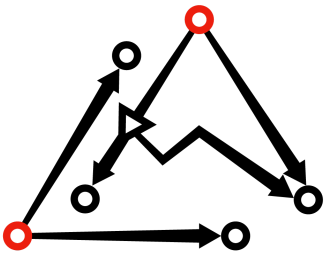

Least Authority

PRIVACY MATTERS

MetaMask Extension: Seed Phrase Implementation

Security Audit Report

ConsenSys AG

Final Audit Report: 29 July 2022

Table of Contents

This audit makes no statements or warranties and is for discussion purposes only.

Overview

Background

MetaMask requested that Least Authority perform a security audit of the MetaMask extension's seed phrase implementation. MetaMask is a browser extension wallet that handles account management and user interaction with the Ethereum blockchain, in addition to interacting with decentralized applications (dApps).

Project Dates

- **November 17 - December 14:** Code review *(Completed)*
- **December 17:** Delivery of Initial Audit Report *(Completed)*
- **July 26 - 28:** Verification completed *(Completed)*
- **July 29:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Alicia Blackett, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer
- Ann-Christine Kycler, Cryptography Researcher and Engineer
- Justin Regele, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the MetaMask Extension: Seed Phrase Implementation 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:

- MetaMask Browser Extension: https://github.com/MetaMask/metamask-extension
- Eth Keyring Controller: https://github.com/MetaMask/KeyringController
- HD Keyring: https://github.com/MetaMask/eth-hd-keyring
- Simple Keyring: https://github.com/MetaMask/eth-simple-keyring

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

MetaMask Browser Extension: `f7c6b3228b7ac0a94a98356415a465157091c31e`

Eth Keyring Controller: `138139aea5427478f33f16347549b2b3797f73bf`

HD Keyring: `65fc900593115511f6aaa9191fb525d4e144fb8a`

Simple Keyring: `a1ce8d2c360e348d49e980915a4a94feb270e4e2`

For the verification, we examined the Git revision:

MetaMask Browser Extension: `1caab93c07dd7bcdf90991f4c8d23d73e706610c`

This audit makes no statements or warranties and is for discussion purposes only.

Eth Keyring Controller: `0a92a4b2ef80b665ae8240881b3a1f2d7715d514`

HD Keyring: `52391a2a384aab545765838ed4bbe25d1608d0ed`

Simple Keyring: `f8c9105825953e926480e84c001e30ded5e8ebdf`

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

MetaMask Browser Extension:
https://github.com/LeastAuthority/metamask-extension-2

Eth Keyring Controller:
https://github.com/LeastAuthority/KeyringController

HD Keyring:
https://github.com/LeastAuthority/eth-hd-keyring

Simple Keyring:
https://github.com/LeastAuthority/eth-simple-keyring

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:

- MetaMask Browser Extension README.md:
  https://github.com/LeastAuthority/metamask-extension-2/blob/develop/README.md
- Eth Keyring Controller README.md:
  https://github.com/LeastAuthority/KeyringController/blob/main/README.md
- HD Keyring README.md:
  https://github.com/LeastAuthority/eth-hd-keyring/blob/main/README.md
- Simple Keyring README.md:
  https://github.com/LeastAuthority/eth-simple-keyring
- The Keyring Class Protocol:
  https://github.com/LeastAuthority/KeyringController/blob/main/docs/keyring.md
- Developer Documentation:
  https://docs.metamask.io/guide/
- MetaMask Internal Documentation:
  https://github.com/LeastAuthority/metamask-extension-2/tree/develop/docs
- Seed Phrase Claims History google document *(shared with Least Authority via Slack on 18 November 2021)*
- Guide.pdf *(shared with Least Authority via Slack on 7 November 2021)*
- Vault-history.zip *(shared with Least Authority via Slack on 18 November 2021)*
- Norton logs-20211119T184714Z-001.zip *(shared with Least Authority via Slack on 19 November 2021)*
- drive-download-20211118T195918Z-001.zip *(shared with Least Authority via Slack on 18 November 2021)*
- Log files analysis.xlsx *(shared with Least Authority via Slack on 19 November 2021)*
- Private key.pdf *(shared with Least Authority via Slack on 19 November 2021)*
- RSP adjust.pdf *(shared with Least Authority via Slack on 19 November 2021)*

- RSP write 2021.04.09.pdf *(shared with Least Authority via Slack on 19 November 2021)*
- TimeLine - Issues history.docx *(shared with Least Authority via Slack on 19 November 2021)*
- RSP adjust.pdf *(shared with Least Authority via Slack on 28 November 2021)*
- TimeLine - Issue history - updated.2021.12.13.docx *(shared with Least Authority via Slack on 13 December 2021)*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation of the seed phrase;
- The possibility of the seed phrase generating different account lists; and
- Anything else identified as critical to these areas during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of the MetaMask extension's seed phrase implementation. In particular, the focus of our investigation was a potential error in the seed phrase implementation, in response to MetaMask user claims that the same seed phrase can generate different account lists. According to the MetaMask team, the occurrence of this error has been credibly reported by a number of users. As a result, a concerted effort has been undertaken to consistently replicate the seed phrase error by MetaMask, one of the MetaMask users who reported the issue, and the Least Authority team. In our independent review, we performed a close investigation of the potential causes of the seed phrase error in an effort to identify appropriate solutions. In doing so, we also examined the overall correctness of the MetaMask extension seed phrase implementation.

Seed Phrase Error

Several MetaMask users have reported the occurrence of the seed phrase error. As a result, the MetaMask team has been in close communication with one MetaMask user in particular, who has reported and documented the inconsistency. Specifically, upon recovering a wallet from the seed phrase, a different wallet is generated that the user has not previously seen, instead of the user's previous wallet. From our analysis of the code and the MetaMask user's accounts, we believe that in some edge cases, the wallet generates a hidden, unknown seed phrase at a very early stage that overwrites the seed phrase the user has saved. However, we are uncertain about when and how this occurs.

Areas of Investigation

With the exception of users reporting the error and this particular MetaMask user's documented evidence of unexplained behavior, there is very limited information available regarding the seed phrase error or evidence of a bug in the implementation resulting in this error. As a result, our team attempted several approaches to reproduce the error, in addition to completing a close review of the seed phrase implementation more generally.

Overwritten Addresses

During our analysis of the documentation provided by the MetaMask user, we confirmed that two addresses had been corrupted into a single address, with one address overwriting a portion of another. A picture of a memory image provided to us displayed two addresses (which have been anonymized for this report):

`0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

and

`0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB`

had become corrupted into a single address of the form:

`0xAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB`

This indicates that corruption in memory or during storage occurred on the operating system (OS) or browser application level. The corrupted address exceeds the expected address length for a public key. Additionally, the offset at which address A is overwritten by address B does not fall on a word boundary, as would be expected if an error in splicing blocks had occurred. This leads us to conclude that an error occurred either in memory, which was later persisted to disk, or when attempting to write the address to disk.

As a result of this evidence of addresses being overwritten either in memory or on disk, the uncertainty of the execution environment in which the MetaMask extension is running guided part of our investigation efforts, which we outline further below.

Mnemonic and Seed Generation

We analyzed the seed generation from the mnemonic implementation and found that it follows best practices and industry standards. While initializing the wallet and generating the seed from the mnemonic, the function `mnemonicToSeed` is called from `bitcoinjs/bip39`. The generated seed is then used to generate an Ethereum HD key through the function `fromMasterSeed` from `ethereumjs/ethereumjs-wallet`, which uses the function `fromMasterSeed` from `ethereum/js-ethereum-cryptography`.

The mnemonic can also be generated before the wallet is initialized, in which case the seed is generated from the newly generated mnemonic. The function `generateMnemonic` is then called from `bitcoinjs/bip39`, which defaults to the 128-bit strength.

We did not identify any issues with the implementation of BIP39 in these libraries. In addition, we did not find any mismatches with the usage of these flows to generate an HD key from a given mnemonic, or when generating a mnemonic first and then passing it to generate an HD key. However, we found that the `addAccounts` function in the HD Keyring class serves a dual function of mnemonic generation and account creation and could have severe side effects in the event that `_initFromMnemonic` does not execute properly. As a result, we recommend decoupling the functionality of generating the mnemonic from `addAccounts`, as it could result in generating a new mnemonic of which the user is unaware (Suggestion 1). In addition, the correctness of mnemonics should be checked in order to adhere to the BIP39 standard. As a result, we recommend validating the passed mnemonics (Suggestion 5).

Race Conditions

We explored the possibility of LevelDB log files persisting between installations of the MetaMask extension. Once a user enters a password in the onboarding flow, a seed phrase is created for a wallet. In the event that a user then abandons the onboarding flow, the seed phrase may still appear in the LevelDB log file, depending on how far the onboarding process proceeds. This could have been the experience of the MetaMask user who reported the problem. In their supporting evidence, they describe “downloading and installing, but not creating, a wallet” on at least two occasions.

When the MetaMask extension is uninstalled, the intended behavior of the Chrome browser is that the LevelDB database found in the Local Extension Settings Directory is deleted. It may be possible that a

This audit makes no statements or warranties and is for discussion purposes only.

race condition occurs when uninstalling a MetaMask extension since removing the MetaMask extension is executed in a separate thread than that of the extension itself. This could result in the MetaMask extension's Local Extension Settings Directory persisting between installs. If LevelDB saves to the log file immediately after it is deleted, but prior to the call to the operating system to remove the directory, then the delete directory API call will fail since the Windows APIs for deleting directories require empty directories (RemoveDirectoryA; RemoveDirectoryW; RemoveDirectoryTransaction). This would leave both the directory and the log file intact.

Upon reinstallation of the MetaMask extension, LevelDB will initialize a new database, in which the log file from the previous installation will be renamed as `LOG.old`. It is unknown to our team how this `LOG.old` file could corrupt the current database, as LevelDB was out of scope for this security audit. However, we discovered particular commits to the LevelDB repository that indicate certain optimizations were made to load older log files when opening the database (see references in Suggestion 7). In addition, inspection of the log files over the course of the audit demonstrated that LevelDB will, at times, attempt to recover older versions. This could indicate that a combination of race conditions and failures could cause the LevelDB storage used by the MetaMask extension to revert to an earlier state. As such, we recommend a thorough security audit of LevelDB's use of older logs and the resulting implications for the functionality of the seed phrase implementation (Suggestion 7).

As mentioned above, we discovered that the seed phrase is generated once a user enters a password during the onboarding flow. If a user aborts at this point in the process prior to seeing the seed phrase, the seed phrase continues to exist in memory and can be written to disk. If the MetaMask extension is then uninstalled but the scenario described above has occurred, a user may end up with a seed phrase that they are unaware of after reinstalling the MetaMask extension.

We identified a sequence of preconditions and events that would likely be necessary to cause the seed phrase error to occur:

1. The MetaMask extension is installed, and a password is set, but the user aborts before seeing the seed phrase. Even though the user is unaware of this seed phrase, it is already loaded into heap memory and is potentially stored to LevelDB;
2. The LevelDB log file (`*.log`) persists the MetaMask extension uninstall through a race condition where LevelDB writes to the log file immediately before the browser attempts to remove the directory;
3. Upon reinstallation of the MetaMask extension, the user uses the same password as before; and
4. After onboarding, LevelDB encounters a problem with the log file and recovers from `LOG.old`.

In one instance, one of the Least Authority security researchers believes they observed this irregular behavior, but were unable to reproduce it further. This occurred while having a breakpoint set in the HD Keyring `addAccounts` function on a newly reinstalled extension, during which a previously stored seed phrase appeared. Due to the documentation by the MetaMask user, and our team observing irregular behavior consistent with the seed phrase error, we recommend a further investigation into LevelDB (Suggestion 7).

Multiple Keyrings Theory

We investigated the possibility that a second keyring was added to the Keyring Controller's keyrings array. The `getKeyringForAccount` function abstracts a way in which a keyring is used for a particular account, which would allow the user to use accounts from a second keyring seamlessly without user interaction.

When an account is added, the MetaMask controller will identify the primary Keyring by finding the first HD Key Tree keyring with a hard coded index of 0 (`addNewAccount`). In the event that there were multiple HD Key Tree keyrings in the Keyring Controller's keyrings array, the second HD Key Tree keyring would be

This audit makes no statements or warranties and is for discussion purposes only.

ignored. In a scenario where one HD Tree keyring has an unknown seed phrase, but a second HD Key Tree keyring has a seed phrase known to the user, this would lead to a scenario as experienced by the MetaMask user (i.e., recovering the wallet does not restore the intended account, if the account was linked to the HD Tree keyring with the unknown seed phrase). However, we were unable to identify a way for the Keyring Controller to contain multiple HD Key Tree keyrings in its keyring array.

Double Onboarding

Our team investigated the possibility of multiple keyrings being generated if a user onboards twice. In `registerOnboarding`, a check is made if `this.completedOnboarding` is `true`, indicating that the user has previously onboarded. However, `this.completedOnboarding` appears to be accessed incorrectly. When a user completes onboarding, the `completeOnboarding` function updates the state of `this.store` setting `this.store.completedOnboarding` to `true`. A proper check in `registerOnboarding` would check the value of `this.store.completedOnboarding`. Instead, a check to verify `this.completedOnboarding` is made, a value that will always be `false` as it is never set. As a result, the function will never abort because the check will always resolve to `false`, enabling users to onboard again. Our team was able to load two onboarding pages and to onboard multiple times to the wallet. Therefore, we recommend that the onboarding check be implemented correctly (Suggestion 10).

If a user onboards twice, we observed that two keyrings are stored, but the second is of type `Ledger Hardware` and has an empty accounts array. From the MetaMask controller, the function `addNewKeyring` is only called from `getKeyringForDevice` and `importAccountWithStrategy`. In the case of `getKeyringForDevice`, the Keyring type is determined by a switch statement that returns an error if a predefined hardware wallet is not used. Due to this switch statement, it would be impossible to create an HD Tree keyring through this function call. With `importAccountWithStrategy`, the keyring type is hard coded as a Simple Key Pair type and can therefore never be an HD Key Tree. Since these two code paths prevent the creation of a second HD Key Tree keyring, double onboarding would not cause a double HD Key Tree keyring to cause the seed phrase error.

`addNewKeyring`

We investigated the occurrences of `addNewKeyring` to see if it could be used to generate multiple HD Key Tree keyrings. This function is only called twice and both instances are called from within the Keyring Controller's `createNewVaultAndRestore` and `createFirstKeyTree` functions. We discovered a subtle difference in the sequence of calls that could potentially cause keyrings in storage to become out of sync in the event there was a temporary failure in the OS writing to disk, as detailed below.

In `createNewVaultAndRestore`, the call to `addNewKeyring` is preceded first by a call to `clearKeyrings`, which sets the keyrings array to an empty array, and then a call to `persistAllKeyrings`, which serializes the now empty keyring array to the vault. However, `createFirstKeyTree` only precedes the call to `addNewKeyring` with a call to `clearKeyrings`. This function is only called in a single place (inside of `createNewVaultAndKeychain`) where it is sandwiched between calls to `persistAllKeyrings`. The first persists the current state of the keyring, and the second overwrites it with the new keyrings.

The `persistAllKeyrings` function will only serialize and encrypt to the vault what is stored in the keyrings array. The subtle difference between `createNewVaultAndRestore` and `createFirstKeyTree` is that, in the former, an empty keyring is persisted to storage by first calling `clearKeyrings` and then `persistAllKeyrings`. In the latter, however, `persistAllKeyrings` is called before `clearKeyrings`, with the assumption that the keyrings array is already empty.

This subtle difference means that if `createNewVaultAndKeychain` is called when a keyring already exists, it would result in two separate writes to the storage of different keyrings, with the second

overwriting the first. Under normal operating circumstances, this would occur without problems. However, since our team has observed the aforementioned instance of a corrupted address (i.e., by being partially overwritten by another address), in the event of an edge case error scenario in the execution environment or in LevelDB, this difference could cause the keyrings in storage to become out of sync with the keyrings stored in memory. A method to protect against this would be to not assume the keyrings array is empty when calling `createNewVaultAndKeychain`. As a result, we recommend clearing the keyring in memory before persisting it to storage, as is done in `createNewVaultAndRestore` (Suggestion 8).

User Interface Investigation

Our team investigated locations in the User Interface (UI) implementation that trigger the creation of the keyrings private key import and json import. In both of these UI components, a button click or pressing enter on the keyboard creates keyrings. We investigated the implication of these two events happening nearly simultaneously and if that would result in two keyrings being stored in the Keyring Controller. We found that both flows create a Keyring through `importAccountWithStrategy`, which results in a Simple Key Pair keyring being created. As a result, an HD Key Tree keyring cannot be created using this method. Therefore, it is impossible to create a hidden HD Key Tree keyring that a user is not aware of.

Irregular API Response

We examined the log files sent by the MetaMask user to identify any past event that may have caused a synchronization issue between LevelDB and the wallet state. On at least two occasions, the b-cube API used by the MetaMask user appeared to drop a decimal point and post request times dated 1975. The erroneous API responses we observed were most likely requested on 6 April 2021 and 13 May 2021. The error handler calls the function `forceUpdateMetaMaskState` when this occurs, which has the potential to change the state of the wallet. The 6 April 2021 date comes after the MetaMask user's unknown wallet had already been created, so we do not believe this particular event could have caused the unknown wallet to be generated. However, we investigated whether a similar prior event could have caused the seed phrase error but were unable to draw any conclusions about an incorrect date causing a wallet's seed phrase to be reset.

Code Quality

Our team found the codebase of the MetaMask extension's seed phrase implementation to be well written and organized, and generally adhering to best practices. As detailed above, the usage of Ethereum and BIP39 libraries adheres to industry standards and we could not identify issues in the implementation of the mnemonic generation.

Tests

The in-scope repositories contain extensive test coverage. However, there are currently no tests implemented for complicated scenarios that could simulate the impact of unexpected user workflows on the application's state. We recommend expanding test coverage to include edge case scenarios more comprehensively, and refining the variables tested to identify process errors as well as results. This would aid in identifying edge case scenarios that could cause the system to behave in unintended or unexpected ways (Suggestion 2).

Documentation

The MetaMask extension seed phrase implementation documentation was thorough and comprehensive. In addition to the project documentation, the MetaMask team provided the MetaMask user's log files, the results of forensic type investigations carried out by the user, working directories used by the users Chrome browser, and a description of the steps the user followed (based on recollection) when the seed

This audit makes no statements or warranties and is for discussion purposes only.

phrase error was first observed. Furthermore, the MetaMask team attempted to collect data points from other impacted users. However, the information was limited, vague, and inconclusive.

Scope

The in-scope repositories for this security audit were limited to the MetaMask extension's seed phrase implementation. All other components of the MetaMask extension were explicitly considered out of scope. As a result, in reviewing the key handling components, we assumed that the rest of the system behaves 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 MetaMask extension in its entirety.

Moreover, due to the unique nature of the seed phrase error described in this report, we recommend that the scope of this investigation be expanded to include areas that were considered out of scope for this review (Suggestion 9).

Process Improvements

Given the limited information available about this error, it is difficult to predict the number of users that may be unknowingly impacted by its occurrence, particularly since users are only made aware of the seed phrase error when they are forced to restore access from the seed phrase. At the time of delivering this report, only a small number of MetaMask users have reported encountering a similar issue. However, this may not represent the actual number of impacted users. As a result, we recommend providing MetaMask users with incentives, resources, and tooling in order to verify that their seed phrase is correct when their private keys are still accessible. If the user seed phrase is stored incorrectly, consistent with the seed phrase error, the seed phrase for all involved keyrings should be discoverable from the LevelDB log files. We suggest simplifying this process for the user by utilizing a recovery mechanism (Suggestion 3).

Furthermore, we recommend defining a clear process for MetaMask users to report errors, which would be helpful in identifying errors and security vulnerabilities in similar situations, and in identifying solutions. We suggest that this process facilitate the gathering of all relevant data and include a questionnaire to gather background information about the nature of the error and the execution environment, in addition to enabling users to share relevant log files, screenshots, and other supporting documentation. Defining an internal process of this nature would also help in tracking user errors effectively, in addition to documenting and sharing findings, which facilitates future research and prevents duplicate effort on similar leads (Suggestion 11).

Specific 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.

SUGGESTIONS	STATUS
Suggestion 1: Decouple Mnemonic Generation from addAccounts	Resolved
Suggestion 2: Expand Test Coverage	Partially Resolved
Suggestion 3: Build Recovery Tool	Unresolved
Suggestion 4: Require User to Deliberately Generate a Mnemonic after Creating Password	Unresolved

This audit makes no statements or warranties and is for discussion purposes only.

Suggestion 5: Validate Generated Mnemonic	Resolved
Suggestion 6: Do Not Allow Users to Bypass Verifying the Generated Mnemonic	Unresolved
Suggestion 7: Investigate LevelDB's Use of Previous Logs	Resolved
Suggestion 8: Clear keyring from Memory before Persisting to Storage in createNewVaultAndKeychain	Resolved
Suggestion 9: Consider Exploring These Areas for Further Investigation	Resolved
Suggestion 10: Fix Onboarding Flow Check	Resolved
Suggestion 11: Develop and Document a Process for User Error Reporting and Response	Resolved

Suggestions

Suggestion 1: Decouple Mnemonic Generation from addAccounts

Location

`main/index.js#L45-L47`

Synopsis

The `addAccounts` function in the `HdKeyring` class serves a dual function of mnemonic generation and account creation. When this function is called, a check if `this.root` is null is made. If `this.root` is null, a mnemonic is generated. Under normal circumstances, `this.root` should always be set after a call to `_initFromMnemonic`. However, considering the presence of an error where users receive a mnemonic that they are unaware of, we consider this to be a dangerous coupling of functionality that would best be separated into different components.

This function is called during the onboarding flow, when unlocking the wallet, and when a new account is added. Creating a wallet begins with deserialization, through the `deserialize()` function, which takes an optional opts argument. If the opts argument includes a mnemonic, then `_initFromMnemonic` is called. Additionally, if opts has `numberOfAccounts`, then `addAccounts` is called. Another critical variable set in this function is `this.root`, which is set to null in the prelude of `deserialize()`.

When the MetaMask extension is first installed, and a fresh wallet is generated, the `deserialize()` function will not have an mnemonic in the opts object, and so, when `addAccount()` is called, `_initFromMnemonic` would not have been called and `this.root` will still be null. As a result, `addAccount()` is used to generate the mnemonic for the entire wallet.

`this.root` is set as the end result of the operations in `_initFromMnemonic`. When `_initFromMnemonic` is called, `this.mnemonic` is set with the argument passed to the function.Therefore, if this function were ever called twice, it would overwrite the original mnemonic.

The danger in this approach is in the following three function calls that result in `this.root` being set. If there is a failure in any of these functions, for any reason, including a failure from the larger execution environment or operating system, `this.root` will not be set, leaving the keyring in a state where the next call to `addAccount` will generate a new mnemonic to overwrite the original.

This audit makes no statements or warranties and is for discussion purposes only.

```
const seed = bip39.mnemonicToSeed(mnemonic);

this.hdWallet = hdkey.fromMasterSeed(seed);

this.root = this.hdWallet.derivePath(this.hdPath);
```

We investigated the code in these function calls for any potential edge cases or anomalous behavior that would cause a failure in rare cases but did not discover any irregularities.

Additionally, we found, through the use of a debugger, that a new seed phrase can only be generated during the unlocking of the wallet or when adding a new account. If the seed phrase is regenerated during wallet creation, the error is caught by `verifySeedPhrase`. However, during wallet unlocking, `addAccounts` is called twice. If the error occurs on the first call to `addAccounts`, then the mnemonic is overwritten silently, and a new wallet is generated. We observed that the MetaMask extension was unpredictable in whether this new mnemonic persisted across the further locking and unlocking of the wallet, indicating another issue regarding the integrity of the persistent storage.

Mitigation

We recommend removing the mnemonic generation from `addAccounts` and implementing it in a separate flow that does not obstruct unlocking the wallet or adding accounts to a keyring.

Status

The MetaMask team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 2: Expand Test Coverage

Location

metamask-extension-2/tree/develop/test

KeyringController/tree/main/test

eth-hd-keyring/tree/main/test

eth-simple-keyring/tree/main/test

Synopsis

Test coverage of the repositories in-scope is extensive. We inspected the end-to-end testing and found that while components are tested to prevent regression, more complicated scenarios that better simulate the impact of unexpected user workflows on the application's state could be added.

For example, the tests `from-import-ui.spec.js` do not test that a previous wallet has been removed, or whether keyrings have been added as expected. Additionally, some tests, such as in `from-import-ui.spec.js` and `lock-account.spec.js` only validate that the resulting account has 25 ETH in it, and not that the account's address is what was expected.

Mitigation

We recommend adding unexpected flows to the end-to-end testing suite to protect against edge cases where unusual user behavior could result in the corruption of the application state. As we suspect the

This audit makes no statements or warranties and is for discussion purposes only.

seed phrase error could be the result of system failure, running tests on systems with limited resources might also prove invaluable.

Additionally, we recommend that the success of tests be determined by more conditions being met than whether the wallet has 25 ETH. If the edge case occurs where the test restores a previously cleared wallet that also had 25 ETH in it, the test would not catch the seed phrase error. We recommend randomizing the ETH value across tests, as well as validating that the address of the wallet remains as expected.

Status

The Metamask team has created a PR to expand test coverage. However, there are components that are still not covered.

Verification

Partially Resolved.

Suggestion 3: Build Recovery Tool

Synopsis

The MetaMask user was able to discover the mnemonic used to generate the account with the lost funds through diligently decrypting every vault they found by manually parsing all the LevelDB files located in the Local Extensions Settings directory of the MetaMask extension. This process was successful because LevelDB is a time store of data and retains historical data rather than overwriting it. However, manually searching the binary data format is tedious and requires a level of technical skill that many users do not have.

Mitigation

We recommend building a recovery tool, as it could reveal any hidden mnemonics in the database if the Local Extension Settings Directory has been preserved. Since LevelDB is an open source project with a known file structure, it would be possible to create a tool that users who experience the issue could use to crawl all LevelDB files and decrypt all vaults found with their chosen password.

Status

The MetaMask team has rejected the suggested mitigation stating that the costs do not justify the benefits.

Verification

Unresolved.

Suggestion 4: Require User to Deliberately Generate a Mnemonic after Creating Password

Synopsis

Currently, the mnemonic phrase is generated and visible in heap memory once the user has created their password. However, it is not visible to the user, and they are unaware that it has been created. Not all users who install a wallet complete the full onboarding flow in one pass, as they may stop and return to the process after generating a password. Therefore, the user may be unaware that there potentially is a mnemonic already created, which could be used by the wallet. The user is then not able to detect if there has been an issue generating the mnemonic or that there might potentially be two mnemonics created through events outside of their control.

This audit makes no statements or warranties and is for discussion purposes only.

Mitigation

We recommend presenting users with a button that generates and reveals the mnemonic on the page where users are given the option to reveal the mnemonic.

Status

The MetaMask team has not implemented the suggested mitigation, stating that the scenario described above is highly unlikely due to the mutex used in the controller to prevent simultaneous vault generation.

Verification

Unresolved.

Suggestion 5: Validate Generated Mnemonic

Location

`index.js#L46`

Synopsis

The `addAccounts` function in the `HdKeyring` class serves a dual function of mnemonic generation and account creation. When this function is called, a check if `this.root` is null is made. If `this.root` is null, a mnemonic is generated through `generateMnemonic` from `bitcoinjs/bip39`. The call to the function `generateMnemonic` defaults to the 128-bit strength and generates the correct checksum through `entropyToMnemonic`.

As the checksum is being generated when `generateMnemonic` is called from `bitcoinjs/bip39`, the correct structure of the mnemonics is not checked in the implementation. The checksum ensures that even if, for example, the generation of mnemonics changes in the future, the code still checks that the mnemonics generated are valid. In addition, the BIP39 standard recommends that this validation be performed to aid in the upholding of the security assumptions of BIP39.

Mitigation

We recommend validating a generated mnemonic through `validateMnemonic` from `bitcoinjs/bip39`.

Status

The MetaMask team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 6: Do Not Allow Users to Bypass Verifying the Generated Mnemonic

Synopsis

Currently, when the user creates their initial wallet, they are given the choice between verifying the seed phrase immediately or at a later time. However, If the seed phrase generated becomes corrupted or is not the seed phrase expected by the user, they may deposit funds into the wallet before realizing that there is a problem. This could lead to the loss of funds if the user must recover the wallet using the seed phrase, but the seed phrase is associated with an account other than the one the user intended.

This audit makes no statements or warranties and is for discussion purposes only.

Mitigation

We recommend requiring the user to verify the mnemonic immediately after it is generated.

Status

The MetaMask team has chosen not to implement the recommended mitigation, stating that this is an intentional design decision.

Verification

Unresolved.

Suggestion 7: Investigate LevelDB's Use of Previous Logs

Location

https://codereview.chromium.org/793423002

https://github.com/google/leveldb/commit/ac1d69da31205a979b5a8510f33c31ae977530f0

Synopsis

LevelDB might be optimizing load time by loading older logs. This could detrimentally affect users if an unknown mnemonic is ever stored in LevelDB. As part of our review, we discovered the above-referenced change that indicates this sort of behavior might be part of the developers' intentions for LevelDB. Since LevelDB was out of scope for the audit, we did not investigate further.

Mitigation

We recommend that an independent auditing team perform a security audit of how LevelDB handles older log files and whether the presence of a mnemonic in an old log could cause the wallet to load an unexpected mnemonic.

Status

Our team and the MetaMask team have agreed that an audit of LevelDB functionality is not feasible for the MetaMask team.

Verification

Resolved.

Suggestion 8: Clear Keyring from Memory before Persisting to Storage in createNewVaultAndKeychain

Location

index.js#L75

Synopsis

The assumption in `createNewVaultAndKeychain` is that the keyrings array is empty. The function calls `persistAllKeyrings` twice: The first time to delete what is currently in storage, and the second time to persist the freshly created keyring. In the first call, the assumption is that the keyrings array is empty. If the keyrings array is not empty, this will result in two keyrings persisted to storage in a short sequence. This should not cause any issues. However, in light of the corrupted address we discovered in

This audit makes no statements or warranties and is for discussion purposes only.

storage, it can be assumed that writing to storage is not guaranteed to be consistent. This could result in the new vault getting out of sync from what is stored in memory.

Mitigation

We recommend preceding the first call to `persistAllKeyrings` in `createNewVaultAndKeychain` with a call to `clearKeyrings`.

Status

The MetaMask team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 9: Consider Exploring These Areas for Further Investigation

Synopsis

The scope of the current security audit was limited to the seed phrase implementation of the in-scope repositories and the documentation provided by the MetaMask user. While we were not able to identify the root cause of the seed phrase error, we do believe the audit highlighted a need for further inquiries into other aspects of the ecosystem in which the MetaMask extension operates.

Mitigation

We recommend that the MetaMask team further investigate:

- LevelDB;
- Chrome's consistency in writing data to storage from memory;
- Engineering an environment to test for the corruption of data in a controlled fashion (For instance, to have memory cleared while the `_initFromMnemonic` function is being run);
- Accumulating more information from users experiencing the problem, such as operating systems and versions, and hardware configurations;
- Reports of Antivirus software deleting Chrome data upon identifying a "Potentially Unwanted Program (PUP)" and deleting some essential LevelDB files in the process;
- Whether external API failures could cause the wallet state to become corrupted;
- The possibility of the Application Update Process causing inconsistency; and
- How similar reports from users of the mobile application could be related.

While auditing closed source systems like the Windows operating system is not possible, an inquiry into issues relating to memory usage and disk IO might provide more information.

Status

The MetaMask team has acknowledged this suggestion and continues to investigate the areas listed above.

Verification

Resolved.

Suggestion 10: Fix Onboarding Flow Check

Location

`scripts/controllers/onboarding.js#L73`

Synopsis

During the completion of the onboarding flow, the `completeOnboarding` function updates the state of `this.store` by setting `this.store.completedOnboarding` to `true`. In the `registerOnboarding` function, however, `this.completedOnboarding` is checked. This results in the check always being false, allowing the user to onboard again.

Mitigation

We recommend checking `this.store.completedOnboarding` in `registerOnboarding`.

Status

The MetaMask team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 11: Develop and Document a Process for User Error Reporting and Response

Synopsis

The MetaMask user's log files and the documentation that was provided to us during the security audit contained important information about the usage and the execution environment, which significantly aided our efforts. Similarly, providing MetaMask users with information about this error would likely assist in solving similar problems and solicit additional user feedback that may help in identifying the cause of the error.

Given that users are only made aware of the seed phrase error when they are forced to restore access from the seed phrase, it is likely that there are users who are unknowingly impacted by its occurrence. As a result, further investigation across the user base may provide MetaMask with helpful information in an effort to resolve the error. Furthermore, notifying users of this potential error would allow them to secure access to their funds before the relevant data is potentially lost.

Mitigation

We recommend developing a process that facilitates error reporting and response that includes, but is not limited to, the following:

- Defining steps for obtaining relevant information about the execution environment, log files, and other relevant application data from MetaMask users reporting similar problems;
- Utilizing the existing MetaMask user error reporting and support tools (e.g., MetaMask user forums) to help the MetaMask team respond effectively to similar reports;
- Creating a questionnaire to gather usage information from users that can aid further investigation into the error; and
- Including a process for properly handling Personal Identifiable Information (PII), given that log files may contain sensitive user information.

Status

The MetaMask team has developed questionnaires for the mobile wallet and the browser extension wallet to collect more specific information on errors in the implementation.

Verification

Resolved.

This audit makes no statements or warranties and is for discussion purposes only.

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, and zero-knowledge 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.