for compromised passwords that have been changed to continue to be a liability for the user. We recommend that the Tally team implement one of the suggested remediations in order to address the issue (Issue B).

Code Quality

The Tally browser extension wallet's key handling code is well written and organized. The in-scope repositories adhere to best practice and classes have clear responsibilities and are loosely coupled.

Tests

Tests have been implemented to check the correct functionality of a small number of valid inputs, however, there are no tests that check that the library correctly fails for invalid inputs. We recommend implementing a robust test suite that checks for all classes of success and failure cases. This aids in identifying implementation errors and edge case scenarios that could cause the system to behave in unintended or unexpected ways (Suggestion 3).

Input Validation

Inputs to the `hd-keyring` library are not always validated sufficiently to be expected values. While we did not identify methods of exploitation, this could lead to unexpected behavior in the future. This applies to integers being in a valid range, but also to matching input salts to stored salt. As a result, we recommend implementing the suggested mitigations to help protect against unexpected behavior (Suggestion 1; Suggestion 2).

Documentation

The Tally team provided accurate and helpful documentation including the README and the Keyring Design document, which describes the general architecture of the system and the intended functionality of the key handling components. In addition, the high-level documentation describing the Tally browser extension wallet's system as a whole was sufficient.

Code Comments

The Tally browser extension wallet key handling implementation contains sufficient inline documentation, which clearly describes the intended behavior of the code. This aids security researchers and maintainers of the code to better understand the intended functionality, and identify potential errors or vulnerabilities in the implementation.

Scope

The in-scope repositories for this security audit were limited to the `hd-keyring` repository and `services/keyring` in the Tally browser extension wallet repository. All other components of the Tally browser extension wallet were explicitly considered out of scope. As a result, in reviewing the key handling components, we assumed that the rest of the system behaved as intended and does not introduce any security vulnerabilities. However, due to the limited scope of the security audit, we are unable to make statements about the general security of the Tally browser extension wallet in its entirety. As a result, we strongly recommend that the remaining components of the Tally browser extension wallet be audited by an independent security auditing team once all features are development complete (Suggestion 7).

Use of Dependencies

We did not identify any critical security concerns relating to the Tally browser extension wallet's use of dependencies. However, we identified an instance where an incorrectly unpinned dependency could cause breaking changes if an update is published. Thus, we recommend adhering to dependency library documentation guidelines (Issue C). Furthermore, we recommend pinning versions for all dependencies in order to reduce the likelihood of supply chain attacks (Suggestion 4).

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: Key Derivation Function Used for Passwords is Trivially Parallelizable	Unresolved
Issue B: No Provision for Old Vaults to be Deleted or Updated	Unresolved
Issue C: Version of @ethersproject/wallet Not Pinned	Resolved
Suggestion 1: Check That Wallet Counter Does Not Overflow Size in BIP32	Resolved
Suggestion 2: Verify the Salts Match During Decryption with SaltedKey	Unresolved
Suggestion 3: Improve Tests for Invalid Inputs	Unresolved
Suggestion 4: Pin Versions for All Dependencies	Unresolved
Suggestion 5: Preserve Address Checksums	Unresolved
Suggestion 6: Include a Counter in AES-GCM Nonce	Unresolved
Suggestion 7: Conduct a Comprehensive Audit of Tally Browser Extension Wallet	Unresolved

Issue A: Key Derivation Function Used for Passwords is Trivially Parallelizable

Location

`tally-extension/background/services/keyring/encryption.ts#L77-L96`

Synopsis

The Tally browser extension wallet 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 loss of user funds.

Preconditions

The attack requires access to the encrypted vaults and this could happen through multiple vectors. A malicious website extension would need to circumvent browser security measures. A regular program running with user permissions usually can 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). This means it has a tight loop and only requires a small amount of memory. This can be efficiently parallelized and even implemented on an FPGA or an 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 Section 4 of the Argon2 RFC, guidance is provided for the choice of parameters. We suggest selecting t=3 iterations, p=4 lanes and 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 Tally team has chosen not to resolve this issue at this time, as they are unsure of the maturity of Argon2. However, we maintain our recommendation of using Argon2id instead of PBKDF2. Agron2 has been declared winner of the Password Hashing Competition in 2015, and the `argon2-browser` package is compiled from the reference c implementation. It has undergone scientific research since publication. This leads us to believe that it is sufficiently tested for production, while also addressing the mentioned problems of PBKDF2.

Verification

Unresolved.

Issue B: No Provision for Old Vaults to be Deleted or Updated

Location

`tally-extension/background/services/keyring/storage.ts#L54-L88`

Synopsis

Every time encrypted data in the Tally browser extension wallet is modified, the new version is stored alongside all old versions. The old versions are kept in order to provide a recovery path in the event that an unexpected issue or error occurs. However, there are cases when deleting old vaults is desirable (e.g. when changing the password or deleting an account).

Impact

This may result in the leakage of private key material or proof of control over an account that has been deleted. This could lead to loss of funds or impact plausible deniability of deleted account data.

Preconditions

The attack requires access to the encrypted vaults (i.e. UI access to the browser or access to the filesystem with user privileges). In addition, one of two things must hold, depending on the scenario. If the user changes their password after noticing it is compromised, the attacker must also know the compromised password. They can then access the private keys as they were before the password was changed.

If the user deleted an account and claims they have never been the owner of that account, the attacker could compel the user to provide the password and decrypt previous vaults, which would prove their control over the account.

Feasibility

If the preconditions hold, the attack is trivial.

Technical Details

In the key handling code, vaults are encrypted containers that store the secret wallet information, such as keys and mnemonics. When the secret wallet information is changed, a new vault is created and added to the data store, instead of overwriting the existing contents. This means that all previous vaults remain in the database. A user who deliberately makes destructive changes to their vault, such as changing the password or deleting an account, would expect that the old data does not remain on the device.

Remediation

We recommend implementing one of the two following remediations:

- Provide a function that clears all encrypted vaults, or a flag to overwrite all existing vaults in the next update; or
- Provide a function that updates previous vaults, either for re-encrypting with a new key, or to sanitize vaults from data that is being deleted.

Status

The Tally team has responded that they will not be addressing this issue at this time because they are concerned about accidental loss of key material. The missing of intended key disposal is a significant security risk, and we consider accidental disposal of keys to be a manageable risks. For example, integrity checks could be added to make sure that the new vault data is not corrupted. Therefore, we still recommend implementing one of the suggested remediations.

Verification

Unresolved.

Issue C: Version of @ethersproject/wallet Not Pinned

Location

`hd-keyring/package.json`

`hd-keyring/src/index.ts#L160`

Synopsis

In the `hd-keyring` repository, the `package.json` file requires the @ethersproject/wallet package to be version 5.4.0 or higher. However, the documentation for the package recommends pinning the package because future versions might break the code due to the renaming of the `function _signTypedData`.

Impact

Future versions of the @ethersproject/wallet package might not work with the current version of the code, leading to broken builds for `hd-keyring`.

Remediation

We recommend pinning the @ethersproject/wallet to a specific version, such that `hd-keyring` can be built reliably, even when @ethersproject/wallet has renamed the function.

Status

The Tally team issued a commit and @ethersproject/wallet is now pinned to version 5.4.0 as recommended.

Verification

Resolved.

Suggestions

Suggestion 1: Check That Wallet Counter Does Not Overflow Size in BIP32

Location

`hd-keyring/src/index.ts#L182`

Synopsis

In the function `addAddressesSync`, the input variable `numNewAccounts` added to the current `addressIndex` should be checked to be less than 2^{31}, which is the maximum value for normal (non-hardened) child keys given in BIP32. Address generation is unlikely to exceed this value, however libraries always should check input values to be valid indexes for the desired kind of derivation (normal derivation instead of hardened derivation). BIP44 requires using normal derivation here, so hardened derivation would be a deviation from specification.

Mitigation
We recommend adding a check for the input value to be in the valid range, in order to perform the correct kind of derivation.

Status
The Tally team has added the input variable check as recommended.

Verification
Resolved.

Suggestion 2: Verify the Salts Match During Decryption with SaltedKey

Location
[tally-extension/background/services/keyring/encryption.ts]

Synopsis
The consumer of the service can specify a `SaltedKey` instead of a password for decrypting the vault, so that the expensive derivation can be cached. The case where the salt in the `SaltedKey` does not match that of the vault is not specifically checked. While this should not result in security issues, checking equality would explicitly catch (possibly accidental) misuse of the library, and provide better error messages.

Mitigation
We recommend checking that the salt of the `SaltedKey` and the `EncryptedVault` match. In the event that they do not match, an exception should be thrown.

Status
The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

Verification
Unresolved.

Suggestion 3: Improve Tests for Invalid Inputs

Location
[hd-keyring/test/keyring.test.ts]

[tally-extension/background/tests/keyring-integration.test.ts]

Synopsis
The currently implemented test cases check mostly the behavior when given valid inputs. However, tests that check for invalid inputs only check that an error is thrown, but do not check the error type.

Mitigation
We recommend improving the test cases to check for invalid inputs, edge cases, and that correct errors are thrown. Testing for edge cases and invalid inputs helps to confirm correct functionality and behavior. In addition, describing the behavior of the functions with test cases helps to prevent bugs in the code and regressions in case new code is introduced in the future.

Status

The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

Verification

Unresolved.

Suggestion 4: Pin Versions for all Dependencies

Location

`hd-keyring/package.json#L42-L61`

`tally-extension/package.json#L40-L98`

Synopsis

Package managers like npm and `yarn` provide fuzzy version matching, which allows specifying a minimum version, and if a newer version is available, it may be used instead. This allows fixes for dependencies to be automatically fetched and requires less time and effort for managing dependency versions. However, this also increases the possibility and feasibility for supply-chain attacks. If a single dependency (or dependency of a dependency) is taken over by an attacker without knowledge that an attack is occuring, the source code can be freshly built with the inclusion of malicious and harmful code in the library.

Mitigation

We recommend exactly pinning the versions of all dependencies, and updating the dependencies manually and conservatively.

Status

The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

Verification

Unresolved.

Suggestion 5: Preserve Address Checksums

Location

`hd-keyring/src/utils.ts#L19-L27`

`hd-keyring/src/index.ts`

Synopsis

Checksums contained in addresses are ignored, and the addresses are normalized to lower-case, eliminating the checksum. Validating checksums helps prevent user errors such as typos. For example, if the checksum is validated, users can be prevented from sending transactions to the wrong addresses because they did not transfer the address correctly.

Mitigation

We recommend validating addresses that are an input to the library, following EIP-55. This should contain a checksum verification, if the address contains one. Backwards compatibility is given in that libraries unaware of the checksum usually accept both lower- and upper-case addresses. Addresses that are returned by the library should also contain a checksum.

Status

The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

Verification

Unresolved.

Suggestion 6: Include a Counter in AES-GCM Nonce

Location

`tally-extension/background/services/keyring/encryption.ts#L128`

Synopsis

In the current implementation, the AES-GCM encryption nonce is 16 bytes long, and randomly sampled before encryption. As a result, there is a small chance of generating the same nonce twice, which would completely break the security of AES-GCM.

Impact

AES-GCM fails catastrophically in the event of a single nonce collision observed by an attacker. It is often recommended to use a counter as the nonce, if possible, which would guarantee uniqueness. However, depending on the attack model, this may not be simple.

Preconditions

The attacker needs access to the extension storage or browser profile of the user, and a nonce collision needs to have occurred.

In order to access the encrypted vaults, the attacker either needs a browser exploit to circumvent its security measures, or execute any program with user privileges. The difficulty of the latter varies, depending on the behavior and awareness of the user.

Feasibility

Nonce collisions are rare, but can occur, even when the attacker is not actively interfering with the functioning of the program.

Technical Details

The following attack models were taken into account:

- The attacker can observe all ciphertexts;
- The attacker can do the above and modify the stored wallet state (e.g. reset the vaults list to a previous state); and
- The attacker can do all of the above as well as trigger new encryptions.

The first attack model describes a passive attacker that can only observe. In this case, while the chance of nonce reuse is low, using a counter would completely resolve the possibility of nonce reuse.

In the second model, only using a counter is catastrophic, since the attacker could reset the wallet state to that of an earlier time and, upon the next encryption, the wallet would use the same nonce again, thus breaking the security of the encryption. To make such events less likely even during an active attack, parts of the nonce should remain random.

The last model gives the attacker a significant amount of power and, arguably, at this point the system could be considered entirely compromised. In this case, the attacker could keep resetting the wallet state and trigger new encryptions, which would lead to a new instance of the birthday problem. However, in this case, the random part of the nonce would be shorter (in order to make space for the counter) and a collision would be more quickly identified. That said, we consider this scenario less important than that with a passive attacker, and hence recommend prioritizing avoiding collisions without adversarial influence.

Mitigation

We recommend a nonce that consists of a four byte counter and eight random bytes. Since it is not expected that 2^{32} or more encryptions are made, a 32-bit unsigned integer value is sufficient in this case. The GHASH component of AES-GCM operates on 16-byte blocks, which also include the 4-byte block counter.

Status

The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

Verification

Unresolved.

Suggestion 7: Conduct a Comprehensive Security Audit of Tally Browser Extension Wallet

Synopsis

Our team performed a security audit of the Tally browser extension wallet key handling components. In doing so, we operated under the assumption that the rest of the system components functioned as intended. However, a comprehensive security audit of the system that includes all functionality is necessary, in order to assess each of the components in the context of its interaction with the system as a whole.

Mitigation

We recommend conducting a full, comprehensive security audit of the Tally browser extension wallet, including all security critical components and their interactions.

Status

The Tally team has responded that they have added this suggestion to their security backlog to be addressed in the future.

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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site 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. While 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, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, 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 though we have not yet 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 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 report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, 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 resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.