Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Audit-Report Eulith MetaMask Snap Build & Codebase 01.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi

Index

Introduction

"Single source solution for on-chain asset management. Eulith provides asset managers and accredited investors with the best technology and financial services to access & manage capital on-chain."

From https://www.eulith.com/

This report describes the results of a security assessment conducted by Cure53 and aimed at examining the Eulith MetaMask snap. The project included a source code audit and feature-review of the Eulith MetaMask snap's build, codebase and server API endpoints.

The work was requested by Eulith in January 2024 and then carried out by Cure53 in the same month, namely in CW04. A total of three days were invested to reach the coverage expected for this project. Two senior testers were assigned to this project's preparation, execution and delivery.

The work was split into two separate work packages (WPs):

- **WP1**: Source code audits against Eulith MetaMask snap build & codebase
- **WP2**: Code audits & feature reviews of Eulith MetaMask snap & server API

Cure53 was provided with sources, URLs, as well as all further means of access required to complete the tests. The methodology chosen here was white-box for the majority of the audit, with the exception of the Eulith backend API endpoints, which were approached with black-box method.

All preparations were done in January 2024, namely in CW03, which means that Cure53 had a smooth start into the testing process. Communications during the test were done in a dedicated shared Slack channel set up between the respective workspaces of Eulith and Cure53 team. All involved personnel from both parties could join test-related exchanges on Slack.

The project could be completed without any major hurdles. Not many questions had to be posed by Cure53 and the quality of all project-related interactions was consistently excellent. Ongoing exchanges contributed positively to the overall outcomes of this project. Significant roadblocks could be avoided thanks to clear and diligent preparation of the scope. While Cure53 offered frequent status updates about the test, no live-reporting was requested or executed.

The Cure53 team managed to get good coverage over the WP1-WP2 scope items. Only two discoveries were made: one was classified to be a security vulnerability and the other was categorized as a general weakness. As the key target and codebase evaluated during this audit can be seen as very small, a long list of problems was not anticipated. Nevertheless, it is a positive outcome that this expectation was met.

The sole vulnerability was filed with *Low* impact score. In addition, it should be noted that the identified miscellaneous issue was considered to be out of scope (OOS), which results in just one scope-relevant finding. In order to inform the Eulith team about the existing weakness, EUL-01-002 was nevertheless included in this report. All in all, the Eulith team can be congratulated on their excellent work so far. The Eulith MetaMask snap has already been implemented with various security measures and is capable of deflecting serious threats.

The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Then, what the Cure53 team did in terms of attack attempts, coverage, and other test-related tasks is explained in a separate chapter on test methodology.

Next, all findings are discussed in grouped vulnerability and miscellaneous categories. In addition to technical descriptions, PoC and mitigation advice will be provided in the corresponding tickets where applicable.

The report closes with drawing broader conclusions relevant to this January 2024 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Eulith MetaMask snap complex, in particular its build, codebase and server API endpoints.

Scope

- **Code audits & security reviews of Eulith MetaMask snap build & codebase**
    - **WP1**: Source code audits against Eulith MetaMask snap build & codebase
        - **Primary focus:**
            - General tests & attacks against browser add-ons, extension snap-ins, considered independently of the specific use-case as a MetaMask snap
        - **Repository URL:**
            - https://github.com/Eulith/eulith-metamask-snap
        - **Audited commit:**
            - d47f88f62e90cd7f2130057e8e59d216e5d09dcc
    - **WP2**: Code audits & feature reviews of Eulith MetaMask snap & server API
        - **Primary focus:**
            - Specific features including - but not limited to:
            - Credit rating checks
            - Reliability and security of relevant server-side APIs
            - Protection against UI spoofing and UI redressing attacks
            - Falsified results
            - General spoofing
        - **Repository URL:**
            - https://github.com/Eulith/eulith-metamask-snap
        - **Audited commit:**
            - d47f88f62e90cd7f2130057e8e59d216e5d09dcc
        - **Test-supporting material was shared with Cure53**
    - **All relevant sources were shared with Cure53**

Technical Methodology Report

This section presents a comprehensive summary of the technical methodology used during the audit of the Eulith snap project. The approach aimed at ensuring thorough coverage of the provided MetaMask snap. It also spanned a cursory black-box evaluation of the Eulith backend API.

The auditing setup involved the following steps:

- **Preparation and planning**. The auditing process began with a detailed planning phase. The team identified key components and features of the Eulith MetaMask snap, as well as its expected interactions with the server API.
- **Environment configuration.** A controlled testing environment was established to mirror the operational environment of the Eulith MetaMask snap. This setup included configuring the necessary tools, setting up the backend, and checking access for the provided test-accounts.
- **Access and tools.** Cure53 was granted access to the Eulith MetaMask snap repository, along with relevant account access and a Docker image of the backend. The team employed manual source code audit as well as black-box API testing during this examination.

The audit was essentially broken down into two phases, corresponding to the WPs.

In WP1, which entailed a source code review (WP1, Eulith MetaMask Snap), Cure53 conducted a detailed and thorough review of the Eulith MetaMask snap's source code. The primary goal was to inspect the relatively compact codebase of the MetaMask snap, in the hope of identifying any potential security vulnerabilities. Assessment of the overall code quality was also a priority here.

The review process placed a strong emphasis on several key areas, including the management of snap-state, which is crucial for maintaining the integrity and functionality of the application. Special attention was also given to how requests are constructed for communication with the Eulith backend, as it entails a critical aspect for ensuring secure and efficient data exchange.

Additionally, the handling of Chain IDs was another focal point, given its importance in the context of blockchain and cryptocurrency transactions. By examining these specific areas, the review aimed to uncover any hidden flaws or weaknesses that could compromise the security and effectiveness of the MetaMask snap.

As for WP2, which was a black-box test of the Eulith API and backend**,** the team carried out a series of semi-randomized tests, These were mostly targeting the server API, with a particular emphasis on interactions based on the Ethereum blockchain. This approach, known as black-box testing, involved examining the system from an external perspective.

As such, it happened without prior knowledge of the internal working patterns, with the goal of understanding how it responds to various inputs and conditions. The testing was primarily focused on assessing the API's robustness and resilience against potential security threats that are common in Ethereum-based transactions and interactions.

Furthermore, although it was not initially within the defined scope of the audit, the team expanded their investigation to include an in-depth analysis of the Docker container associated with the Eulith API and backend. This additional examination included a thorough analysis of the container's logs, which can provide valuable insights into the system's operations and potential security issues.

The team also scrutinized the methods used for storing secrets within the container, as secure secret storage is critical for maintaining the overall security and integrity of the system. This approach to testing was taken on to pinpoint any vulnerabilities or issues that might not be immediately apparent in a more narrowly-scoped audit.

Throughout the audit process, Cure53 maintained an open communications channel with the Eulith team, which resulted in quick updates on the status of the audit, communicating on the shared understanding of the WPs, and sharing useful audit assets, such as a Docker image for the backend.

Given the extremely small nature of the main auditing target (~350 lines of TypeScript code mostly dealing with formulating JSON RPC requests as well as some basic MetaMask snap-state management), it comes as no significant surprise that only one minor in-scope finding was documented.

The finding, EUL-01-001, merely identifies a potentially inaccurate logic for identifying URIs for local vs. remote backend instances when looking up chain IDs. To that end, Cure53 suggests adoption of a more comprehensive logic in order to remedy the issue.

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *EUL-01-001*) to facilitate any future follow-up correspondence.

EUL-01-001 WP1: Imprecise conversion of Chain ID to URI *(Low)*

The Eulith MetaMask snap uses a *startsWith* check in order to quickly differentiate between calls to Chain IDs being made towards a locally hosted Eulith instance (hosted on *localhost*) and those hosted on remote servers.

The *startsWith* check is imprecise, which means it can be fooled if, for example, the *eulithDomain* value is *localhostButNotReally.com*. The latter could refer to a third-party server.

Affected file:
txInsights.ts

Affected code:
```
function chainIdToUrl(eulithDomain: string, chainId: number): URL {
  if (eulithDomain.startsWith('localhost')) {
    return new URL(`http://${eulithDomain}/v0`);
  }

  let network;
  if (chainId === 1) {
    network = 'eth-main';
  } else if (chainId === 137) {
    network = 'poly-main';
  } else if (chainId === 42161) {
    network = 'arb-main';
  } else {
    throw new Error(
      `The current chain (id = ${chainId}) is not supported. Please contact
Eulith if you are interested in trading on this chain.`,
    );
  }
  return new URL(`https://${network}.${eulithDomain}/v0`);
}
```

It is recommended to use a strict regular expression check instead of *startsWith('localhost').* The regular expression could, for example, take the format: *^localhost\:\d{1,4}$.* Such a regular expression would also include the port. In the end, it would be harder to bypass.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

EUL-01-002 OOS: HSM PIN stored in cleartext *(Info)*

***Note**: This finding is recognized as being out-of-scope for this audit and is presented here purely as a courtesy. It should not be counted as a formal audit finding. Furthermore, Eulith has confirmed that local secrets are only used during testing and that AWS Secrets Manager[1] is used in production, strongly limiting the relevance of this finding even more.*

Peeking into the provided (publicly available) Docker image for the Eulith server, it was found that secrets were stored in cleartext in a *secrets.txt* file.

Affected file:
secrets.txt

Affected code:
```
HSM_USER_PIN=[REDACTED]
DATABASE_PASSWORD=[REDACTED]
POCKET_UNIVERSE_ID_HEADER=testkey
```

Two points should be made here. First, the Docker image contains binary blobs and, second, reverse-engineering them is well out-of-scope for this audit. Therefore, it is unclear whether these secret values are of any real significance or if their exposure even endangers the security or integrity of the Docker instance in any way.

Nevertheless, it may be considered better practice to at least pass these secret values as *environment* variables, or to load them from some other external service for secrets. This could be done in order to prevent their storage in cleartext as constants in every Docker instance of the Eulith server.

[1] https://aws.amazon.com/secrets-manager/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The audit of the Eulith MetaMask Snap and backend API was conducted with a focus on identifying security vulnerabilities and code quality-related issues. This included both a manual source code audit (WP1) and black-box API testing (WP2).

For preparation, key components and interactions of the Eulith MetaMask snap and server API were identified. A controlled testing environment was set up, mirroring the operational environment of the Eulith MetaMask snap. This was fostered by the Eulith team granting Cure53 access to the Eulith MetaMask snap's source code, as well as provisioning test-accounts and a Docker image of the backend.

WP1 consisted of a detailed examination of the MetaMask snap codebase focused on security vulnerabilities in areas such as snap-state management, request construction and Chain ID handling. One minor inaccuracy in the logic was identified in EUL-01-001. The recommended fix for this problem is to implement a strict regular expression check to replace the *startsWith* method. This will offer a more accurate identification of URI origins.

WP2 consisted of semi-randomized testing against the server API, with a focus on Ethereum-based interactions. Out-of-scope examination of the Docker container included log analysis and secret storage inspection.

Some information is also noted in EUL-01-002, in terms of considering the cleartext storage of sensitive keys in the development realm. The Eulith team has confirmed that this does not occur in production, and this note applies to out-of-scope elements anyway. As such, it is included purely for completeness and should not be considered a formal component of the audit's results.

In conclusion, the audit revealed one minor in-scope finding related to the identification logic for URIs, and one out-of-scope observation regarding secrets-management. Overall, given the small size of the codebase (~350 lines of TypeScript code), the limited number of findings was not especially surprising. The identified issue was presented with an easy-to-implement remedy and with a specific recommendation for improvement.

Cure53 would like to thank Ian Fisher from the Eulith team for his excellent project coordination, support and assistance, both before and during this assignment.