Secure Code Review

Findings and Recommendations Report Presented to:

Aleph Zero

May 30th, 2023

Version: 1.2

Presented by:
EMEA Blockchain and AppSec Advisory Team
Kudelski Security

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Aleph Zero engaged Kudelski Security to perform a secure code assessment of their Aleph Zero Signer extension for Chrome and Firefox.

The assessment was conducted remotely by the Kudelski Security Team.
Testing took place on March 23rd, 2023 – April 21st, 2023, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered with their browser extension.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our security evaluation.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Insufficient requirements for password strength.

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code was clean and well structured.
- The developers were very helpful and reactive in answering our questions that we had about the code base and the functionality of their application.
- Tests were also provided as part of the project, which is convenient for better understanding the functionality of different parts of the application.
- Finally, we had regular and very enriching technical exchanges on various topics.

While our comprehensive secure code review has highlighted security vulnerabilities into the Aleph Zero Signer extension, and these findings have been all addressed in the final reviewed codebase, it is important to recognize that this assessment does not guarantee the identification of all potential vulnerabilities, as the constantly evolving nature of the threat landscape requires ongoing vigilance and adaptation.

Scope and Rules of Engagement

Kudelski Security performed a Secure Code Review for Aleph Zero. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied with the commit hashes in public repositories at:

- `https://github.com/Cardinal-Cryptography/aleph-zero-signer/tree/93e89d7e06b0bb08423b6ca8cf4d91bef907d497` .
    - Subfolder `packages` .
    - Written in Typescript and using the React.js framework.

In-Scope Folders
aleph-zero-signer
packages/ ├── extension ├── extension-base ├── extension-chain ├── extension-compat-metamask ├── extension-dapp ├── extension-inject ├── extension-mocks ├── extension-ui

Table 1: Scope

The codebase makes heavy use of the `@polkadot/util-crypto`, `@polkadot/keyring`, `@polkadot/api` libraries. Although these libraries remain out of scope, we make the following statement about their impact on security overall:

- Implementation of cryptographic primitives in these libraries have not ---to the best of our knowledge--- been audited. A bug or vulnerability in the code could potentially introduce vulnerabilities such as, but not limited to, incorrect computation of digital signature, incorrect handling and leakage of secrets, use of improper security parameters in cryptographic algorithms.
- The discovery or introduction of bugs in these libraries could lead to supply-chain attacks, where a vulnerability in a third-party dependency could make the whole application vulnerable and/or allow attackers to gain unauthorized access to sensitive data.

Follow-Up

After the initial report (V1.0) was delivered, Aleph Zero Team addressed all findings in the following codebase revision:

- https://github.com/Cardinal-Cryptography/aleph-zero-signer/pull/82 (commit 608c0c7)

Hence, the final commit of the audited repository is commit 608c0c7.

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we discovered 1 finding that had a medium severity rating, as well as 4 of low severity.

The following chart displays the findings by severity.

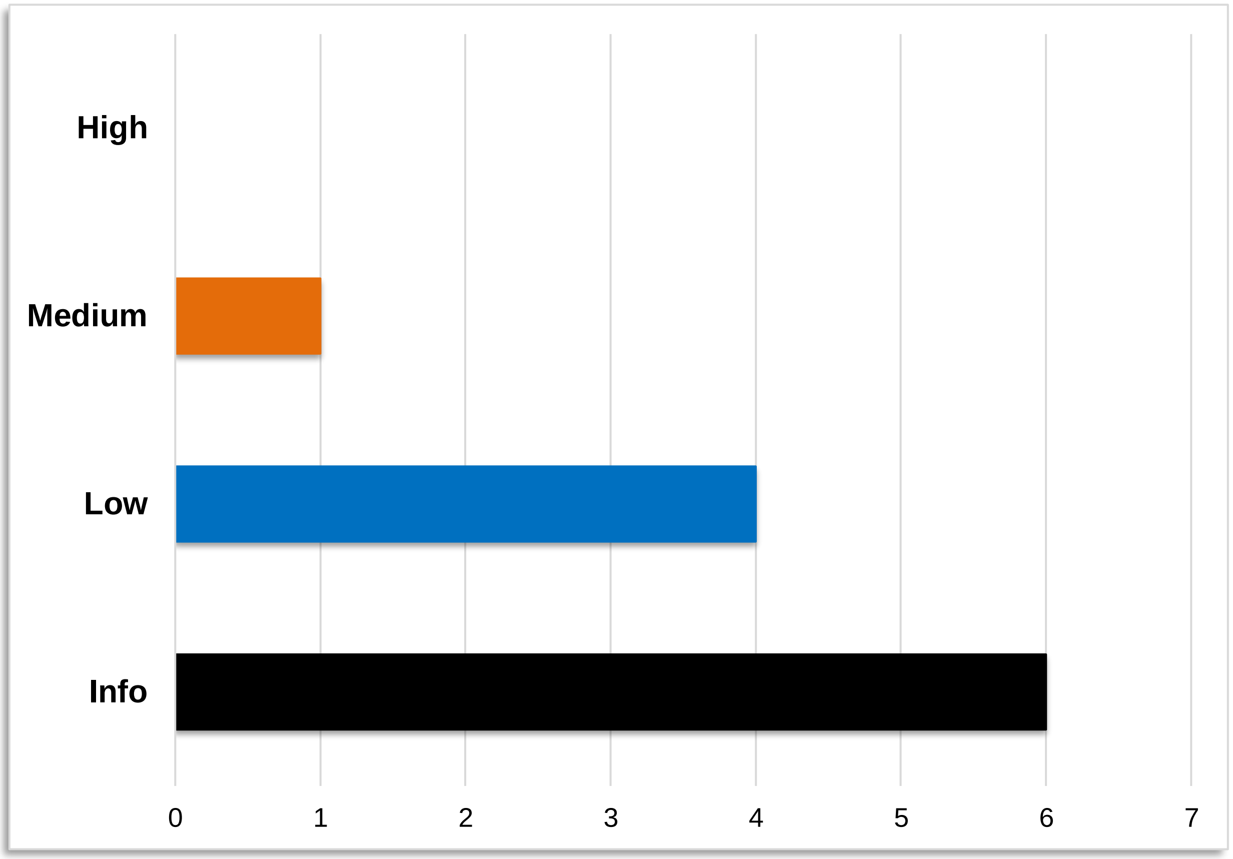

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references. Note that these findings have been all addressed in the final reviewed codebase (commit 608c0c7).

The following table provides an overview of the findings.

#	Severity	Description	Status
KS-AZSE-01	Medium	Insufficient password length	Resolved
KS-AZSE-02	Low	No cleanup of clipboard after use	Resolved
KS-AZSE-03	Low	Outdated external dependencies	Resolved
KS-AZSE-04	Low	Unprotected metadata	Resolved
KS-AZSE-05	Low	No verification of encrypted keypair stored	Resolved
KS-AZSE-06	**Informational**	Changing password not possible	Resolved
KS-AZSE-07	**Informational**	Key encryption at rest can be improved	Resolved
KS-AZSE-08	**Informational**	Unresolved TODO and FIXME	Resolved
KS-AZSE-09	**Informational**	Inadequate code practice	Resolved
KS-AZSE-10	**Informational**	User experience can be improved	Resolved
KS-AZSE-11	**Informational**	Manifest V2 is deprecated	Acknowledged

Table 2: Findings Overview

KS-AZSE-01– Insufficient password length

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Medium	Moderate

Description

The Kudelski Security team has found insufficient checks on the password security strength. This password is used to encrypt the user's pair of public and private key. In particular:

- The only check performed on password strength is whether it contains 6 or more characters.
    - The constant `MIN_LENGTH = 6` is defined in multiple instances across the code base.
    - In two particular instances, the constant is set to 0.
- No other checks on the password are performed (complexity, dictionary attacks…).

KS-AZSE-02 – No cleanup of clipboard after use

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Difficult

Description

Possible private information disclosure after the user copies the mnemonic to clipboard and then continues with the account creation and other operations.

KS-AZSE-03 – Outdated external dependencies

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description

The Kudelski Security team found that the versions of external libraries `@polkadot/api`, `@polkadot/keyring` and `@polkadot/util-crypt` are outdated.

KS-AZSE-04 – Unprotected exported metadata

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description

The metadata section in the exported JSON file containing the account's secret key is not protected against tampering. In particular, the field `genesisHash` that identifies which chain the address is registered with is stored without an integrity check.

KS-AZSE-05 – No verification of encrypted keypair stored

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description

When an account is created, a keypair is created and encrypted by a password. However, the outcome of the keypair storage is not verified. Similarly, an account password is changed, a keypair is re-encrypted with a new password but the result of encryption is not verified. If the encrypted keypair is not stored successfully by some reason, the encrypted keypair may not be restored from the local storage.

KS-AZSE-06 – Changing password not possible

Severity	INFORMATIONAL
Status	RESOLVED

Description

No way to change the account password after an account is created. There exists a functionality for changing a password in the code, but it is not used in the Signer.

KS-AZSE-07 – Key encryption at rest can be improved

Severity	**INFORMATIONAL**
Status	**RESOLVED**

Description
The same password used to authorize a signature in the browser extension is used to derive an encryption key to encrypt the signing key at rest in the JSON file.
The password is used to derive an encryption key through the scrypt function. Scrypt is a password-based key derivation function that is designed to be more memory-intensive than other cryptographic hashes, making it resistant to brute-force attacks. It uses a combination of parameters to define CPU and memory resources. The goal of these parameters is to throttle brute-force attacks by requiring large allocations of memory. In the `@polkadot/util-crypto` implementation it uses the parameters

- N=2**15=32'768, p=1, r=8

Which would be adequate for interactive login purposes, but for storage at rest it may be recommended to increase these parameters to strengthen long term security. According to the specification of the scrypt algorithm, the parameters above will require a minimum of about 32MiB of memory allocated (per password hash). From `@polkadot/util-crypto` source code, it appears that it is not possible to deviate from these default values.

KS-AZSE-08 – Unresolved TODO and FIXME

Severity	**INFORMATIONAL**
Status	**RESOLVED**

Description

Some functionalities are not implemented or marked with comments like TODO and FIXME.

KS-AZSE-09 – Inadequate code practice

Severity	**INFORMATIONAL**
Status	**RESOLVED**

Description

The Kudelski Security Team identified coding snippets and patterns that do not follow industry-standard best practices. These issues were not found to result in identifiable vulnerabilities but may lead to security issues or contract stability problems.

KS-AZSE-10 – User experience can be improved

Severity	**INFORMATIONAL**
Status	**RESOLVED**

Description

The Signer should be easy to use and be injected to the Web wallet without confusion. The improved usability could lead to the usage of Aleph Zero Signer, in turn, the security level is improved.

KS-AZSE-11 – Manifest V2 is deprecated

Severity	**INFORMATIONAL**
Status	**ACKNOWLEDGED**

Description

When building the Aleph Zero Signer and deploying it on the Chrome Browser, the following message applies: “Manifest version 2 is deprecated, and support will be removed from 2023.” A manifest is a `.json` file that tells the web browser what API to use to interact with the extension. Since 2018, Chrome has been migrating to Manifest V3, an improved and more secure version. Support for the existing version, Manifest V2, is slowly being phased out.

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common vulnerabilities such as the OWASP Top 10.

Tools

The following tools were used during this portion of the test. A link for more information about each tool is provided as well.

- Semgrep (https://github.com/returntocorp/semgrep)
- ts-dependency-graph (https://github.com/PSeitz/ts-dependency-graph)
- typescript-eslint (https://typescript-eslint.io/)

Vulnerability Scoring System

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system.

Impact
The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

High:
The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client

Medium:
It is reasonable to assume that the vulnerability would have a measurable affect on the company and systems that may cause minor financial or reputational damage.

Low:
There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood
The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

High:
It is extremely likely that this vulnerability will be discovered and abused

Medium:
It is likely that this vulnerability will be discovered and abused by a skilled attacker

Low:
It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty
Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

Easy:
The vulnerability is easy to exploit or has readily available techniques for exploit

Moderate:
The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.

Difficult:
The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

CONCLUSION

The Kudelski Security Team has identified, during the review of the codebase, a few security concerns, ranging from LOW to MEDIUM. All of those findings have been addressed by the Aleph Zero Team in the final version of this report.

Based on our results, the revision codebase meets adequate code maturity and security requirements.

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Joo Yeon Cho	**Blockchain Expert**	**joo.cho@kudelskisecurity.com**
Adina Nedelcu	**Security Engineer**	**adina.nedelcu@kudelskisecurity.com**
Luca Dolfi	**Blockchain Engineer**	**luca.dolfi@kudelskisecurity.com**