Least Authority
PRIVACY MATTERS

Clorio Wallet + Mina Ledger JS

Security Audit Report

Mina Foundation

Final Audit Report: 30 June 2021

Table of Contents

Overview

Background

Mina Foundation requested that Least Authority perform a security audit of the Clorio wallet and Mina Ledger JS. The Clorio wallet consists of the Clorio client implementation, a wallet client built on top of the Mina Protocol, and the Clorio server assistant, a GraphQL wrapper for the Clorio client to interact with a Mina Protocol node, using Hasura as a proxy. Mina Ledger JS is a client library that communicates with `ledger-app-mina` running on the Ledger Nano S/X.

Project Dates

- **April 26 - May 14**: Code review *(Completed)*
- **May 19**: Delivery of Initial Audit Report *(Completed)*
- **June 25 - 29:** Verification *(Completed)*
- **June 30:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Ramakrishnan Muthukrishnan, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Clorio Wallet and Mina Ledger JS followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Clorio Wallet
  - Clorio Client: https://github.com/nerdvibe/clorio-client
  - Clorio Server Assistant: https://github.com/nerdvibe/clorio-server-assistant
- Mina Ledger JS: https://github.com/nerdvibe/mina-ledger-js

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

clorio-client: `eb412a6c0c01625ff82888535fa187b771369849`

clorio-server-assistant: `8d6d1e6badbe8da1cb8cc5f3849d3668fab98576`

mina-ledger-js: `98f4258dba73c9f75d7b6d17ac91bf276b28ef7e`

For the verification, we examined the Git revision:

Clorio-client: `a2445f2f0699dc9eae729a02e0fece98cc386ed3`

For the review, these repositories were cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/clorio-client

https://github.com/LeastAuthority/clorio-server-assistant https://github.com/LeastAuthority/mina-ledger-js

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:

- Mina Protocol Documentation: https://minaprotocol.com/docs
- Clorio Wallet
    - Clorio Wallet User Documentation: https://docs.clor.io/
    - Clorio Client README.md: https://github.com/nerdvibe/clorio-client/blob/main/README.md
    - Clorio Server Assistant README.md: https://github.com/nerdvibe/clorio-server-assistant
    - Clorio Technical Documentation: Clorio Tech Docs rev1.pdf shared with Least Authority via Discord on 26 April 2021
- Mina Ledger JS README.md: https://github.com/nerdvibe/mina-ledger-js/blob/main/README.md

In addition, this audit report references the following documents:

- Electron App documentation: https://www.electronjs.org/docs

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and its adherence to the specification;
- Common and case-specific implementation errors;
- Vulnerabilities in the code, as well as secure interaction between the client, the library, and other network components;
- Key management: secure private key storage and proper management of encryption and signing keys;
- Attacks that impacts funds, such as draining or manipulating of funds;
- Mismanagement of funds via transactions;
- Secure communication between the nodes;
- Denial of Service (DoS) attacks and other methods of exploitation;
- Adversarial actions and protection against malicious attacks on the client and the library;
- Exposure of any critical information during interactions with users and external libraries;
- 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

Clorio is an open source, full-stack wallet for the Mina Protocol. Our security review focused on three core system components: the Clorio wallet client implementation, the Clorio wallet's server assistant implementation for hosted use cases, and the Mina Ledger JS implementation that facilitates client communication with the Mina Ledger hardware wallet for signing transactions.

System Design

Through comprehensive review of the Clorio wallet and Mina Ledger JS, it is apparent that security has been considered in the respective system design and implementations. This is particularly demonstrated through the general adherence to development and documentation best practices, and the implementations' adherence to the documented system design. In addition, delegating the application of cryptography to the utilization of the Ledger hardware wallet and not storing secrets locally in the browser by the Clorio wallet client demonstrates diligence in the system's design.

However, we identified several areas of improvement that would contribute to the overall security of the system, as detailed below.

Cryptographic Functions

The Clorio wallet delegates cryptographic functions such as signing transactions to a hardware wallet (i.e. Ledger Nano S/X) such that secrets are not stored locally. However, for the non-Ledger hardware wallet use case, Clorio utilizes the Coda Client Javascript SDK library for key pair generation, which generates key pairs non-deterministically. We recommend always delegating key pair generation to the Mina Ledger hardware wallet (Issue A).

In addition, we found that when generating a keypair, the user is provided with a PDF copy of the public and private keys, as well as given the option to copy the keys into the operating system (OS) clipboard. OS clipboards and download folders are not secure storage environments for secret keys, and can put the user's private keys at risk of compromise. As a result, we recommend that private key storage be delegated to a hardware wallet or, alternatively, the user can be prompted to write down their private keys instead (Issue A).

Electron Application

The Clorio client is an Electron application implementation of the wallet, which has not been optimized for security according to the security recommendations provided by Electron. A sub-optimal security configuration increases the attack surface and may potentially lead to full remote system compromise. In order to reduce the risk of a security vulnerability, we recommend adhering to the Electron application best practice recommendations (Issue B, Issue C; Issue D; Suggestion 2).

Furthermore, our team conducted an Electronegativity test on the Clorio client implementation in order to identify misconfigurations and security anti-patterns in Electron-based applications. We identified several results warranting further investigation. We recommend that the Clorio team perform an Electronegativity test on the Electron application implementation of the Clorio wallet and thoroughly investigate and address each of the issues returned (Suggestion 3).

Browser-Related Security

We did not identify any vulnerabilities in the Clorio server assistant, however, we discovered that additional security risk is introduced by using a browser client to interact with the hosted version of the Clorio wallet, which increases the attack surface. In particular, the browser client is prone to clickjacking attacks or a malicious browser extension may read or have control over users' secrets. As a result, we recommend that users are encouraged to use the Electron version of the wallet where possible (Suggestion 1).

Automatic Update and Verification

Building the Clorio wallet from the binary requires a manual step to verify the checksum of the downloaded executable, as opposed to automating this process so it does not rely solely on the user. As a result, there is a high probability that this step is skipped, making users vulnerable to using a malicious version of the wallet that could compromise private keys. Thus, we recommend an automated process to verify the build and code signature of the binary (Issue E).

Code Quality

The Clorio wallet and Mina Ledger JS codebases are well organized and generally adhere to development best practices. The code is well commented, which facilitates easy understanding of the intended functionality for each component.

However, all system components are missing tests, which serve as an important layer of security to check if the system is functioning as intended. A robust test suite should include all success and failure cases, which demonstrates that the implementation behaves as intended, and helps identify potential edge cases and protects against errors and bugs. We recommend implementing a test suite, including unit tests and end-to-end tests for the Clorio client, Clorio server assistant, and for Mina Ledger JS (Suggestion 4).

Documentation

We found that the Clorio documentation was accurate and helpful in explaining the general functionality and purpose of each of the components. The instructions provided to set up a Mina node and run a wallet locally are clear and concise.

Scope

The scope of the security review was generally sufficient. However, we noted that the cryptographic functionality of the protocol is carried out by the dependency Coda Client Javascript SDK, which was not in the scope of this security review. Given that this library is a dependency that performs a security critical function, we strongly recommend it undergo a third party security review (Suggestion 5).

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: Secrets Stored in Insecure Medium	Resolved
Issue B: No Content Security Policy in Clorio Electron Application	Resolved
Issue C: Node Integration Enabled in Clorio Electron Application	Resolved
Issue D: WebSecurity Sandbox Not Set in Clorio Electron Application	Resolved
Issue E: Application Lacks Automatic Update and Verification	Resolved
Suggestion 1: Review Offered Versions of Clorio	Resolved
Suggestion 2: Comply with Electron's Security Recommendations	Resolved
Suggestion 3: Investigate Electronegativity Results More Thoroughly	Resolved

Suggestion 4: Implement Unit Tests, End-to-End Tests, and Ledger Test	Unresolved
Suggestion 5: Conduct an Audit of Coda Client Javascript SDK	Unresolved

Issue A: Secrets Stored in Insecure Medium

Location

`UI/registration/RegistrationStep.tsx`

Synopsis

Upon wallet creation, users are given the option to copy the private key into the clipboard or download the wallet information as a PDF file. A clipboard is not a secure medium as it can be read by all running processes on the OS. In addition, it may possibly be read by other pages in the browser.

Impact

The exposure of secrets used to access wallets may lead to the loss of funds.

Preconditions

A malicious process or browser tab is monitoring the clipboard.

Technical Details

The common format of the private and public keys makes it difficult for end users to memorize and store keys in a non-digital format. In Clorio, secrets may be copied to the clipboard upon finishing the registration process. The clipboard is considered a public medium, which can be accessed by almost any program or process running on the user's device without any prior permission or notification. The code uses `document.execCommand()`, which has been deprecated and may be removed in the near future.

Mitigation

We recommend that secrets not be stored on the clipboard or placed in insecure PDF files stored in the plain filesystem. Additionally, we recommend that users be prompted to write down their secrets on a physical medium, instead of using a clipboard or a PDF download.

Remediation

We recommend always delegating key pair generation to the Mina Ledger hardware wallet.

Status

The Clorio team has removed the `copyToClipboard()` function and added a text warning to the user to write down the private key. Additionally, the Clorio team has implemented a BIP-39 mnemonic generator, which allows the user to generate a key pair deterministically using a mnemonic passphrase.

Verification

Resolved.

Issue B: No Content Security Policy in Clorio Electron Application

Location

`/clorio-client/public/electron.js`

Synopsis

Clorio is currently missing a Content Security Policy (CSP). A CSP permits the server that is serving content to restrict and control the resources Electron is able to load for a given web page. This applies to any HTML document that is loaded by Electron.

Impact

Without a CSP in place, requests to arbitrary and untrusted resources will not raise or prevent any errors, which could be exploited to facilitate data exfiltration. Depending on the precondition, the absence of a CSP provides an attacker with additional leverage over any foothold they have.

Preconditions

A dependency, update, or trusted remote code would need to be vulnerable or compromised.

Mitigation

We recommend publishing an update with the most restrictive CSP possible, including restrictive fallbacks that allow the wallet application to function in its current state. Defining a CSP provides a baseline for security practices and serves as a starting point for a remediation.

Remediation

We recommend referencing the information provided on CSPs by the electron security recommendations, and defining the most restrictive CSP possible with restrictive fallbacks. CSP Evaluator can be used to evaluate introduced CPS.

Status

The Clorio team has restricted the fetching of assets such as CSS, scripts, fonts, and others to a limited set of domains by reconfiguring the renderer with CSP data using the node package `csp-html-webpack-plugin`.

Verification

Resolved.

Issue C: Node Integration Enabled in Clorio Electron Application

Location

`/clorio-client/public/electron.js`

Synopsis

Node integration is currently enabled in the renderer process.

Impact

The Electronegativity Document, NODE_INTEGRATION_JS_CHECK outlines the following impact:

> *"If enabled, `nodeIntegration` allows JavaScript to leverage Node.js primitives and modules. This could lead to full remote system compromise if you are rendering untrusted content."*

Preconditions

A dependency, update, or trusted remote code would need to be vulnerable or compromised.

Technical Details

Enabling `NodeIntegration` results in making node APIs available to the renderer process.

Remediation

We recommend disabling node integration and using a tool like Webpack or Browserify to build for a browser-based environment. We also suggest using polyfill or ponyfill libraries, or a combination of both, in order to accommodate differences in the runtime environment where necessary.

Status

The Clorio team has disabled `nodeIntegration` and enabled `contextIsolation`, resulting in `preload.js` and Electron App logic running in separate contexts.

Verification

Resolved.

Issue D: WebSecurity Sandbox Not Set in Clorio Electron Application

Location

`/clorio-client/public/electron.js`

Synopsis

Remote code has unnecessary access to Electron APIs.

Impact

Exposing the Electron renderer API to remote code increases the attack surface and leaks information about the user's system.

Technical Details

The Electronegativity document, `SANDBOX_JS_CHECK` outlines the following impact:

> *"Electron extends the default JavaScript APIs (e.g. window.open returns an instance of BrowserWindowProxy) which leads to a larger attack surface.*
>
> *Instead, sandboxed renderers expose default JavaScript APIs. Additionally, a sandboxed renderer does not have a Node.js environment running (with the exception of preload scripts) and the renderers can only make changes to the system by delegating tasks to the main process via IPC.*
>
> *Even with nodeIntegration disabled, the current implementation of Electron does not completely mitigate all risks introduced by loading untrusted resources. As such, it is recommended to enable `sandbox`."*

Remediation

We recommend enabling sandbox for remote content and using message passing to facilitate any necessary calls to sensitive APIs outside of the sandbox environment.

Status

The Clorio team has set `electron.js`'s `webPreferences` sandbox to true. The result is that the renderer and Electron Application logic run in separate contexts.

Verification
Resolved.

Issue E: Application Lacks Automatic Update and Verification

Location
`/clorio-client/`

Synopsis
The `clorio-client` release page contains manual steps to update and verify the binary. The automation of updates through a well-known single source and verification improves and contributes to better end user security.

Impact
Users may end up running random unverified and unofficial binaries that would compromise the wallet and result in the loss of funds.

Technical Details
The `clorio-client` GitHub release page provides manual steps to download and verify the released binaries for major operating systems. There are several problems with this approach of release, as detailed below.

1. The checksum approach is used by a number of free and open source projects. However, most end users are not developers and may end up skipping this step. Additionally, the release checksums are not signed with a public-key with this approach;
2. The Windows and macOS binaries are not notarized, which would result in a warning when the users invoke the application. This does not provide the user with a sense of security when using the application; and
3. The update process for a new release is manual. A user is not notified when a new release is available and may end up using an outdated, buggy release without knowing that a new release that fixes known bugs is available.

Remediation
We recommend using `electronforge` or `electron-builder`. `Electron-builder` is already being used in the project for building. In addition, the `electron-builder` project provides ways to auto-update and do code-signing.

Status
The Clorio team has implemented code to use the module `electron-updater` to check for new releases from the GitHub release page of the Clorio wallet.

Verification
Resolved.

Suggestions

Suggestion 1: Review Offered Versions of Clorio

Synopsis

The hosted version of the Clorio wallet has a wider attack surface, since it can be prone to clickjacking attacks or a malicious browser extension may read or have control over users' secrets. This is fundamentally due to the security model of browser clients.

Mitigation

We recommend explicitly recommending to users that they utilize the Electron version of Clorio for optimal security.

Status

The Clorio team has added code for a visual prompt that includes a download for the Electron Version of Clorio in the hosted version.

Verification

Resolved.

Suggestion 2: Comply with Electron's Security Recommendations

Location

`docs/tutorial/security#checklist-security-recommendations`

Synopsis

The Clorio wallet Electron application is non-compliant with many recommendations in the Security Recommendations Checklist provided in the Electron Documentation (Issue B; Issue C; Issue D)

Mitigation

We recommend carefully following and implementing the steps outlined in the Electron checklist in order to improve the security of the Clorio wallet Electron application.

Status

The Clorio team has demonstrated in the Git revision provided for verification that they have sufficiently adopted the electron security recommendations.

Verification

Resolved.

Suggestion 3: Investigate Electronegativity Results More Thoroughly

Synopsis

Our team conducted testing with Electronegativity, a tool used to identify misconfigurations and security anti-patterns in Electron-based applications, which resulted in a significant number of results. Upon investigation, we found that some of the results that have a minor security impact on the current Clorio implementation may result in bigger concerns in future feature additions and development.

Mitigation

We recommend conducting a more in depth investigation of the Electronegativity results, which can be provided by our team, in order to identify which are valid and require further action. We suggest documenting the findings for valid issues and the steps taken to mitigate them, which can then be verified by our team during the verification phase. Furthermore, Electronegativity can be included as part of the Clorio Wallet CI/CD pipeline to enable automated detection of such issues.

Status

The Clorio team has demonstrated that they have investigated issues returned by the Electronegativity report, and have sufficiently addressed found issues.

Verification

Resolved.

Suggestion 4: Implement Unit Tests, End-to-End Tests, and Ledger Tests

Location

`clorio-client`

`clorio-server-assistant`

`mina-ledger-js`

Synopsis

There are no unit tests or end-to-end tests for `clorio-client`, `clorio-server-assistant` and `mina-ledger-js`. Adding tests and measuring the test coverage provides greater confidence in the code and is considered a best practice. In addition, tests contribute to the early detection of bugs.

Mitigation

We recommend writing unit tests and end-to-end tests, measure test coverage, and then run tests as part of the Continuous Integration (CI) run for every commit. Ledger tests can be completed without an actual hardware waller, but by using the Ledger emulator that can run the Mina Ledger Application.

Status

The Clorio team has responded that they are committed to increasing test coverage and are in the process of adding tests. However, this was not completed at the time of this verification. As such, this suggestion remains unresolved.

Verification

Unresolved.

Suggestion 5: Conduct an Audit of Coda Client Javascript SDK

Location

`package/@o1labs/client-sdk`

Synopsis

The cryptographic functionality of the protocol is carried out by the dependency Coda Client Javascript SDK. As this dependency is outside the scope of the audit, it was not included in our security review. This dependency performs a security critical function and should be thoroughly reviewed for security vulnerabilities.

Mitigation

We recommend a third party security review of the Coda Client Javascript SDK to check for security vulnerabilities.

Status

The Clorio team has responded that they believe this suggestion falls under the responsibility of the O(1) Labs and Mina Foundation teams. As such, this suggestion remains unresolved.

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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site 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. While 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, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, 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 though we have not yet 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 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 report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, 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 resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.