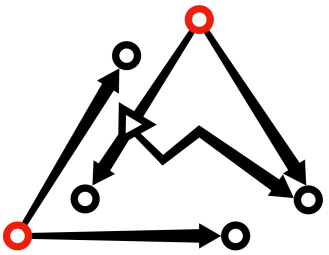

Least Authority

PRIVACY MATTERS

MetaMask Snap

Security Audit Report

Generative Labs

Final Audit Report: 30 August 2023

Table of Contents

Overview

Background

Generative Labs has requested that Least Authority perform a security audit of their MetaMask Snap.

Project Dates

- **August 16, 2023 - August 18, 2023:** Initial Code Review *(Completed)*
- **August 22, 2023:** Delivery of Initial Audit Report *(Completed)*
- **August 29, 2023:** Verification Review *(Completed)*
- **August 30, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Mukesh Jaiswal, 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:

- Web3MQ Snap:
  https://github.com/Generative-Labs/Web3MQ-Snap

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

- `b12e6ec9cac02d7d98e63f41b33b7d4d1590d02a`

For the verification, we examined the Git revision:

- `41ce45227e064944b089750166d23f99`

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

- Web3MQ Snap:
  https://github.com/LeastAuthority/generative-labs-Web3MQ-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:

- Website:
  https://www.generativelabs.co
- Intro | Web3MQ Documentation:
  https://docs.web3mq.com/docs/Intro

In addition, this audit report references the following documents:

- M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC." *NIST Special Publication 800-38D*, 2007, [Dworkin07]
- PBKDF2: https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/deriveKey#pbkdf2
- `escape()`: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/escape
- `encodeURIComponent()`: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent
- `encodeURI`: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeUR
- `snap_getEntropy` https://docs.metamask.io/snaps/reference/rpc-api/#snap_getentropy
- `zxcvbn`: https://github.com/dropbox/zxcvbn
- `SubtleCrypto`: https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the Snap implementation;
- Potential misuse and gaming of the Snap;
- 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 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 security review of the Web3MQ Snap, a client interface that allows its users to interact with the Web3MQ social messaging network.

We investigated the coded implementation to identify security vulnerabilities and implementation errors and reviewed Web3MQ's utilization of the MetaMask security framework and adherence to the security best practices. In our review, we investigated the handling of sensitive data, secure use of libraries, input validation, as well as the use of permissions.

We identified several issues and suggestions that would improve the quality of the code and contribute to the overall security of the implementation, as detailed below.

System Design

During our review, we found a series of issues related to the use of cryptography and key generation. Security guarantees offered by the implemented encryption function `AES-GCM` are dependent on the entropy of the password, the key derivation function (KDF) used, and the quality of the randomness of the Initialization Vector (IV). Our team found that HKDF, the key derivation function based on the hash-based message authentication code (HMAC), is used to derive keys, which is not a sufficiently secure algorithm for the purpose of deriving keys from user-selected passwords (Issue A). Furthermore, the initialization factor used is not unique, and we recommend using a cryptographically strong pseudorandom number generator (PRNG) instead (Issue C). Our team also found that there are no constraints on passwords that a user may select, which would allow users to select a low entropy password (Suggestion 7). Additionally, we investigated the libraries used and found that non-standard libraries are implemented to handle sensitive functionalities (Issue B).

Code Quality

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

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 2).

Documentation

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

Code Comments

There are insufficient code comments describing security-critical components and functions in the codebase. We recommend improving code comments (Suggestion 6).

Scope

The scope of this review was sufficient and included all security-critical components.

Dependencies

We examined all the dependencies implemented in the codebase and identified several instances of unused dependencies and devDependencies. We recommend improving dependency management (Issue D).

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: Weak Key Derivation Algorithm Used	Resolved
Issue B: Unnecessary Usage of Non-Standard Libraries	Unresolved

Issue C: Initialization Vector Used Does Not Meet Recommended Best Practices	Unresolved
Issue D: Vulnerable and Unused Dependencies Detected in the Codebase	Resolved
Suggestion 1: Replace Deprecated escape Function	Partially Resolved
Suggestion 2: Implement a Test Suite	Unresolved
Suggestion 3: Use the External Time Server	Unresolved
Suggestion 4: Improve Error Handling	Resolved
Suggestion 5: Avoid Using console.log	Resolved
Suggestion 6: Improve Code Comments	Unresolved
Suggestion 7: Do Not Allow Low Entropy Passwords	Unresolved
Suggestion 8: Consider Using the Entropy Provided by MetaMask	Unresolved

Issue A: Weak Key Derivation Algorithm Used

Location

`snap/src/encryption/index.ts#L84`

Synopsis

In the `GetAESBase64Key` function, the password is used as input for keydata in the `crypto.subtel.importKey` function, which can result in weak key generation.

Impact

In the `GetAESBase64Key` function, `crypto.subtel.importKey(format, keyData, algorithm, extractable, keyUsage)` uses HKDF as an algorithm for key derivation. However, since HKDF is not designed for low entropy input, the derived master key will not be secure enough.

Remediation

We recommend using PBKDF2, instead of HKDF, as it is specifically designed for deriving keys from passwords.

Status

The Generative Labs team has implemented the remediation as recommended.

Verification

Resolved.

Issue B: Unnecessary Usage of Non-Standard Libraries

Location

snap/src/utils.ts#L31

snap/src/utils.ts#L270

Synopsis

While the libraries used do not appear to be insecure, there are standard libraries available that should be used instead.

Impact

The use of non-standard libraries to handle sensitive functionalities introduces the potential for implementation errors and increases the possibility of a supply chain attack that may result in serious vulnerabilities.

Remediation

We recommend the use of standardized, trusted, and audited alternatives, such as `SubtleCrypto`.

Additionally, for the cases where the implementation uses SHA3-224, although it is not implemented in any of the WebAPI libraries to the best of our knowledge, SHA256 can be utilized instead, and the output hash can be trimmed with negligible impact on performance.

Status

The Generative Labs team responded that the npm package @noble/ed25519 is both recommended and user-friendly. While our team agrees with this statement, this Issue was identified in order to suggest alternatives for the usage of libraries, such as `js-sha224` and `js-sha3`. Our team continues to recommend the usage of SHA256 instead of SHA3-224.

Verification

Unresolved.

Issue C: Initialization Vector Used Does Not Meet Recommended Best Practices

Location

src/register/index.ts#L257

Synopsis

A key is a piece of information used in ciphering data and must remain secret, while an Initialization Vector (IV) is used to ensure that the same plaintext encrypted with the same key results in different ciphertexts, thus providing semantic security. In addition, in specific cases, the IV can be public and retaining uniqueness can be sufficient, according to [Dworkin07]. However, the current implementation uses parts of the derived secret key as an IV. Mixing the roles of the aforementioned two inputs can jeopardize the strength of the symmetric encryption algorithm used.

Impact

The purpose of an IV is to add randomness to the encryption process. If IVs are not unique, the security of the encryption is weakened. Attackers might be able to predict or manipulate the ciphertext, leading to potential data leakage.

Remediation

We recommend using a pseudorandom number generator (PRNG) to generate an IV instead of reusing parts of the secret key.

Status

The Generative Labs team stated that decoding and encoding require consistent IV parameters, so the user's password is used as the original parameter for SHA256.
However, our team found that the IV is still being derived from the password using SHA256 and is then converted to a Base64 string. While this approach is better than using a plaintext password as an IV, it is still not ideal. As noted in the synopsis of this Issue, it is generally recommended to use a randomly generated IV for each encryption operation, and to ensure that there is no overlap between the secret key and the IV (which can be public or safely stored in plaintext format).

Verification

Unresolved.

Issue D: Vulnerable and Unused Dependencies Detected in the Codebase

Synopsis

Analyzing the code using `npm audit` and `depcheck` shows that the dependencies used have 10 known vulnerabilities (1 Critical, 9 Moderate). According to `npx depcheck`, the following unused dependencies and devDependencies were identified:

Unused dependencies:

- `@metamask/snaps-ui`
- `@protobuf-ts/plugin`
- `buffer`
- `web3`

Unused devDependencies:

- `@lavamoat/allow-scripts`
- `@metamask/auto-changelog`
- `prettier-plugin-packagejson`

Impact

Using unmaintained and outdated dependencies and devDependencies may lead to critical security vulnerabilities in the codebase.

Remediation

We recommend proper management and maintenance of dependencies and devDependencies, as detailed below:

- Manually auditing and updating dependencies, in order to avoid known issues in unmaintained and outdated dependencies and conducting extensive testing to confirm there are no backward compatibility issues introduced by upgrading dependencies;
- Including the automated dependency auditing into the CI workflow or enabling Dependabot on GitHub, which automatically notifies developers about published security advisories relevant to the codebase;
- Acting on published advisories and updating dependencies accordingly when fixes are released; and

- Pinning specific dependency versions to mitigate any potential supply chain attacks or the possibility of newer versions of dependencies breaking the code.

Status

The Generative Labs team has implemented the remediation as recommended.

Verification

Resolved.

Suggestions

Suggestion 1: Replace Deprecated escape Function

Location

`snap/src/utils.ts#L127`

Synopsis

The `escape` function has been deprecated and should be replaced.

Mitigation

We recommend using `encodeURIComponent` or `encodeURI` instead.

Status

The Generative Labs team has removed the deprecated function but does not verify the result of the `window.atob` function.

Verification

Partially Resolved.

Suggestion 2: Implement a Test Suite

Synopsis

Our team found no tests in the repository in scope. Sufficient test coverage should include tests for success and failure cases, which helps identify potential edge cases, and protect against errors and bugs that may lead to vulnerabilities or exploits. 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 for the Snap implementation to facilitate identifying implementation errors and potential security vulnerabilities by developers and security researchers.

Status

The Generative Labs team stated that they plan on improving test coverage in the future. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 3: Use the External Time Server

Location

snap/src/register/index.ts#L150

Synopsis

The JavaScript `Date` object fetches the date and time from the host system, which can be changed by changing the time in the host machine.

Mitigation

We recommend fetching the date and time from the external time server instead.

Status

The Generative Labs team stated that the developer can regenerate a new temporary key pair by encrypting the private key, and the `Snap-SDK` simply sets a default value when the developer does not enter this parameter.

Additionally, the team noted that clients are generally incentivized to use whatever timeout value suits them, and the length of the session key is determined by the frontend. Therefore, the team believes that relying on the backend time would not resolve the suggestion. Additionally, they noted that there is not much incentive and gain resulting from the use of an incorrect time because if the frontend client wanted to change length of timeout time, they could always do so by passing in a different and desired honest time.

Verification

Unresolved.

Suggestion 4: Improve Error Handling

Location

Examples (non-exhaustive):

snap/src/utils.ts#L75

snap/src/utils.ts#L120

Synopsis

If the call to the endpoint fails in the aforementioned locations, it could result in unexpected behavior. Error handling can be improved in similar cases in order to provide user-friendly feedback, further aid developers in debugging possible issues, and enhance code maintainability and quality.

Mitigation

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

Status

The Generative Labs team stated that only nodes with successful requests are added to the selected list. Hence, when all nodes fail, an error is thrown indicating that the user's network connection has failed.

Verification

Resolved.

Suggestion 5: Avoid Using console.log

Location

Examples (non-exhaustive):

snap/src/utils.ts#L104

src/register/index.ts#L194

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 Generative Labs team has removed `console.log` from the production environment.

Verification

Resolved.

Suggestion 6: Improve Code Comments

Synopsis

Currently, the codebase lacks explanation in areas that handle sensitive functionalities, such as the use of cryptography and authentication. In addition, the `utils` package is also insufficiently commented. This reduces the readability of the code and, as a result, makes reasoning about the security of the system more difficult. Comprehensive in-line documentation explaining, for example, expected function behavior and usage, input arguments, variables, and code branches can greatly benefit the readability, maintainability, and auditability of the codebase. This allows both maintainers and reviewers of the codebase to comprehensively understand the intended functionality of the implementation and system design, which increases the likelihood for identifying potential errors that may lead to security vulnerabilities.

Mitigation

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

Status

At the time of the verification, the suggested mitigation has not been resolved.

Verification

Unresolved.

Suggestion 7: Do Not Allow Low Entropy Passwords

Location

`src/register/index.ts#L256C44-L256C44`

Synopsis

Encryption keys that are derived from weak passwords are susceptible to dictionary attacks.

Remediation

We recommend using the `zxcvbn` library for a password composition policy intended to prevent the use of passwords obtained from previous breaches, common passwords, and passwords containing repetitive or sequential characters, as recommended by the NIST guidelines. We suggest requiring that all passwords meet `zxcvbn`'s strength rating of 4.

Status

The Generative Labs team stated that they prefer to delegate this to the frontend application instead of implementing a hard requirement at the protocol layer. They further noted that they might add some user interface feedback for password strength to improve communication.

Verification

Unresolved.

Suggestion 8: Consider Using the Entropy Provided by MetaMask

Location

`src/register/index.ts#L224-L244`

Synopsis

The current implementation derives the main key pair from the output obtained after signing the hash output of a message with a user-specified password. This can be replaced with an improved flow that better achieves account coupling with an Ethereum account while being locked with a user-specified password.

Remediation

We recommend using the snap-specific, deterministic 256-bit entropy value provided by Metamask, which can be used to generate a private key.

Status

The Generative Labs team stated that although the entropy provided by MetaMask would theoretically solve their business needs, since Web3MQ accounts need to be interoperable across multiple clients and not just Snaps, they cannot implement this method outside of Snaps. Therefore, this suggestion conflicts with the broader Web3MQ ecosystem.

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.