Cryptonics

Morpher Zero Wallet - Audit Report

February 24, 2021

Cryptonics Consulting S.L.
Ramiro de Maeztu 7
46022 Valencia
SPAIN

https://cryptonics.consulting/

Table of Contents

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHORS AND THEIR EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS NOT A SECURITY WARRANTY, INVESTMENT ADVICE, OR AN ENDORSEMENT OF THE CLIENT OR ITS PRODUCTS. THIS AUDIT DOES NOT PROVIDE A SECURITY OR CORRECTNESS GUARANTEE OF THE AUDITED SOFTWARE.

Introduction

Purpose of this Report

Cryptonics Consulting has been engaged by Morpher Labs GmbH to perform a security audit of the Zero Wallet recoverable keystore. The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine implementation bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the code submitted in the following GitHub repository:

https://github.com/Morpher-io/zerowallet-recoverable

Commit no: `5dd7631e82b4a65002e08907231f15c261dfae07`

Updates were received on 24 February and the latest commit number covered by the fx review is:

`96b872497cc50ed0ff08517dc329b026004973a2`

Methodology

The audit has been performed by a team of full-stack auditors and cryptography consultants.

The following steps were performed:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. API security
    b. Cryptographic primitives
    c. Key management vulnerabilities
    d. Permission issues
    e. Logic errors
4. Report preparation

The results were then discussed between the auditors in a consensus meeting and integrated into this joint report.

Functionality Overview

The submitted code implements a browser-based wallet. The wallet executes in the context of the front-end. The encrypted seed phrase is stored in local storage and in the backend. A social media recovery option allows users to create a version of their encrypted seed phrase that can be recovered using an OAuth login flow with social media accounts as a challenge.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentatio**n, and **test coverage**. We include a table with these criteria for each module, in the corresponding findings section.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

The Morpher Wallet codebase was found to contain no critical issues, 1 major issue, 4 minor issues and 1 informational note:

No	Description	Severity	Status
1	backend-node/helpers/funcitons/utils.ts: Key size mismatch in AES encryption	Major	Resolved
2	backend-node/helpers/funcitons/utils.ts: Backend encryption uses SHA-256 for key derivation	Minor	Resolved
3	Email 2FA codes do not expire	Minor	Resolved
4	Outdated dependencies in Backend	Minor	Resolved
5	Outdated dependencies in SDK	Minor	Resolved
6	Consider using GCM mode for server-side encryption	Informational	Resolved

Code Quality Criteria:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	Medium-high	-

Notes on Architecture / Functionality

Apart from the code implementation security review, the audit team has performed a review of the currently supported functionality and architecture. The following a number of informal observations not directly related to any issue in particular:

Social Recovery Feature Implications and Terminology

The architecture allows users to recover their seed phrases through a social media login operation. This, of course, a **voluntary trade-off between usability and security**.
The security implications are as follows:

- The Morpher team can gain access to the key if the social media user id of the user is discovered. Whilst encryption is done client-side, there are other ways to obtain a user-id.
- The encryption keyspace is reduced to the way the social media platform generates user-id. This might reduce the number of possible encryption key options and facilitate a brute force attack.

Whilst such trade-offs are unavoidable, users should be made aware of them.

Furthermore, the term *social recovery wallet* is generally associated with a different concept, more akin to multi-signature wallets (e.g. https://vitalik.ca/general/2021/01/11/recovery.html).

Temporary Password Storage

The encryption password is temporarily stored client-side. Again, this is a trade-off in favor of enhanced user experience. This design decision slightly reduces the security of the solution, in the case of local storage compromise, but is a common UX solution.

Notes on Cryptographic Protocols and Parameters

Given that a wallet application relies heavily on cryptography for secure key management, both in terms of key generation and storage, we have performed an analysis of the cryptographic protocols and the parameters used.

Client-Side Key Generation

A Javascript BIP39 library (https://github.com/bitcoinjs/bip39) is used to generate mnemonic phrases. The library is used in its standard configuration with an underlying entropy of 128 bit.

`ethereumjs-wallet` (https://github.com/ethereumjs/ethereumjs-wallet) is used to convert the mnemonics into seeds to generate HD wallets using derivation path `m/44'/60'/0'/0/`.

Mnenomics are stored encrypted using symmetric encryption (see below).

Client-Side Encryption

Data stored client-side is encrypted using AES encryption with password-derived 256-bit keys. The following parameters are used:

- GCM mode
- 128-bit random initialization vector generated with `window.crypto.getRandomValues()`
- Password derived keys using `PBKDF2` with the following setup:
  - `SHA256` hash function
  - 100,000 iterations
  - 96-bit random salt

The above client-side cryptographic parameters are generally considered secure and follow cryptographic best practice recommendations.

Server-Signed Encryption

Server-side encryption is performed using AES encryption with a single 256 key derived from an environment variable. The following parameters are used:

- CBC mode with PKCS#7 padding (https://tools.ietf.org/html/rfc5652#section-6.3)
- 128-bit random initialization vector generated with built-in Node.js crypto library

The encryption key is always derived as the top 128-bits of the SHA-256 of an environment variable. This environment variable is called `DB_BACKEND_SALT`, even though in this case its use is more akin to that of a password used to derive a key.

The absence of a secure key derivation function, such as that used client-side renders server-side encryption less secure. Furthermore, the keyspace is reduced to 128 bits (see security issues below). However, this risk is mitigated by the data already having been encrypted client-side.

UPDATE: The server-side encryption has been adapted to the suggestions outlined above.

Signature-based Authentication

Authenticated API routes require a signature to be passed in the request header. The body of the request acts as the message to be signed.

The server-side code resolves the signature and compares the Ethereum address to that stored with each user id in the database. A nonce is used to provide replay protection.

Detailed Findings

1. `backend-node/helpers/funcitons/utils.ts`: Key size mismatch in AES encryption

Severity: Major

Server-side encryption only uses the top 16 bytes of the derived value as an encryption key:

```
const key = sha256(secret).substr(0, 32);
```

The above code derives the key as the top 32 characters of a hex string, meaning the actual keyspace is reduced to 128 bits, even though AES-256 is used.

Recommendation

Use 256 keys.

Status: Resolved

2. `backend-node/helpers/functions/utils.ts`: Backend encryption uses SHA-256 for key derivation

Severity: Minor

The functions `encrypt()` and `decrypt()` derive the encryption key hashing the environment variable `DB_BACKEND_SALT`. SHA-256 on its own is not a secure key derivation function. This issue is marked as minor since the encrypted data has already been encrypted client-side.

Recommendation

Consider using `PBKDF2` as in the client-side encryption, or forego key derivation, using a global 256-bit key from the environment (in case of performance concerns with PBKDF2.

Status: Resolved

3. Email 2FA codes do not expire

Severity: Minor

Email 2FA codes are generated in `backend-node/controllers/wallet.controller.ts`, function

`updateEmail2fa()`. However, these codes remain valid indefinitely. It is generally considered good practice to limit the time between first- and second-factor authentication.

Recommendation

Consider adding an expiry time for email 2FA codes to the data model and verifying this.

Status: Resolved

4. Outdated dependencies in Backend

Severity: Minor

Outdated versions for the `ethereumjs-tx`, `ethereumjs-util` and `web3-eth-accounts` npm packages are used. These modules introduce a nested dependency to an insecure version of the `elliptic` cryptography library.

Recommendation

Update outdated dependencies.

Status: Resolved

5. Outdated dependencies in SDK

Severity: Minor

Outdated versions for the `web3` and `web3-provider-engine` npm packages are used. Newer versions of these two modules introduce minor security fixes of nested dependencies.

Recommendation

Update outdated dependencies.

Status: Resolved

6. Consider using GCM mode for server-side encryption

Severity: Informational

The backend uses CBC mode with PKCS#7 padding for 256-bit AES encryption. CBC mode is generally considered less secure because of its inherent vulnerability to oracle padding attacks. Whilst this type of attack should not be an issue in this particular application, GCM is generally recommended as a more secure option.

Recommendation

Consider using GCM mode.

Status: Resolved