Least Authority

PRIVACY MATTERS

Masca MetaMask Snap

Security Audit Report

Blockchain Lab

Final Audit Report: 6 September 2023

Table of Contents

Overview

Background

Blockchain Lab has requested that Least Authority perform a security audit of their Masca MetaMask Snap.

Project Dates

- **August 14, 2023 - August 23, 2023:** Initial Code Review *(Completed)*
- **August 25, 2023:** Delivery of Initial Audit Report *(Completed)*
- **September 5:** Verification Review *(Completed)*
- **September 6:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, 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 the Masca 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:
- Blockchain Lab Masca MetaMask Snap:
  https://github.com/blockchain-lab-um/masca

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

- `60865a14936351a18ac0b8db06824f1c3038ddc3`

For the verification, we examined the Git revision:

- `a9cce8e50db942744799e0d9396d69703f5ed37f`

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

- Blockchain Lab Masca MetaMask Snap:
  https://github.com/LeastAuthority/blockchain-lab-masca-snap

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:
- Project website:
  https://www.masca.io

- Company website:
  https://blockchain-lab.um.si/?lang=en
- Introducing Masca (prev. SSI Snap):
  https://medium.com/@blockchainlabum/introducing-masca-prev-ssi-snap-ba86023ec8
- Masca Docs:
  https://www.docs.masca.io

In addition, this audit report references the following documents:

- A. Biryukov, D. Khovratovich, and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function." *Crypto Forum Research Group*, 2020, [BKJ20]
- `argon2-browser`:
  https://github.com/antelle/argon2-browser
- PBKDF2:
  https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2

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 the 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 conducted a comprehensive security review of Masca, a MetaMask extension that aims to integrate decentralized identity management, specifically Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs), into the MetaMask ecosystem. We investigated the coded implementation to identify security vulnerabilities and implementation errors and reviewed Masca's utilization of the MetaMask security framework and adherence to the security best practices.

Our team found that security has been taken into consideration in Masca's current implementation and design. However, we identified several issues and suggestions that would improve the overall security of the implementation.

System Design

Our team found that zero-fill Buffers are used to hold secrets, whereas best practice recommends zero-filling buffers containing sensitive information after usage (Issue A). Additionally, we identified a number of Issues relating to encryption and the handling of keys (Issue B, Issue D).

We also found an instance of an incorrect implementation of force boolean, whereby users could perform critical actions without being warned (Issue C). Furthermore, we reviewed input validation and error handling and identified several areas of improvement (Suggestion 1, Suggestion 2).

Code Quality

We performed a manual review of the repositories in scope and found the codebases to be generally organized and well-written.

Tests

Our team found sufficient test coverage of the Masca MetaMask Snap has been implemented.

Documentation

The project documentation provided for this review provides a sufficient overview of the system and its intended behavior.

Code Comments

We found that the implementation is sparsely but sufficiently commented, with code comments generally describing the intended behavior of security-critical components and functions.

Scope

The scope of this review was sufficient and included all security-critical components. However, our findings show that the Masca MetaMask Snap is still in early stages of implementation and that some of the libraries and services implemented in the Snap are in an experimental phase as well.

Dependencies

We examined all the dependencies implemented in the codebase and identified some instances of vulnerable dependencies and several instances of unused dependencies and devDependencies. We recommend improving dependency management (Suggestion 4).

Specific Issues & Suggestions

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

ISSUE / SUGGESTION	STATUS
Issue A: Zero-fill Buffers Are Used To Hold Secrets	Resolved
Issue B: The Crypto Subtle Key Is Marked as Extractable	Resolved
Issue C: Users Can Still Execute Security-Critical Actions Without Being Prompted	Resolved
Issue D: The Key Utilized for the Encryption Does Not Meet Security Best Practice Requirements	Partially Resolved
Suggestion 1: Improve Error Handling and Reporting	Resolved

Suggestion 2: Sanitize, Filter, or Escape Before Validating	Resolved
Suggestion 3: Host Instance of Universal Resolver	Resolved
Suggestion 4: Update and Maintain Dependencies	Resolved
Suggestion 5: Avoid Using console.log	Resolved

Issue A: Zero-fill Buffers Are Used To Hold Secrets

Location

`snap/src/Encryption.service.ts#L19`

`snap/src/Encryption.service.ts#L57`

Synopsis

When using the Buffer number constructor, memory space is reserved without initializing it with zeroes. Instead, the allocated buffer retains whatever data was present in memory at that moment.

Impact

This Issue could result in the leakage of sensitive data.

Technical Details

The content of the newly created `Buffer` will be the residual of whatever value was held by that memory space before, if not zero-filled.

Remediation

We recommend always zero-filling `Buffers` that contain sensitive information or secrets after usage.

Status

The Blockchain Lab team has implemented the recommended remediation.

Verification

Resolved.

Issue B: The Crypto Subtle Key Is Marked as Extractable

Location

`snap/src/Encryption.service.ts#L24C7-L24C11`

`snap/src/Encryption.service.ts#L58`

Synopsis

The flag that allows the keys to be exported is set to `true`. As a result, the crypto subtle key is marked as extractable. Our team did not observe any use case that may call for exporting the CryptoKey. Therefore, allowing it to be exported unnecessarily opens a new attack vector.

Impact
This Issue can result in the leakage of sensitive data and secrets.

Remediation
We recommend turning the key into an unextractable to reduce the attack vector here and here.

Status
The Blockchain Lab team has implemented the recommended remediation.

Verification
Resolved.

Issue C: Users Can Still Execute Security-Critical Actions Without Being Prompted

Location
`snap/src/UI.service.ts#L50`

`snap/src/UI.service.ts#L29`

Synopsis
The implementation uses force boolean to prevent users from circumventing displayed pop-ups for even friendly applications selected by the user. However, it is not used in any of the `snapConfirm` or `snapAlert` calls.

Impact
This Issue can lead to users executing security-critical actions without being prompted.

Remediation
We recommend that force booleans be used for critical actions, such as the `state` export.

Status
The Blockchain Lab team has updated the implementation to force a notification display and obtain user consent even when notifications are turned off by the user for friendly dApps.

Verification
Resolved.

Issue D: The Key Utilized for the Encryption Does Not Meet Security Best Practice Requirements

Location
`snap/src/Encryption.service.ts#L20`

Synopsis
Our team found that the entropy provided by MetaMask's `snap_getentropy` is being used directly as a secret value for an `AES-GCM` encryption in which the key derivation function is not utilized.

A key derivation function turns any source of entropy into a key in a deterministic and secure manner and increases resistance to a variety of password-cracking attacks, such as brute-force attacks. Using a key derivation function (KDF) is generally recommended for better security and key management.

Impact

The strength of the encryption algorithm could be compromised, making the system more susceptible to password cracking attacks, such as brute-force attacks.

Remediation

For deriving a key from the entropy, we recommend utilizing the Argon2id function library implemented with a memory parameter of 64 MB, and following the recommendations explained in [BKJ20]. Additionally, an iteration count (called `OPSLIMIT` by sodium) of 3 should be used. It is important to note that a longer processing time provides better protection against brute-force attacks.

Mitigation

If the Blockchain Lab team is unable to use Argon2 then, as a last resort, we recommend using PBKDF2 to derive secret values from user-supplied passwords, in accordance with the OWASP recommendations.

Status

The Blockchain Lab team stated that they faced some issues when integrating Argon2id into their systems and therefore chose to use PBKDF2. Additionally, the team increased the iterations by six times from the default of 100K.
However, our team recommends that the Blockchain Lab team continue to monitor the security of PBKDF2 and switch to stronger KDF, as described in the Remediation section.

Verification

Partially Resolved.

Suggestions

Suggestion 1: Improve Error Handling and Reporting

Location

`snap/src/UniversalResolver.service.ts#L29`

Synopsis

In the aforementioned location, errors do not provide any useful message. Error handling can be improved in order to provide user-friendly feedback, further aid developers in debugging possible issues, and enhance code maintainability and quality.

Mitigation

We recommend implementing sufficient error handling, such that errors are handled consistently and useful information is provided to help users resolve errors.

Status

The Blockchain Lab team has implemented the recommended remediation.

Verification

Resolved.

Suggestion 2: Sanitize, Filter, or Escape Before Validating

Location

[snap/src/UniversalResolver.service.ts#L22C25-L22C33]

Synopsis

Our team found that the code sends a fetch request to an external server. If this server is compromised, it could return malicious data.

Mitigation

We recommend sanitizing, filtering, or escaping before validating the response from external sources. Also, `type` assertion may not guarantee safe casting into a `DIDResolutionResult` object as opposed to mapping the response to an `interface` type.

We also recommend adding a request timeout to prevent extended freeze time in case of an unresponsive server.

Status

The Blockchain Lab team has implemented the recommended remediation.

Status

Resolved.

Suggestion 3: Host Instance of Universal Resolver

Location

snap/src/UniversalResolver.service.ts#L20

Synopsis

The `UniversalResolver` service relies on an external DID resolver, which is not meant for production purposes and is subject to change.

Impact

If the `Universal Resolver` makes malicious or incompatible changes to the DID resolution process, the DID resolution could lead to arbitrary results, including resolving a malicious DID document.

Mitigation

We recommend that the Blockchain Lab team host their own instance of the `Universal Resolver`.

Status

The Blockchain Lab team has implemented the recommended remediation.

Verification

Resolved.

Suggestion 4: Update and Maintain Dependencies

Location

packages/snap/package.json

Synopsis
Analyzing `package.json` for dependency versions using `pnpm audit` shows that the dependencies used in the Masca MetaMask Snap have 5 reported known vulnerabilities (2 Moderate, 2 High, 1 Critical). Additionally, the `npx depcheck` tool reported several unused dependencies and devDependencies.

Impact
Using unmaintained dependencies and devDependencies or packages with known vulnerabilities may lead to critical security vulnerabilities in the codebase.

Mitigation
We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities to the Masca MetaMask Snap and to mitigate supply chain attacks, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Pinning dependencies to secure versions when upgrading vulnerable dependencies to secure ones, including pinning build-level dependencies in the `package.json` file to a specific version;
- Only upgrading dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities; and
- Including Automated Dependency auditing reports in the project's CI/CD workflow.

Status
The Blockchain Lab team has implemented the recommended remediation.

Verification
Resolved.

Suggestion 5: Avoid Using console.log

Location
Examples (non-exhaustive):

`src/veramo/Veramo.service.ts#L579`

`src/veramo/Veramo.service.ts#L682`

`src/veramo/Veramo.service.ts#L747`

`src/polygon-id/Polygon.service.ts#L354`

`src/polygon-id/Polygon.service.ts#L388`

Synopsis
Using `console.log` in a production environment can lead to the leakage of sensitive information like user data, which can cause significant security risk.

Mitigation
We recommend refraining from using `console.log` in a production environment.

Status
The Blockchain Lab team has implemented the recommended remediation.

Verification

Resolved.

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.