FYEO

Security Assessment for the SUI Blockchain sui-system

Mysten Labs

June 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

Mysten Labs engaged FYEO Inc. to perform a Security Assessment for the SUI Blockchain sui-system.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on April 24 - May 09, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- 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 results of our tests.

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

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-SUI-ID-01 – Add safeguard for ensuring only allowing Genesis execution
- FYEO-SUI-ID-02 – Order in parameter content assumed by functions like compute_adjusted_reward_distribution
- FYEO-SUI-ID-03 – Order of execution
- FYEO-SUI-ID-04 – Variable not named correctly in function derive_reference_gas_price
- FYEO-SUI-ID-05 – Voting power has too few significant figures to handle rounding errors.

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment for the SUI Blockchain sui-system. 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 through a public repository at https://github.com/MystenLabs/sui/tree/releases/sui-v0.33.0-release/crates/sui-framework/packages/sui-system/sources with the commit hash 867e5d318285419156e577a503d69354764b2037.

Files included in the code review

```
sui/
└── crates/
    └── sui-framework/
        └── packages/
            └── sui-system/
                └── sources/
                    ├── genesis.move
                    ├── stake_subsidy.move
                    ├── staking_pool.move
                    ├── storage_fund.move
                    ├── sui_system.move
                    ├── sui_system_state_inner.move
                    ├── validator.move
                    ├── validator_cap.move
                    ├── validator_set.move
                    ├── validator_wrapper.move
                    └── voting_power.move
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment for the SUI Blockchain sui-system, we discovered:

- 2 findings with LOW severity rating.
- 3 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

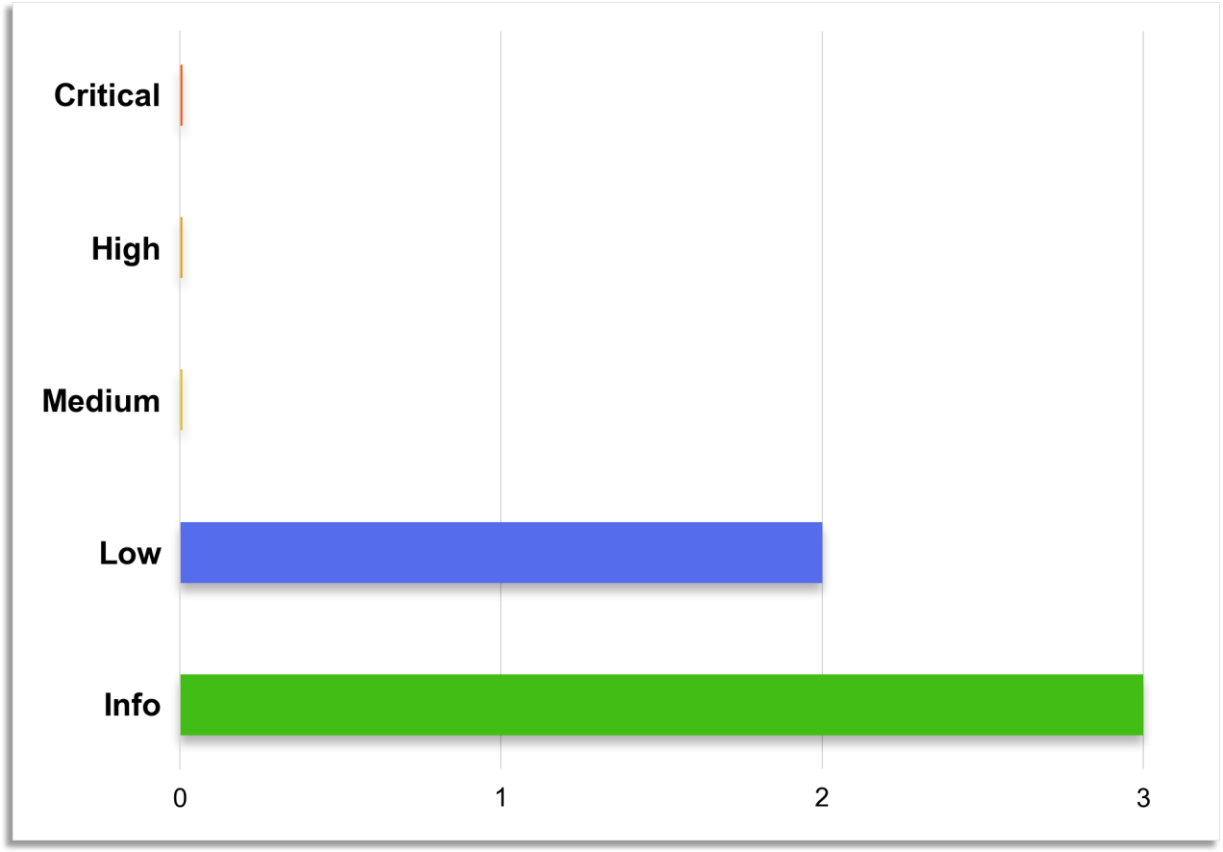


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-SUI-ID-01	**Low**	Add safeguard for ensuring only allowing Genesis execution
FYEO-SUI-ID-02	**Low**	Order in parameter content assumed by functions like compute_adjusted_reward_distribution
FYEO-SUI-ID-03	**Informational**	Order of execution
FYEO-SUI-ID-04	**Informational**	Variable not named correctly in function derive_reference_gas_price
FYEO-SUI-ID-05	**Informational**	Voting power has too few significant figures to handle rounding errors.

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

CONCLUSION

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The Move source code appears to be well-written and organized, which is why no critical, high, or medium vulnerabilities have been found.

Based on this very good result, we have focused on the maintainability of the code base to assure that future work on this central code of the SUI blockchain will be as secure in the future as it is now.

We have observed that the reviewed code base has a mix of programming styles and paradigms. We recommend creating a style guide for the development so that future maintenance will be easy to implement.

When reviewing the code coverage, we were pleased to see that most files in the package had a very decent code coverage percentage, apart from the genesis.move.

```
+-------------------------+
| Move Coverage Summary   |
+-------------------------+
Module validator_cap          >>> % Module coverage: 91.30
Module staking_pool           >>> % Module coverage: 83.57
Module validator              >>> % Module coverage: 92.77
Module voting_power           >>> % Module coverage: 94.26
Module validator_wrapper      >>> % Module coverage: 93.75
Module validator_set          >>> % Module coverage: 92.82
Module storage_fund           >>> % Module coverage: 100.00
Module stake_subsidy          >>> % Module coverage: 59.52
Module sui_system_state_inner >>> % Module coverage: 86.63
Module sui_system             >>> % Module coverage: 92.63
Module genesis                >>> % Module coverage: 0.00
+-------------------------+
| % Move Coverage: 87.44  |
+-------------------------+
```

ADD SAFEGUARD FOR ENSURING ONLY ALLOWING GENESIS EXECUTION

Finding ID: FYEO-SUI-ID-01
Severity: **Low**
Status: **Open**

Description

Most functions that are supposed only to be called at Genesis have no check for that to be true.

Proof of Issue

File name: storage_fund.move

Line number: 23

```
    /// Called by `sui_system` at genesis time.
    public(friend) fun new(initial_fund: Balance<SUI>) : StorageFund {
        StorageFund {
            // At the beginning there's no object in the storage yet
            total_object_storage_rebates: balance::zero(),
            non_refundable_balance: initial_fund,
        }
    }
```

Severity and Impact Summary

If a function that is supposed only to be called at Genesis is called at any other time than Genesis, there would be serious problems for the chain.

Recommendation

To mitigate this issue, we recommend adding an assertion that will abort if the function is called at any other time than Genesis.

There is a usage of the assertion in some functions.

```
assert!(tx_context::epoch(ctx) == 0, ENotCalledAtGenesis);
```

Even though there are transactions made in Epoch 0 that aren't Genesis this is the best that can be done at this point.

References

N/A

ORDER IN PARAMETER CONTENT ASSUMED BY FUNCTIONS LIKE COMPUTE_ADJUSTED_REWARD_DISTRIBUTION

Finding ID: FYEO-SUI-ID-02
Severity: **Low**
Status: **Open**

Description

Parameters that are used when calling the function like compute_adjusted_reward_distribution assume that the order of Validators, unadjusted_staking_reward_amounts,unadjusted_storage_fund_reward_amounts,individual_staking_reward_adjustments, and individual_storage_fund_reward_adjustments are the same. This is not checked in the function!

Proof of Issue

File name: validator_set.move

Line number: 1068

```
    fun compute_adjusted_reward_distribution(
         validators: &vector<Validator>,
         total_voting_power: u64,
         total_slashed_validator_voting_power: u64,
         unadjusted_staking_reward_amounts: vector<u64>,
         unadjusted_storage_fund_reward_amounts: vector<u64>,
         total_staking_reward_adjustment: u64,
         individual_staking_reward_adjustments: VecMap<u64, u64>,
         total_storage_fund_reward_adjustment: u64,
         individual_storage_fund_reward_adjustments: VecMap<u64, u64>,
     ): (vector<u64>, vector<u64>)
```

Severity and Impact Summary

If there is any chance of the order in any of the vectors not being in the correct order assumed by the called function, the result can be disastrous. Any rewards could be allocated to the wrong recipient and, in the end, create an unstable economy in the system as any predictability of the outcome is null.

Recommendation

Create a Map with a mapping from validator addresses to the respective amount so that there is a clear way of determining the connection between the validator and the value to be used in the calculation.

References

N/A

ORDER OF EXECUTION

Finding ID: FYEO-SUI-ID-03
Severity: **Informational**
Status: **Open**

Description

There is a specific order execution that must take place in the function advance_epoch. Today there is no way of ensuring that any further maintenance can interfere with that order and result in unpredictable behavior.

Proof of Issue

File name: validator_set.move

Line number: 348

```
    /// Update the validator set at the end of epoch.
    /// It does the following things:
    ///   1. Distribute stake award.
    ///   2. Process pending stake deposits and withdraws for each validator
(`adjust_stake`).
    ///   3. Process pending stake deposits, and withdraws.
    ///   4. Process pending validator application and withdraws.
    ///   5. At the end, we calculate the total stake for the new epoch.
```

Severity and Impact Summary

If the order is not followed, the calculations that are taking place in the function advance_epoch can result in incorrect values, and the chain's stability is in jeopardy.

Recommendation

We recommend a strategy where the code is refactored, and each step of the process is contained within a function. That way, you can send a parameter from the previous function call to the next, flagging that the calls are in the right order.

You could use a parameter that is changed at the end of the function to the name of the next step. When the next step function is called, you pass the parameter along, and as the first line, you make an assertion checking that the called function is the one to be called.

There are also other ways of being sure that the order is correct, but this would make it very hard not to follow the correct call sequence.

References

N/A

VARIABLE NOT NAMED CORRECTLY IN FUNCTION DERIVE_REFERENCE_GAS_PRICE

Finding ID: FYEO-SUI-ID-04
Severity: **Informational**
Status: **Open**

Description

In the function derive_reference_gas_price, the variable name stake should be renamed to voting_power to clarify that it is the voting power and not the Stake that is compared.

Proof of Issue

File name: validator_set.move

Line number: 528

```
        let pq = pq::new(entries);
        let sum = 0;
        let threshold = voting_power::total_voting_power() -
voting_power::quorum_threshold();
        let result = 0;
        while (sum < threshold) {
            let (gas_price, stake) = pq::pop_max(&mut pq);
            result = gas_price;
            sum = sum + stake;
        };
        result
```

Severity and Impact Summary

Having improper labeled variables can lead to future logical errors in the development process as the naming suggests another functionality than expected.

Recommendation

Rename the variable stake to the voting_power.

References

N/A

VOTING POWER HAS TOO FEW SIGNIFICANT FIGURES TO HANDLE ROUNDING ERRORS.

Finding ID: FYEO-SUI-ID-05
Severity: **Informational**
Status: **Open**

Description

In voting_power.move, TOTAL_VOTING_POWER is set to 10000, which is only four significant figures and can cause nonnegligible rounding errors. For example, on line 80 of voting_power.move, the minimum 15M stake will have rounding errors of up to 1500 $SUI, and a stake of 1B will have rounding errors up to $100000 $SUI. The remainders combine to form the remaining_voting_power, which is then redistributed among the validators in adjust_voting_power (line 118) and can lead to further deviations from the expected stake ratio. NOTE: Since the remaining_voting_power is split equally among all validators, an economically incentivized validator would split their stake into as many accounts as possible to maximize the remaining_voting_power they receive. This can all be simplified by simply including more significant figures in TOTAL_VOTING_POWER.

Proof of Issue

File name: voting_power.move

Line number: 33

```
    /// Set total_voting_power as 10_000 by convention. Individual voting powers can
be interpreted
    /// as easily understandable basis points (e.g., voting_power: 100 = 1%,
voting_power: 1 = 0.01%) rather than
    /// opaque quantities whose meaning changes from epoch to epoch as the total
amount staked shifts.
    /// Fixing the total voting power allows clients to hardcode the quorum threshold
and total_voting power rather
    /// than recomputing these.
    const TOTAL_VOTING_POWER: u64 = 10_000;
```

which is used in

```
    /// Distribute remaining_power to validators that are not capped at threshold.
    fun adjust_voting_power(info_list: &mut vector<VotingPowerInfoV2>, threshold: u64,
remaining_power: u64) {
        let i = 0;
        let len = vector::length(info_list);
        while (i < len && remaining_power > 0) {
            let v = vector::borrow_mut(info_list, i);
            // planned is the amount of extra power we want to distribute to this
validator.
            let planned = divide_and_round_up(remaining_power, len - i);
            // target is the targeting power this validator will reach, capped by
threshold.
            let target = math::min(threshold, v.voting_power + planned);
            // actual is the actual amount of power we will be distributing to this
validator.
            let actual = math::min(remaining_power, target - v.voting_power);
            v.voting_power = v.voting_power + actual;
```

```
            assert!(v.voting_power <= threshold, EVotingPowerOverThreshold);
            remaining_power = remaining_power - actual;
            i = i + 1;
        };
        assert!(remaining_power == 0, ETotalPowerMismatch);
    }
```

Severity and Impact Summary

The remainders combine to form the remaining_voting_power, which is then redistributed among the validators in adjust_voting_power (line 118) and can lead to further deviations from the expected stake ratio.

Based on a deeper discussion and inspection of the full impact, the following is clear and explains the reasoning behind the solution. The voting power of a validator isn't meant to accurately reflect its stake. The rounding that can happen is taken into account when using the value. The main purpose is to cap the voting power of a validator at 10%, even when their stake is > 10%, while still making sure that the relative relationship of two validators' stake amounts is respected, i.e. if A has more stake than B then A should have at least as much voting power as B. A validator's voting power can never fall below 15 because of that; a validator can't have less than 0.15% of the stake because a validator needs 15M of stake to stay in the validator set. Since the total SUI supply is 10B, the voting power of a validator needs to be at least 15M / 10B = 0.15%.

Recommendation

If a need arises to reflect the voting power more accurately, the solution would be to add more significant figures to TOTAL_VOTING_POWER. In that case, we would recommend setting it to the total supply of SUI, so all rounding errors are one SUI or less.

References

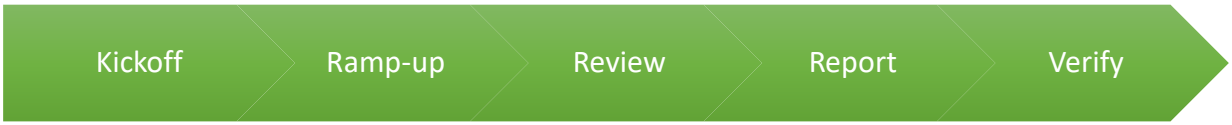
N/A

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

REPORTING

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations