Cure53

Fine penetration tests for fine websites

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

Audit-Report Rubic MetaMask Snap Build & Codebase 02.2024

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

Index

Introduction

"As one of the seasoned players in the cross-chain market, Rubic has elaborated on the robust practices of maintaining security for its users along with SDK and widget integrators."
From https://docs.rubic.finance/rubic/security

This document presents the findings of a combined penetration test and source code audit conducted against the Rubic MetaMask Snap and associated server-side APIs. The engagement was initiated at the request of Chapter LTD (Rubic) in February 2024 and executed by Cure53 in the same month (week CW07). A total of three days were dedicated to achieving the desired coverage for this project.

The assessment was divided into two distinct work packages (WPs):

- **WP1**: Source code audits against Rubic MetaMask Snap build & sources
- **WP2**: Code audits & feature reviews against Rubic MetaMask Snap & server API

For this engagement, Cure53 employed a white-box methodology and was granted full access to source code, URLs, documentation, and other necessary materials. A dedicated team of two senior testers prepared, executed, and finalized the project. All preparatory initiatives were conducted in early February 2024 (week CW06) to ensure a seamless commencement of testing.

Clear and consistent communication was facilitated through a shared Telegram channel established solely for this project. Representatives from both Rubic and Cure53 actively participated in this channel, minimizing disruptions and maintaining smooth dialogue. The well-defined and comprehensive scope proved effective, leading to efficient testing with no significant roadblocks encountered throughout the process.

Cure53 provided regular status updates regarding the assessment and its findings. While live-reporting was not explicitly requested for this engagement, the team remained committed to transparent communication.

Through extended testing across the WP1 and WP2 scope, Cure53 identified a single vulnerability requiring attention. This finding, documented as ticket RUB-01-001, was assigned a *Medium* impact score. Generally, the examination of the Rubic MetaMask Snap build and codebase revealed an overall robust security posture, with only this sole vulnerability noted.

The identified vulnerability pertains to insufficient validation of Ethereum addresses within the application. Specifically, the checks for valid hexadecimal characters and EIP-55 checksums were found to be inadequate. This could potentially allow exploitation by malicious actors seeking to introduce invalid addresses into the system.

The security assessment was conducted using a thorough methodology that combined deployment-based and source-code-based testing techniques. This comprehensive approach was essential in identifying the single vulnerability, despite the generally safeguarded environment. In conclusion, the Rubic MetaMask Snap and associated server-side APIs exhibited a commendable security posture, confirmed by the presence of only one discovered flaw.

Onward, the *Scope* chapter next details the defined composition of the assessment, outlining the specific components of the Rubic MetaMask Snap and server-side APIs included in the testing. This section is followed by a breakdown of the comprehensive testing strategies utilized during the evaluation. This will offer transparency to the client and demonstrate the thoroughness of the review process, despite the limited number of identified vulnerabilities. Next, all identified vulnerabilities and general weaknesses discovered during the assessment are provided in ticket format and in chronological order. These are accompanied by a high-level rundown, Proof-of-Concept (PoC) and/or steps to reproduce, and recommendations for mitigation or remediation.

The concluding section will summarize the key findings and overall security posture of the Rubic MetaMask Snap and server-side APIs based on the assessment results. Cure53 will offer its professional insights and recommendations for further enhancing the security posture of the application.

Scope

- **Code audits & security reviews against Rubic MetaMask Snap build & codebase**
  - **WP1**: Source code audits against Rubic MetaMask Snap build & sources
    - **Test environment Docker container:**
      - https://github.com/Cryptorubic/rubic-snap-frontend/tree/audit
    - **Source code:**
      - *All relevant code was shared with Cure53 in the form of a .zip file.*
      - *rubic-snap-backend-develop.zip*
    - **Primary focus:**
      - General tests & attacks against browser add-ons and extension snap-ins, independently of specific use case as a MetaMask snap.
  - **WP2**: Code audits & feature reviews against Rubic MetaMask Snap & server API
    - **Test environment Docker container:**
      - https://github.com/Cryptorubic/rubic-snap-frontend/tree/audit
    - **Source code:**
      - *All relevant code was shared with Cure53 in the form of a .zip file*
      - *rubic-snap-backend-develop.zip*
    - **Primary focus:**
      - Specific features, reliability, and security of relevant server-side APIs, protection against UI spoofing and UI redressing attacks, falsified results, general spoofing, etc.
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Test Methodology

The test methodology adopted for the security audit of the Rubic MetaMask Snap build and codebase was designed to provide comprehensive analysis of both the application’s source code and its operational environment. This methodology encompassed a combination of deployment- and source-code-based testing, aiming at identifying potential security vulnerabilities and operational weaknesses that could impact the security posture of the application.

Preparation Phase

The preparatory phase of the engagement focused on establishing a firm foundation for successful testing. This involved two key activities: firstly the material review, whereby all essential testing materials provided through Google Drive and other platforms were meticulously perused. This comprehensive analysis examined the application's architecture, functionality, and potential attack surfaces, ensuring a deep understanding of the system under assessment.

Subsequent to the material review, the team constructed a controlled testing environment replicating the production environment as closely as possible. This involved deploying the Rubic MetaMask Snap build, server-side APIs, and any required dependent services. This mirrored architecture ensured that testing activities would not interfere with live operations, upholding data integrity and system functionality. This rigorous preparation provided an intimate understanding of the application and enabled precise planning for the subsequent testing phases.

White-Box Audit

The white-box audit phase consisted of several key activities designed to comprehensively assess the application's security. This phase commenced with a thorough source code audit, with the Cure53 consultants reviewing the codebase with particular emphasis on critical components such as the storage backend. The objective was to uncover security vulnerabilities, identify potential code quality issues ("code smells"), and pinpoint areas for optimization. This rigorous scrutiny ensured that the most essential elements of the application were evaluated for potential security risks.

Further bolstering the white-box code analysis was a rigorous examination of the project's dependencies. Leveraging tools like OWASP Dependency-Check alongside others tailored for Django environments, this analysis sought to expose any known vulnerabilities residing within external components. Identifying and rectifying such weaknesses within dependencies is often neglected, yet remains critical for robust software security.

Complementing these activities, manual penetration testing was conducted within the provided Docker environment. This focused assessment scrutinized critical areas like authentication, session management, and input validation. Particular attention was paid to replicating production conditions, simulating real-world attack scenarios to rigorously evaluate the application's resilience against advanced threats.

The combined efforts of penetration testing and a thorough source code review provided a holistic understanding of the application's security posture. This iterative process ensured a responsive and adaptable approach, dynamically refining the audit's focus based on emergent findings to address the evolving security landscape.

The audit's results, particularly the absence of significant findings in areas such as input validation, reflect the development team's effective implementation of security measures. The robust input validation mechanisms in place were instrumental toward nullifying common vulnerabilities such as SQL injection, XSS, and command injection. Additionally, the limited attack surface presented by the MetaMask Snap's architecture, operating within a sandboxed environment, significantly reduced potential attack vectors. This not only constrained direct access to critical blockchain functionalities and user assets, but also minimized the potential impact of a security breach. The audit concluded with one finding, RUB-01-001, which was promptly reported to the Rubic team and addressed the next business day. In tandem, the aforementioned factors underscored an environment with minimal exploitable vulnerabilities, attesting to the Rubic team's prioritization of security for their design and coding practices.

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, all tickets are given a unique identifier (e.g., *RUB-01-001*) to facilitate any future follow-up correspondence.

RUB-01-001 WP2: Imprecise ETH address validation *(Medium)*

Cure53 noted that the current implementation of Ethereum-like (ETH) address validation in the *ethlike_address_is_valid* method checks for two conditions only: the length of the address (which should comprise 42 characters, including the *0x* prefix) and the presence of the *0x* prefix itself. While these checks are necessary, they are insufficient for the purpose of accurately validating an ETH address. Post-hashing, ethereum addresses are represented as 40 hexadecimal characters prefixed with *0x*; thus, the total length constitutes 42 characters. However, not all 42-character strings starting with 0x are valid Ethereum addresses. The insufficiencies regarding the current validation protocols are detailed next:

- **Hexadecimal characters:** The current validation does not ensure that the characters following the *0x* prefix are hexadecimal (0-9, a-f, or A-F). Non-hexadecimal characters would render the address invalid but would pass the current validation.
- **Checksum validation:** Ethereum addresses generated from EIP-55 contain a checksum to protect against typos or case errors. The current validation does not check for EIP-55 checksum compliance, which is crucial for verifying that the address has not been tampered with or mistyped.

The affected code snippet for validating the Ethereum address is displayed below.

Affected file:
storage_backend/base/utils.py

Affected code:
```
def ethlike_address_is_valid(address: ETHLikeAddress) -> bool:
    if len(address) != 42 or not address.startswith('0x'):
        return False
    return True
```

To mitigate this issue, Cure53 recommends adopting a two-pronged approach, as extrapolated below:

- **Hexadecimal check:** After verifying the presence of the *0x* prefix and the correct length, one must insert a check to ensure that the remainder of the address consists only of hexadecimal characters. This can be achieved using a regular expression or a similar method to validate each character.
- **EIP-55 checksum implementation:** Introduce the EIP-55 checksum validation to enhance the address verification process. This involves converting the address to a specific case format based on the hash of the lowercase hexadecimal address. Libraries or utilities that already perform EIP-55 checks should be utilized to simplify this implementation.

An implementation of the fix recommendation outlined above could resemble the following:

Example code containing fixes:
```
import re
from eth_utils import is_checksum_address, to_checksum_address

def ethlike_address_is_valid(address: str) -> bool:
    # Check for correct length and '0x' prefix
    if len(address) != 42 or not address.startswith('0x'):
        return False
    # Ensure the address is hexadecimal
    if not re.fullmatch(r'0x[a-fA-F0-9]{40}', address):
        return False
    # Validate against EIP-55 checksum (optional, based on use case)
    if not is_checksum_address(address):
        # Attempt to convert to checksum address and compare
        try:
            checksum_address = to_checksum_address(address)
            return address == checksum_address
        except ValueError:
            return False
    return True
```

By enhancing the *ethlike_address_is_valid* function to include hexadecimal validation and EIP-55 checksum verification, the accuracy and security of Ethereum address validation can be significantly improved. This will help prevent errors and potential vulnerabilities related to the handling of Ethereum addresses in the application.

Conclusions

The impressions gained during this report, which details and extrapolates on all findings identified during the CW07 2024 testing against the Rubic MetaMask Snap and server-side APIs by Cure53, will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered an excellent impression, as corroborated by the detection of only one security-relevant finding. Albeit, this verdict is likely attributable to the highly constrained scope and attack surface.

The security assessment identified a single *Medium*-severity vulnerability within the Rubic MetaMask Snap build and codebase, designated as RUB-01-001. This vulnerability stemmed from insufficiently stringent validation of Ethereum addresses, a critically important function within the Rubic ecosystem.

The comprehensive test methodology, encompassing both deployment-based and source code-based testing, facilitated the successful detection of this issue despite the absence of further vulnerabilities. This combined approach ensured a thorough evaluation, covering both potential operational and code-level weaknesses.

The assessment extended beyond a traditional source code review, incorporating both dependency analysis and environment penetration testing. This approach ensured a rigorous examination of potential vulnerabilities within external components and the application's operating environment. While no specific vulnerabilities were identified in these areas, the findings suggest a well-configured and secure dependency setup.

While the identified vulnerability pertaining to insufficient Ethereum address validation presents a limited attack surface within the Rubic MetaMask Snap build and codebase, its significance cannot be understated. This vulnerability arises due to the Snap's constrained interaction model with the Ethereum blockchain, typically restricting direct access to critical functionalities and user assets. This inherent limitation naturally focuses potential attack vectors on specific operational functionalities such as address validation, increasing the relevancy of the identified issue despite the narrow attack surface.

The inherently limited attack surface found in MetaMask snaps is a direct consequence of their sandboxed design, restricting access to the Ethereum network and user data. This architecture significantly bolsters security by isolating snaps from sensitive components.

However, this very isolation necessitates meticulous validation of all external inputs and blockchain interactions. Even minor vulnerabilities within these tightly scoped functionalities can have magnified consequences due to the restricted operational environment.

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

To conclude, the Rubic MetaMask Snap and associated server-side APIs demonstrated a commendable security posture throughout the assessment. This is evidenced by the limited attack surface observed and the identification of only a single *Medium*-severity vulnerability. While the overall attack surface may be inherently restricted due to the snap architecture, this finding reaffirms the effectiveness of the security measures implemented.

Cure53 would like to thank Stanislav Iliutkin, Ilya S., and Dmitrii Sleta from the Chapter LTD (Rubic) team for their excellent project coordination, support, and assistance, both before and during this assignment.