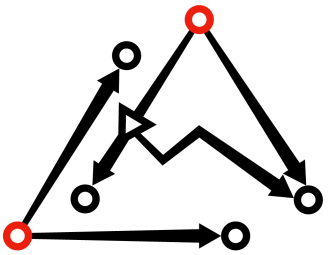

Least Authority

PRIVACY MATTERS

MetaMask Snaps-Extension Integration

Security Audit Report

ConsenSys Software, Inc.

Initial Audit Report: 19 July 2023

This Initial Audit Report is intended for internal use and discussion purposes only. We advise against sharing this report beyond trusted team members and recommend that publication take place only after the verification has been completed and the Final Audit Report has been delivered.

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - Areas of Investigation
    - System Design
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: No Calls to SnapsRegistry:updateBlockedSnaps Found
  - Suggestions
    - Suggestion 1: Remove Override Functionality
    - Suggestion 2: Check Origin of Untrusted Communication Channels
- About Least Authority
- Our Methodology

Overview

Background

ConsenSys Software Inc. has requested that Least Authority perform a security audit of their MetaMask Snaps Extension.

Project Dates

- **June 13, 2023 - July 17, 2023:** Initial Code Review *(Completed)*
- **July 19, 2023:** Delivery of Initial Audit Report *(Completed)*
- **TBD:** Verification Review
- **TBD:** Delivery of Final Audit Report

The dates for verification and delivery of the Final Audit Report will be determined upon notification from the MetaMask team that the code is ready for verification.

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Benoit Donneaux, Security Researcher and Engineer
- Ann-Christine Kycler, Security Researcher and Engineer
- JR, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the MetaMask Snaps Extension 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 Extension:
  https://github.com/MetaMask/metamask-extension
  - Only code inside fences is in scope ///:BEGIN:ONLY_INCLUDE_IN(snaps), and any code outside of the fences is out of scope for this review.

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

- `1a8a263cc1ffc420056ac1ede6659349a4bb5efa`

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

- MetaMask Extension:
  https://github.com/LeastAuthority/MetaMask_Extension_Snaps

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 Introduction Documentation: https://docs.metamask.io/guide/snaps.html
- Snaps Platform Audit Scope 2022-11.pdf *(shared with Least Authority via email on 9 November 2022)*
- MetaMask Snaps Diagram.pdf *(shared with Least Authority via email on 9 November 2022)*
- Snaps Vector Attack Tree Example.pdf *(shared with Least Authority via email on 8 December 2022)*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the wallet;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Malicious attacks and security exploits that would impact the wallet;
- Vulnerabilities in the wallet code and whether the interaction between the related and network components is secure;
- Exposure of any critical or sensitive information during user interactions with the wallet and use of external libraries and dependencies;
- Proper management of encryption and storage of private keys;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Additionally, the MetaMask team requests that the following be addressed in the Snaps Platform Audit:

Possible Critical Issues:

- Exfiltrating MetaMask's internal state.

Possible Major Issues:

- Corrupting MetaMask's internal state (either in-memory or persistent);
- Executing Denial of Service (DoS) attacks on the main MetaMask extension; and
- Exfiltrating state of other Snaps (either in-memory or persisted through `snap_manageState` endowment).

Possible Minor Issues:

- Corrupting another Snap's internal state (either in-memory or persisted through `snap_manageState` endowment);
- Corrupting the execution environment iframe's state;
- Allowing the dApp to connect to a Snap without previous user approval; and
- Executing DoS attacks on other Snaps.

Findings

General Comments

MetaMask Snaps (Snaps) is a platform intended to enable the interaction between the MetaMask Extension Wallet users and dApps. The Snaps system is composed of a wallet component that is controlled by the MetaMask team, and a plugin (Snaps) component that is implemented by third-party

developers. Snaps provides a mechanism for a user to grant dApps permissions to execute actions in the wallet.

Our team reviewed the MetaMask Snaps implementation in a previous review, with the initial audit report delivered on June 12, 2023. In this review, our team performed a security audit focusing on the Snaps - MetaMask Extension integration. The scope of this review was limited to the developments to the MetaMask Extension needed to enable Snaps functionality.

A noteworthy characteristic of the threat model adopted by the MetaMask team in considering the design and implementation of the MetaMask Snaps and the Extension, is the assumption that the user device is secure. This implies that compromise of the user device or filesystem would render user secret data vulnerable. As a result of these assumptions, the classes of attacks considered by our team during this review were limited.

The areas of concern listed above and the threat model outlined by the MetaMask team guided our review. In parallel, our team supplemented and reinforced the review by performing a threat tree exercise, which facilitated brainstorming, documenting, and fleshing out potential attack vectors and additional areas of concern.

Areas of Investigation

During the review, our team examined several areas for potential security vulnerabilities. We looked at how the extension renders Snaps-generated content, and investigated possible Cross-Site Scripting (XSS) attacks. We found that the Dialog enforces strict controls on what is rendered, and that the rendering is performed in React.

Build and Registry Bypass

We examined how the extension build settings affect whether Snaps could be installed locally or only from a designated registry. While we were not able to exploit any registry bypass, we do have concerns over whether a malicious npm package could manipulate the code to construct an arbitrary registry URL. The affected code is here. The concern is that the `username` could be an arbitrary domain, and the `password` an arbitrary port, both controlled by an attacker. Because this code appends an `@` character to the URL, this would prevent a malicious URL from becoming valid, unless the URL comment character `#` is injected into the `password` component. We were not able to exploit this, so the attack vector is entirely theoretical at this time.

Memory Leaks

We investigated whether Snaps could read data from memory using unallocated Buffers. Although unallocated buffers are known to leak memory in a Node environment, we found that this is not the case in the browser. In the browser, all buffers – even those allocated unsafely – are empty (filled with null bytes) since they are all initialized as `UInt` arrays. While a memory leak vulnerability in the V8 or SpiderMonkey JavaScript engines could lead to the exposure of data from different Snaps, these classes of vulnerabilities are considered out of scope for this audit and were not considered as potential attack vectors.

Communications

We investigated the use of communication channels in the target implementation and did not identify security vulnerabilities in this area. However, we recommend checking the origin of untrusted communication channels (Suggestion 2).

SES Override

We found an optional override function, `createSnapExecutionService`, that is the remnants from an experimental desktop feature no longer in use. This function has the crucial task of building the SES execution environment for Snaps. Therefore, the security of the entire Snaps platform relies on the secure implementation of SES. Because the override behavior is undefined, we have no way to determine whether it would be secure. It is therefore our recommendation that this override function be removed to prevent any misuse in the future (Suggestion 1).

System Design

We found that security has been taken into consideration in the design of the Snaps functionality. Our team only identified a single Issue in the design and implementation of the review's target functionality. We also identified concerns that are out of scope, but critical nonetheless. For example, the nested nature of the system with many layers of software, in addition to the update and patch management, could put the Snaps system at risk.

Moreover, Snaps must be allowlisted after a security audit. However, there is no clear process for approving updates to allowlisted Snaps and removing Snaps that are no longer allowlisted. Hence, the scaling of the Snaps functionality could be unwieldy, putting the system and its users at risk. Additionally, our team identified an Issue in the handling of blocked Snaps by the extension (Issue A).

Code Quality

Our team found that the implementation is consistent in its quality with our previous review. The in-scope code is well-written and adheres to best practice in implementation and tests. We found that the general architecture of the MetaMask Snaps Extension is well-designed.

Tests

We found that sufficient test coverage of Snaps is implemented to check for implementation errors and unexpected behavior that could lead to security vulnerabilities.

Documentation

The MetaMask ecosystem, including Snaps and the extensions, is generally well-documented.

Code Comments

The in-scope code is well-commented, with functions and their parameters well-described.

Scope

The second phase of this review targeted the functionality within the MetaMask Extension supporting Snaps. Given our previous review of the Snaps implementations, our team found the scope of this review included all security-critical components.

Dependencies

The MetaMask Extension uses LavaMoat to safeguard against dependency risks and supply chain attacks.

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: No Calls to SnapsRegistry:updateBlockedSnaps Found	Reported
Suggestion 1: Remove Override Functionality	Reported
Suggestion 2: Check Origin of Untrusted Communication Channels	Reported

Issue A: No Calls to SnapsRegistry:updateBlockedSnaps Found

Location

`src/snaps/SnapController.ts#L953`

Synopsis

The `SnapsRegistry:updateBlockedSnaps` is exposed to the MetaMask system to update the blocked Snaps list and disable any Snaps that have been deemed dangerous by the MetaMask team. No calls were found to update the blocked list and block currently installed Snaps in the Snaps Monorepo, the Extension, or the Core repository. Because of this, if a Snap is installed and later becomes blocked, the system designed to protect the user from the blocked Snap is not being activated.

Impact

A blocked Snap will remain active in a user's wallet after it has been blocked.

Preconditions

A Snap would need to be installed before it is added to the blocked list.

Remediation

We recommend making the extension perform the call `SnapsRegistry:updateBlockedSnaps` at reasonable intervals.

Status

Reported.

Suggestions

Suggestion 1: Remove Override Functionality

Location

`app/scripts/metamask-controller.js#L868`

Synopsis

An override functionality is provided to create the SES Execution Environment. Since this override function is not defined, our team was unable to assess its security.

Mitigation

Due to the critical nature of the code's function, we recommend opting for a clearly defined code path with no potentially dangerous uses in the future. Additionally, we recommend removing the override functionality and only instantiating the `IframeExecutionService` directly.

Status

Reported.

Suggestion 2: Check Origin of Untrusted Communication Channels

Location

app/scripts/metamask-controller.js#L3714-L3716

Synopsis

The `setupProviderConnection` function has an Issue that is unlikely to be exploitable except in incredibly extreme conditions that we were not able to replicate.

The function begins with an `if/else` branch that checks whether the `SubjectType` is `Internal` or `Snap`. If the `SubjectType` is `Internal`, then the `origin` variable is set to the value `ORIGIN_METAMASK`, which is the simple string `metamask`. If the communication channel is set up to come from a Snap, this branch will not be taken. Instead, the branch where the `SubjectType` is `Snap` will be taken. Here the `origin` is set to the value of `sender.snapId`. Because the `snapId` is partially controlled by the attacker, we observed that if `snapId` could be somehow set to the string `metamask`, this would have the same effects as `SubjectType` being set to `Internal`. However, we were unable to find a way to get a `snapId` to equal the plain string `metamask` because it required either manipulating the network or npm registries in extreme ways. Considering that all Snaps will require being audited, our team considers that the scenario where a third-party Snap is allowed to have the `snapId` "`metamask`" is highly unlikely.

Mitigation

We recommend including a check to verify that the `senderId` in the Snaps code branch does not equal `ORIGIN_METAMASK` before proceeding with setting up the communication channel.

Status

Reported.

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.