Least Authority
PRIVACY MATTERS

MetaMask Snap

Security Audit Report

Capsule

Final Audit Report: 4 January 2024

Table of Contents

Overview

Background

Capsule has requested that Least Authority perform a security audit of their MetaMask Snap and conduct their review in accordance with industry best practices.

Project Dates

- **October 24, 2023 - October 26, 2023:** Initial Code Review *(Completed)*
- **October 26, 2023 - November 1, 2023:** Project on hold
- **November 1, 2023 - November 3, 2023:** Initial Code Review - continued *(Completed)*
- **November 6, 2023:** Delivery of Initial Audit Report *(Completed)*
- **January 4, 2024:** Verification Review *(Completed)*
- **January 4, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nikos Iliakis, 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 Capsule's MetaMask Snap followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- Capsule MetaMask Snap:
  https://github.com/capsule-org/mm-snap-keyring/tree/norwood/start

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

- `df579ef05697baa94b233c00604582bf11bc280b`

For the verification, we examined the Git revision:

- `dcc59675fcc3203ec793bade8522a29daa534c1d`

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

- Capsule MetaMask Snap:
  https://github.com/LeastAuthority/capsule-mm-snaps/tree/norwood/start

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:

- Capsule Docs:
  https://docs.usecapsule.com

In addition, this audit report references the following documents:
- MetaMask Documentation | getAccount():
  https://docs.metamask.io/snaps/reference/keyring-api/type-aliases/Keyring/#getaccount

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the Snap implementation;
- Potential misuse and gaming of the Snap;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Adversarial actions and other attacks on the network;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the Snap or disrupt the execution of the Snap capabilities;
- Vulnerabilities in the Snap code;
- Protection against malicious attacks and other ways to exploit Snap code;
- 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 comprehensive security review of the Capsule MetaMask Snap. The Snap is a wrapper of the Capsule Signing and Permissioning Toolkit, which provides keyring functionality through MetaMask. We investigated the coded implementation to identify security vulnerabilities and implementation errors and reviewed Capsule's utilization of the MetaMask security framework and adherence to security best practices.

System Design

Our team found that security has been taken into consideration in the design of the Capsule MetaMask Snap, as demonstrated by the due diligence undertaken to ensure that sensitive information does not leak outside to the secure JavaScript environment. We investigated the storage and handling of secrets in the implementation, in addition to Capsule's utilization of the MetaMask security framework and the use of permissions, and could not identify any issues.
While our team did not identify any security vulnerabilities in the use of permissions, we caution that using the origin header of a request to specify which permissions are available is not a best practice due to the same-origin policy (SOP) relaxation/cross-origin resource sharing (CORS) required for Snap usage. Additionally, we identified several areas of improvement relating to error handling, UX functionality, and the logging of sensitive information (Suggestion 1) (Suggestion 2) (Suggestion 3).

Code Quality

Our team found the Snap implementation to be well-organized and mostly in adherence to the guidelines set out by MetaMask.

Tests

During our review, our team found no tests within the codebase. We recommend implementing a test suite, which helps identify implementation errors that could lead to security vulnerabilities (Suggestion 5).

Documentation and Code Comments

While the Capsule toolkit has project documentation, the MetaMask Snap package currently lacks these resources, likely due to its early development stage. We recommend that the Snap documentation be improved (Suggestion 6). Our team also found that there were only some auto-generated comments about the inputs and outputs for some functions. We recommend improving code comments to sufficiently describe security-critical components and functions in the codebase (Suggestion 4).

Scope

The scope of this review was sufficient and included all security-critical components. Given that the underlying functionality has already been reviewed in a separate audit delivered by our team on October 6, 2023, the functionality provided by this Snap has been audited end to end.

Dependencies

Our team did not identify any security concerns resulting from the unsafe use of dependencies.

Specific Issues & Suggestions

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

ISSUE / SUGGESTION	STATUS
Suggestion 1: Adhere to MetaMask Keyring API Guidelines	Resolved
Suggestion 2: Improve the UX on Capsule Connect	Unresolved
Suggestion 3: Remove Logger in Production	Unresolved
Suggestion 4: Improve Code Comments	Unresolved
Suggestion 5: Implement a Test Suite	Unresolved
Suggestion 6: Improve Documentation	Unresolved

Suggestions

Suggestion 1: Adhere to MetaMask Keyring API Guidelines

Location

`snap/src/keyring.ts#L122`

Synopsis
The function `getAccount` in the [Metamask documentation]() has the signature `Promise< undefined | keyringAccount>` and should thus return `undefined` if it fails to find an account. However, the current implementation throws an error.

Mitigation
We recommend updating the implementation to adhere to all MetaMask implementation guidelines.

Status
The Capsule team has updated the `getAccount` function, as recommended.

Verification
Resolved.

Suggestion 2: Improve the UX on Capsule Connect

Location
[`src/pages/index.tsx#L67`]()

Synopsis
Using the `Connect` login requires the user to use a passkey, which might not be an option in certain cases (for example, if an organization does not allow it). Additionally, the sign-up process is unidirectional and does not allow the user to change their details on a previous step. Furthermore, there is no logout functionality. Hence, if the process fails, the site and Snap have to be torn down and rebuilt.

Mitigation
We recommend resolving the aforementioned issues.

Status
This suggestion remains unresolved at the time the verification review was performed.

Verification
Unresolved.

Suggestion 3: Remove Logger in Production

Location
Multiple locations throughout the codebase.

Synopsis
A logger is used to display error messages for debugging purposes and warn the users, depending on the environment (production or not).

Mitigation
We recommend removing the logger in production to avoid leaking sensitive data to the console by inadvertently propagating information provided by MetaMask.

Status
This suggestion remains unresolved at the time the verification review was performed..

Verification

Unresolved.

Suggestion 4: Improve Code Comments

Location

`snap/src/keyring.ts`

Synopsis

Currently, the codebase lacks explanation, specifically for the keyring that handle sensitive functionalities, such as message signing. Comprehensive in-line documentation (in a selected format that is consistently followed in the rest of the codebase)explaining expected function behavior and usage, input arguments, variables, and code branches can greatly benefit the readability, maintainability, and auditability of the codebase.

Mitigation

We recommend expanding and improving the code comments within the aforementioned areas to facilitate reasoning about the security properties of the system.

Status

This suggestion remains unresolved at the time the verification review was performed..

Verification

Unresolved.

Suggestion 5: Implement a Test Suite

Synopsis

Our team found no tests in the repository in scope. A test suite that includes a minimum of unit tests and integration tests adheres to development best practices. In addition, end-to-end testing is also recommended to assess if the implementation behaves as intended.

Mitigation

We recommend creating a test suite that includes a minimum of unit tests for the keyring functions, and integration tests to ensure that permission control is proper.

Status

This suggestion remains unresolved at the time the verification review was performed..

Verification

Unresolved.

Suggestion 6: Improve Documentation

Synopsis

The Snap component is insufficiently documented, which inhibits maintenance, security review, and safe use of the system by users.

Mitigation

We recommend creating user documentation to help users make informed security decisions when using the Snap. The documentation should detail the permissions utilized by the Snap as well as the reason justifying the use of these permissions.

Status

This suggestion remains unresolved at the time the verification review was performed..

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.