HALBORN

MystenLabs - JSON-RPC API

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **March 31st, 2023 - April 28th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/18/2023	Philippe Vogler
0.2	Document Edits	04/18/2023	Erlantz Saenz
0.3	Draft Review	04/18/2023	Gabi Urrutia
0.4	Draft Updates	04/28/2023	Philippe Vogler
0.5	Draft Updates Review	05/01/2023	Erlantz Saenz
0.6	Draft Updates Review	05/01/2023	Gabi Urrutia
0.7	Draft Additions	05/05/2023	Philippe Vogler
0.8	Draft Review	05/09/2023	Erlantz Saenz
1.0	Remediation Plan	05/18/2023	Philippe Vogler
1.1	Remediation Plan Review	05/19/2023	Erlantz Saenz

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Philippe Vogler	Halborn	Philippe.Vogler@halborn.com
Erlantz Saenz	Halborn	Erlantz.Saenz@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MystenLabs engaged Halborn to conduct a security audit on their JSON-RPC API beginning on March 31st, 2023 and ending on April 28th, 2023. The security assessment was scoped to the API endpoints detailed in the official SUI documentation.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the API. The security engineer is an advanced penetration testing expert with knowledge of smart contract security testing and multiple blockchain protocols.

In summary, Halborn identified a single low-risk security issue that was accepted by the MystenLabs team. The issue could allow an attacked to disclose sensitive information about the server that it could be used in a targeted attack.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices.

The following phases were conducted throughout the term of the audit:

- Technology stack-specific vulnerabilities and code audit
- Known vulnerabilities in 3rd party / OSS dependencies
- Application Logic Flaws

- Authentication / Authorization flaws
- Input Handling
- Fuzzing of all input parameters
- Testing for different types of sensitive information leakages: memory, clipboard, etc.
- Testing for injection vulnerabilities (SQL/JSON/HTML/JS/Command/Directories. . .)
- Testing for CRLF
- Testing SSRF/Open Redirects
- Testing for client-side and server-side prototype pollution
- Perform static analysis on code
- Ensure that coding best practices are being followed by MystenLabs team
- Code review and test of the BCS implementation for deserialization vulnerabilities
- Server-side misconfigurations
- Testing the handling of custom transactions

Logic tests performed during the engagement have been further described in the last section of this report - FURTHER LOGIC TEST CASES CONDUCTED.

In-depth manual testing of transactions was conducted using custom test cases Rust implementations. The Rust test cases were run through Visual Code's debugger to edit values on the stack during the test case's execution, thereby creating custom/malformed transactions or signatures. This was achieved using the Visual Code plugin CodeLLDB and a hexadecimal editor.

Two Rust test cases have been written for this engagement, respectively to send transactions from a multisig address, or single address. A multisig transaction requires more than one private key to authorize it. The latter implementation can be used to transfer SUI or objects with a single signature, while the former handles the multisig transaction.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

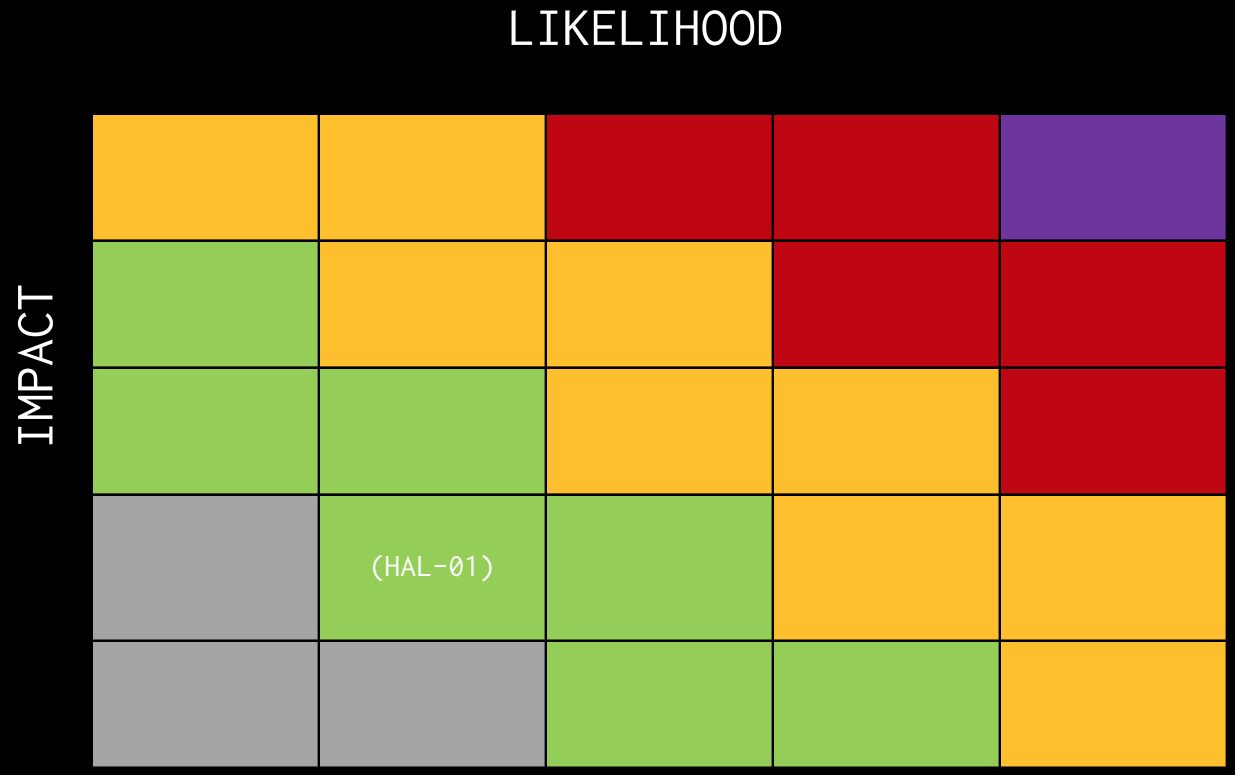
1.4 SCOPE

JSON-RPC APIs in a local network.

Documentation provided:
- https://docs.sui.io/sui-jsonrpc

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	0



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 VERBOSE ERROR MESSAGES	Low	RISK ACCEPTED - 05/18/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) VERBOSE ERROR MESSAGES - LOW

Description:

During the deserialization routine performed on the transaction signer's Sui address parameter, the server was returning verbose errors along with the full path to the file in which the error occurred. This could allow an attacker to further formulate a targeted attack against the server, or against the local user disclosed in the full path via vectors such as brute force attacks.

Evidences:

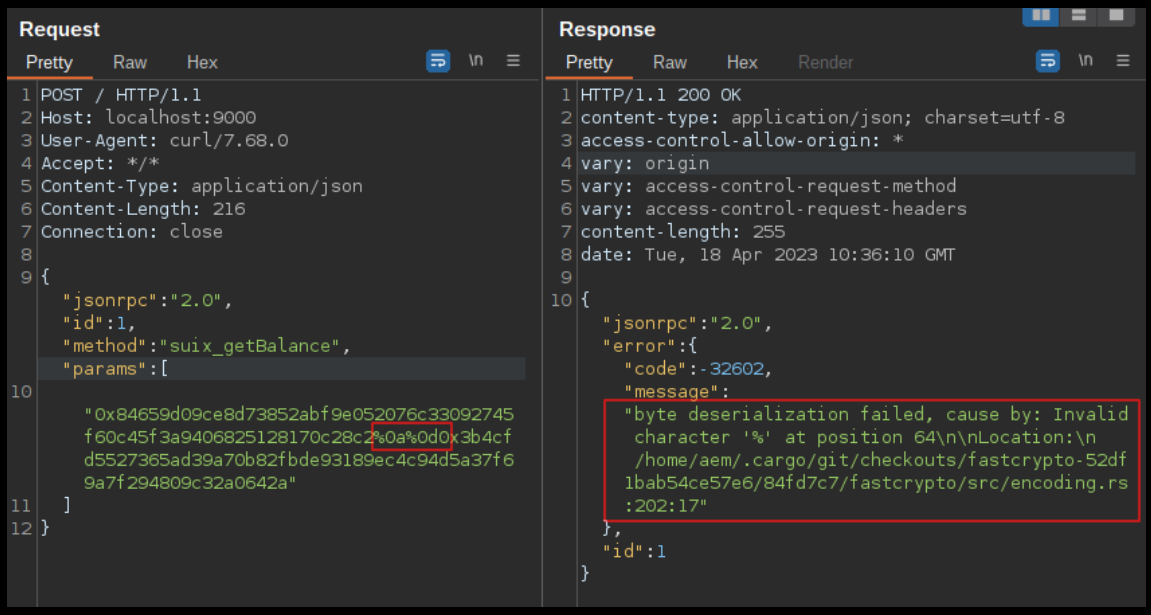


Figure 1: Verbose error disclosing the local user

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

Return generic error messages to the user and log the exception/error server-side.

Remediation Plan:

RISK ACCEPTED: The risk of the issue was assumed by the Mysten Labs team and accepted by Halborn.

3.2 FURTHER LOGIC TEST CASES CONDUCTED

- Send SUI to the same address as sender.
  - Endpoint attacked: transfer_sui, unsafe_paySui, unsafe_payAllSui, unsafe_pay
  - Parameters tested: signer, recipients, recipient
  - Result: Failed. SUI balance remains unchanged.
- Send SUI, but pay gas with another owner's object ID.
  - Endpoint attacked: transfer_sui, unsafe_paySui, unsafe_payAllSui, unsafe_pay
  - Parameters tested: sui_object_id, input_coins, gas
  - Result: Failed. Cannot use another owner's object ID.
- Send SUI without paying for gas.
  - Endpoint attacked: transfer_sui, unsafe_paySui, unsafe_payAllSui, unsafe_pay
  - Parameters tested: gas, gas_budget
  - Result: Failed. Minimum gas fees are checked.
- Send more SUI than available in balance.
  - Endpoint attacked: transfer_sui, unsafe_paySui, unsafe_pay
  - Parameters tested: amount, amounts
  - Result: Failed. The balance is properly checked.
- Transfer object, but pay gas with another owner's object ID.
  - Endpoint attacked: unsafe_transferObject
  - Parameters tested: gas, object_id
  - Result: Failed. Cannot use another owner's object ID.
- Transfer SUI or object to multiple addresses.
  - Endpoint attacked: unsafe_transferObject, unsafe_payAllSui, unsafe_transferSui
  - Parameters tested: sui_object_id, recipient, object_id
  - Result: Failed. Found no way to send SUI or transfer object ID to multiple address. Input is properly checked.

- Tamper with SUI values, such as sending null values, negative values, or improper types.
  - Endpoint attacked: `unsafe_transferObject`, `unsafe_payAllSui`, `unsafe_transferSui`, `unsafe_paySui`, `unsafe_pay`, `unsafe_publish`, `unsafe_requestAddStake`, `unsafe_requestWithdrawStake`, `unsafe_moveCall`, `unsafe_splitCoin`, `unsafe_splitCoinEqual`
  - Parameters tested: `gas_budget`, `amount`, `amounts`, `coins`, `split_count`, `split_amounts`.
  - Results: Failed. Thresholds and input types are properly checked.
- Transfer the same object twice.
  - Endpoint attacked: `unsafe_transferObject`
  - Parameters tested: `object_id`
  - Results: Failed. Cannot transfer the same object twice.
- Send invalid tx bytes but valid sig.
  - Endpoint attacked: `sui_executeTransactionBlock`
  - Parameters tested: `tx_bytes`
  - Results: Failed. Transaction bytes are checked in conjunction with the tx signature. Both have to be valid and match.
- Send invalid sig but valid tx bytes.
  - Endpoint attacked: `sui_executeTransactionBlock`
  - Parameters tested: `signatures`
  - Results: Failed. The signature is checked in conjunction with the tx bytes. Both have to be valid and match.
- Include more than 10 signatures in a transaction.
  - Endpoint attacked: `sui_executeTransactionBlock`
  - Parameters tested: `signatures`
  - Results: Failed. Could not specify more than 10 signatures.
- Inject additional JSON parameters to force values, i.e. object version or object digest.
  - Endpoint attacked: `transfer_sui`, `unsafe_paySui`, `unsafe_payAllSui`, `unsafe_pay`

    - Results: Failed. The integrity of the request is checked. Could not duplicate any JSON parameter to force values.
- Create malformed requests to observe any server panic.
    - Endpoint attacked: `sui_executeTransactionBlock`
    - Parameters tested: `tx_bytes`, `signatures`, `options`, `request_type`
    - Results: Failed. Malformed requests are rejected.
- Withdraw another user's staked coins.
    - Endpoint attacked: `unsafe_requestWithdrawStake`
    - Parameters tested: `staked_sui`, `signer`, `gas`
    - Results: Failed. Staked coins could only be withdrawn by the expected signer.
- Stake coins, but pay gas with another owner's object ID.
    - Endpoint attacked: `unsafe_requestWithdrawStake`, `unsafe_requestAddStake`
    - Parameters tested: `gas`
    - Result: Failed. Cannot use another owner's object ID.
- Specify an address, which is not a validator when staking SUI.
    - Endpoint attacked: `unsafe_requestAddStake`
    - Parameters tested: `validator`
    - Results: Failed. Could stake coins only to a validator.
- Split coins, but specify a higher value than available ones.
    - Endpoint attacked: `unsafe_splitCoin`
    - Parameters tested: `split_amounts`
    - Results: Failed. Could not split into amounts larger than the total balance.
- Tamper with the signature by changing the Base64 encoding.
    - Endpoint attacked: `sui_executeTransactionBlock`
    - Parameters tested: `signatures`
    - Results: Failed. The signature is checked in conjunction with the tx bytes. Both have to be valid and match.
- Verify that all signatures are required in a multisig transaction.
    - Endpoint attacked: `sui_executeTransactionBlock`

- Parameters tested: signatures
- Results: Error message if one of the signature is missing in a multisig transaction.

THANK YOU FOR CHOOSING

// HALBORN