Least Authority

PRIVACY MATTERS

Metamask Snap

Security Audit Report

Blockfence

Final Audit Report: 25 July 2023

Table of Contents

Overview

Background

Blockfence has requested that Least Authority perform a security audit of their Metamask Snap implementation.

Project Dates

- **June 19, 2023 - June 21, 2023:** Initial Code Review *(Completed)*
- **June 22, 2023:** Delivery of Initial Audit Report *(Completed)*
- **July 25, 2023:** Verification Review *(Completed)*
- **July 25, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Alejandro Flores, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the MetaMask Snap 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:
- Blockfence-io Snap:
  https://github.com/blockfence-io/snap

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

- `e9c4a6c0f8d2355e98dc13d97d6dd196311dbc61`

For the verification, we examined the Git revision:

- `a09c9bd69327f40848ff27170c4c45fa32217223`

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

- Blockfence-io Snap:
  https://github.com/LeastAuthority/Blockfence_Snap-estimation

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:
- N/A

In addition, this audit report references the following documents:

- Snaps design guidelines: https://docs.metamask.io/snaps/concepts/design-guidelines
- Snaps permissions: https://docs.metamask.io/snaps/reference/permissions
- Restricted methods: https://docs.metamask.io/snaps/reference/rpc-api/#restricted-methods

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Potential misuse and gaming;
- Attacks that impacts 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 its execution;
- Vulnerabilities in the code;
- Protection against malicious attacks and other ways to exploit 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

Blockfence is a system that provides users of smart contracts and dApps security information by flagging transactions that are suspected or known to be malicious.

Our team performed a comprehensive review of the MetaMask Snap implementation, which allows users to integrate Blockfence functionality into a MetaMask wallet. We focused on the adherence of the implementation to MetaMask Snap development guidelines, and found that the naming convention in the implementation does not adhere to the recommendations (Suggestion 5). Our team investigated the permissions granted to the Blockfence Snap and did not identify errors in the implementation of excess authority.

In our review, we found that although the Blockfence Snap can learn sensitive user-identifying data, it does not utilize functionality that handles user secret data. We recommend improving the quality of the implementation and considering the security and handling of user private data in the Blockfence system at large.

Code Quality

Our team found the Blockfence Snap implementation code to be in need of improvement, with instances of unused code and permissions in the implementation (Suggestion 2). We also identified opportunities for improvement in error handling (Suggestion 1).

Tests

We found that no tests are implemented in the in-scope repositories. We suggest implementing a test suite to aid in identifying implementation errors and potential security vulnerabilities (Suggestion 3).

Documentation

There was no project documentation provided for this review. A lack of documentation hinders the ability to understand the intention of the code, which is critical for assessing the security and the correctness of the implementation. We recommend creating comprehensive project documentation for the Snap implementation (Suggestion 8).

Scope

The scope of this project was restricted to the Snap. However, our team found that the scope should have been expanded to additionally include the user interface. While reviewing the Snap, our team assumed that there are no vulnerabilities in out-of-scope components, and that the system is operating as intended. However, our findings show that the MetaMask Snap is still in its early stages of implementation. We recommend that the Blockfence team commission a comprehensive security audit of the entire system once development is finalized and design features are complete (Suggestion 6).

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: Expand Error Handling and Verbosity	Resolved
Suggestion 2: Remove Unused Code and Permissions	Resolved
Suggestion 3: Create Tests	Resolved
Suggestion 4: Remove Logging for Production	Resolved
Suggestion 5: Adhere to MetaMask Best Practices	Resolved
Suggestion 6: Perform Comprehensive System Audit (Out of Scope)	Unresolved
Suggestion 7: Assess the Impact of Hard Coding the API Key for Blockfence's API	Unresolved
Suggestion 8: Create Project Documentation	Unresolved

Suggestions

Suggestion 1: Expand Error Handling and Verbosity

Location

`snap/src/index.ts#L46`

Synopsis

When reaching Blockfence's API endpoint, if a response is not accepted, an error would occur. However, the user would be unable to determine the type of error, as only one generic error would be displayed, and different possible errors are not handled individually.

Mitigation

We recommend adding different errors for users to facilitate easier understanding, troubleshooting, and reporting of the failure. We further suggest experimenting with different error scenarios so all expected errors can be handled.

Status

The Blockfence team has implemented more descriptive server-side errors and the Snap currently shows generic error only if it fails to fetch a specific error message from the server side.

Verification

Resolved.

Suggestion 2: Remove Unused Code and Permissions

Location

`packages/snap/snap.manifest.json#20`

Synopsis

There are instances of unused code, including the code adapted from the MetaMask Snaps repository, that reduce the readability of the code, thereby increasing the risk of vulnerabilities being missed.

Mitigation

We recommend removing all unused code from the codebase.

Status

The Blockfence team has removed unused permissions from the Snap Manifest file. Additionally, unused code residuals from cloning MetaMask's `template-snap-monorepo` have also been removed.

Verification

Resolved.

Suggestion 3: Create Tests

Synopsis

Sufficient test coverage can help identify potential edge cases and protect against errors and bugs that may lead to vulnerabilities or exploits.

Mitigation
We recommend that the Blockfence team create a test suite for the Snap implementation, including tests for API errors and handling of data.

Status
The Blockfence team acknowledged the suggestion but stated that there is not enough logic within the Snap's functionality to justify creating an automated testing suite. Therefore, we consider this suggestion resolved.

Verification
Resolved.

Suggestion 4: Remove Logging for Production

Synopsis
The current implementation has logging enabled for debugging purposes. Although this is useful in the development process, it can be a vector for leaking sensitive data in a production release.

Mitigation
We recommend removing logging functionality from the production version of the Snap.

Status
The Blockfence team has removed logging functionality from the codebase.

Verification
Resolved.

Suggestion 5: Adhere to MetaMask Best Practices

Location
`packages/src/snap.manifest.json#3`

Synopsis
We found that the Snap implementation does not adhere to MetaMask naming conventions, which could result in unintended behavior.

Mitigation
We recommend adhering to all MetaMask implementation guidelines.

Status
The Blockfence team has modified the Snap codebase to make it more compliant with MetaMask's naming conventions and implementations guidelines.

Verification
Resolved.

Suggestion 6: Perform Comprehensive System Audit (Out of Scope)

Synopsis

During this review, our team operated under the assumption that the out-of-scope components of the Blockfence system are safe, and function as intended. Given that the system is able to collect user identifying data, a comprehensive audit of the system by a third party is needed to review the overall security and handling of sensitive data throughout the system.

Mitigation

We recommend performing a comprehensive security audit that includes all the components of the Blockfence system.

Status

The Blockfence team responded that the suggestion is out of scope.

Verification

Unresolved.

Suggestion 7: Assess the Impact of Hard Coding the API Key for Blockfence's API

Location

`packages/snap/src/index.ts#L12`

Synopsis

The API Key is hard coded and is used to perform requests to Blockfence's API. Exposing the key to a resource that works only with authenticated tokens raises the question of whether this key should be in plaintext or even available to anyone that can access the source code. This opens an attack vector for exploiting other endpoints of the API and overwriting or stealing data. Additionally, an attacker might potentially overload the backend service, which can lead to incurred cost or availability problems for the system.

Mitigation

We recommend that the Blockfence team evaluate the use of hard coded API keys in the codebase and consider other solutions to execute API calls, such as using a proxy server to provide authentication tokens to requests only coming from a Snap protected by an integrity check. We further recommend adding timestamps in requests sent to the backend servers and actively rejecting delayed calls to provide extra protection against replay attacks.

Status

The Blockfence team stated that since the API Key is already public, there should not be any issue with it being exposed in the code. Although our team cannot assess the actual benefits of using an authentication proxy for this implementation since it handles parts of the implementation that were outside of the scope of this audit, we still suggest that the Blockfence team evaluate the mitigation and consider keeping the API keys/tokens uncompromised, as some benefit of doing so can include:

1. preventing the actual API back-end IP/URL addresses from being exposed (reduced attack surface);
2. keeping the token private and behind a proxy server, thereby reducing the probability of attackers leveraging executing attacks such as MiTM and DDoS; and

3. creating a centralized security gateway for authenticating or managing access to the backend, which would allow for future seamless horizontal scaling, changes in the authentication schemes, and even support for different client types.

Verification

Unresolved.

Suggestion 8: Create Project Documentation

Synopsis

A lack of documentation can lead to misunderstandings in the future development process, which could result in security vulnerabilities being missed. Documentation on how components of the system function serves as a critical reference point that can be compared against what has been implemented in the codebase.

Mitigation

We recommend creating project documentation that consists of:

- a user-oriented description of the Snap, including its purpose, usage, as well as an explanation justifying why the permissions requested are needed;
- general security recommendations and implications in case of a misuse (if applicable); and
- a clear process for users and maintainers to respectively provide and collect the feedback needed to determine the causes of possible bugs.

Status

The Blockfence development team acknowledged the need for more comprehensive documentation and stated that they plan to create and expand the project documentation prior to publishing a public release.

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.