MystenLabs - Sui-Core

L1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 6th, 2023 - April 21st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/06/2023	Alexis Fabre
0.2	Document Updates	04/10/2023	Alexis Fabre
0.3	Draft Version	04/21/2023	Alexis Fabre
0.4	Draft Review	04/22/2023	Luis Quispe Gonzales
0.5	Draft Review	04/24/2023	Gabi Urrutia
1.0	Remediation Plan	04/28/2023	Alexis Fabre
1.1	Remediation Plan Review	04/28/2023	Luis Quispe Gonzales
1.2	Remediation Plan Review	05/01/2023	Gabi Urrutia

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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com
Alexis Fabre	Halborn	Alexis.Fabre@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MystenLabs engaged Halborn to conduct a security audit on their Layer 1 blockchain, beginning on March 6th, 2023 and ending on April 21st, 2023. The security assessment was scoped to the repository provided to the Halborn team.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

- Ensure that the L1 functions operate as intended
- Identify potential security issues with the codebase

In summary, Halborn identified some security risks that were partially addressed by the MystenLabs team. The main ones are the following:

- Ensure all gas objects to have an owner address.
- Assess whether the separate charge fields in 'SuiGasStatus' and 'GasStatus' structures are necessary or whether they can be consolidated into a single field.
- Update the loop to check is the batch size limits are reached before including new effects.

1.3 SCOPE

1. Repository: sui-core

    (a) Commit ID: cb043989

The commit ID was moved to a fresher version on March 23rd 2023. The updated scope is:

1. Repository: sui-core

    (a) Commit ID: 9629845

1.4 CAVEATS

Please note that due to the large scope of the audit and time constraints, the efforts were concentrated on the most sensitive areas, and not all functionalities could be thoroughly tested. As a result, there may be areas that were not fully evaluated. The reviewed scope includes transactions and messages validation, management, orchestration, and checkpoint execution, but was less focused on metrics and consensus adapter.

1.5 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:

- Research into architecture and purpose.
- Manual code review and walkthrough of the scoped code and imported functions.
- Analyze of sensitive rust native functions (assertions, option, and result unwrapping, vector insertion. . .)
- Study of the behavior and logic and the code.
- Non structure aware fuzzing on parsing functions.
- Scanning of Rust files for vulnerabilities, security hotspots or bugs.

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 Exploitability

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 Impact

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 Severity Coefficient

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
IMMUTABLE GAS COIN DELETION	Medium (5.6)	SOLVED - 04/07/2023
REDUNDANCY IN CHARGE FIELDS FOR GAS STATUS	Informational (0.0)	ACKNOWLEDGED
EXCEEDING MAXIMUM CHECKPOINTS AND TRANSACTIONS IN BATCH DURING EFFECTS PRUNING	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) IMMUTABLE GAS COIN DELETION - MEDIUM (5.6)

Description:

The gas module does check that the first gas coin has an address owner, but does not check the rest of the gas coins. This allowed the passing of immutable coins, which do not have an address owner innately, and led to their unintended deletion through gas smashing.

Code Location:

Only the first gas object is verified:

Listing 1: crates/sui-types/src/gas.rs (Lines 549,552,555)

```
548 pub fn check_gas_balance(
549     gas_object: &Object,
550     gas_budget: u64,
551     gas_price: u64,
552     more_gas_objs: Vec<Object>,
553     cost_table: &SuiCostTable,
554 ) -> UserInputResult {
555     if !(matches!(gas_object.owner, Owner::AddressOwner(_))) {
556         return Err(UserInputError::GasObjectNotOwnedObject {
557             owner: gas_object.owner,
558         });
559     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:L/Y:N/R:N/S:U (5.6)

Recommendation:

It is recommended to perform the verification on the more_gas_objs to ensure all gas coins passed to the function are checked for having an

address owner, preventing the unintended deletion of immutable gas coins.

Remediation Plan:

SOLVED: The issue was remediated in the pull request 10511, where all gas coins were tested for having an address owner.

4.2 (HAL-02) REDUNDANCY IN CHARGE FIELDS FOR GAS STATUS - INFORMATIONAL (0.0)

Description:

A redundancy was identified in the codebase, where both the SuiGasStatus and GasStatus structures have a charge field. Since SuiGasStatus owns GasStatus, it already has direct access to the charge field of GasStatus. This raises concerns about the possibility of future changes leading to inconsistencies in the charge state between these structures.

While no direct issues or vulnerabilities have been found related to this redundancy, maintaining two separate charge fields could increase the likelihood of introducing bugs in the future. In a worst-case scenario, it might lead to incorrect calculations or improper handling of gas charges, which could impact the overall functionality of the system.

Code Location:

The GasStatus owns a charge field:

Listing 2: crates/sui-cost-tables/src/tier_based/tables.rs (Line 51)

```
48 pub struct GasStatus<'a> {
49     cost_table: &'a CostTable,
50     gas_left: InternalGas,
51     charge: bool,
52
53     // The current height of the operand stack, and the maximal
↳ height that it has reached.
54     stack_height_high_water_mark: u64,
55     stack_height_current: u64,
56     stack_height_next_tier_start: Option<u64>,
57     stack_height_current_tier_mult: u64,
58
59     // The current (abstract) size  of the operand stack and the
↳ maximal size that it has reached.
```

```
60      stack_size_high_water_mark: u64,
61      stack_size_current: u64,
62      stack_size_next_tier_start: Option<u64>,
63      stack_size_current_tier_mult: u64,
64
65      // The total number of bytecode instructions that have been
 ↳ executed in the transaction.
66      instructions_executed: u64,
67      instructions_next_tier_start: Option<u64>,
68      instructions_current_tier_mult: u64,
69  }
```

The SuiGasStatus owns a charge field, but also the SuiGasStatus structure, which also owns a charge field:

Listing 3: crates/sui-types/src/gas.rs (Lines 306,308)

```
305 pub struct SuiGasStatus<'a> {
306     gas_status: GasStatus<'a>,
307     init_budget: GasUnits,
308     charge: bool,
309     computation_gas_unit_price: ComputeGasPricePerUnit,
310     storage_gas_unit_price: ComputeGasPricePerUnit,
311     /// storage_cost is the total storage gas units charged so far
 ↳ , due to writes into storage.
312     /// It will be multiplied by the storage gas unit price in the
 ↳  end to obtain the Sui cost.
313     storage_gas_units: GasUnits,
314     /// storage_rebate is the total storage rebate (in Sui)
 ↳ accumulated in this transaction.
315     /// It's directly coming from each mutated object's storage
 ↳ rebate field, which
316     /// was the storage cost paid when the object was last mutated
 ↳ . It is not affected
317     /// by the current storage gas unit price.
318     storage_rebate: SuiGas,
319
320     cost_table: SuiCostTable,
321 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to assess whether the separate charge fields in both structures are necessary or whether they can be consolidated into a single field.

Remediation Plan:

ACKNOWLEDGED: The MystenLabs team acknowledged this finding.

4.3 (HAL-03) EXCEEDING MAXIMUM CHECKPOINTS AND TRANSACTIONS IN BATCH DURING EFFECTS PRUNING - INFORMATIONAL (0.0)

Description:

A potential issue was identified in the pruning function, where effects are pruned even when the number of checkpoints of transactions in the batch exceeds the configured maximum limits. The issue lies in the object pruning function, where objects are included in the batch before checking whether the batch size has exceeded the configured limits for checkpoints and transactions. This could cause the batch to process more checkpoints or transactions than the configured limits, which may lead to slightly greater resource consumption.

Code Location:

The batch effects are extended before checking for the configured limit:

Listing 4: crates/sui-core/src/authority/authority_store_pruner.rs (Lines 159,161)

```
159 batch_effects.extend(effects.into_iter().flatten());
160
161 if batch_effects.len() >= config.max_transactions_in_batch
162     || checkpoints_in_batch >= config.max_checkpoints_in_batch
163 {
164     Self::prune_effects(
165         batch_effects,
166         perpetual_db,
167         objects_lock_table,
168         checkpoint_number,
169         deletion_method,
170     )
171     .await?;
172     batch_effects = vec![];
```

```
173     checkpoints_in_batch = 0;
174 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to update the loop to check is the batch size limits are reached before including new effects in the batch.

Remediation Plan:

ACKNOWLEDGED: The MystenLabs team acknowledged this finding and also mentioned that batches might slightly exceed configured limits, but that could be acceptable as configured limits are used more for heuristics than hard limits.

THANK YOU FOR CHOOSING

// HALBORN