Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Sui Liquid Staking Protocol

Veridise Inc.

May 23, 2024

I Prepared For:

ANKR
https://www.ankr.com/

I Prepared By:

Jon Stephens

I Contact Us: contact@veridise.com

I Version History:

June 17, 2023 Initial Draft

Contents

1 Executive Summary

From June 10, 2023 to June 13, 2023, ANKR engaged Veridise to review the security of their Sui Liquid Staking Protocol. The review covered the liquid staking pool and certificate coin for the Sui Liquid Staking Protocol. Veridise conducted the assessment over 3 person-days, with 1 engineers reviewing code over 3 days. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Sui Liquid Staking Protocol developers provided the source code of the Sui Liquid Staking Protocol contracts for review. To facilitate the Veridise auditors' understanding of the code, the Sui Liquid Staking Protocol developers also provided test cases demonstrating the behavior of the protocol. The source code implements a liquid staking pool for the SUI coin. Users will stake their SUI coins to the protocol in return for ankrSUI certificate tokens. These tokens are then staked to validators via an intermediary. When users unstake their certificate tokens, they are placed in a refund queue for reimbursement when rewards are distributed.

Summary of issues detected. The audit uncovered 6 issues, none of which are assessed to be of high or critical severity by the Veridise auditors. The Veridise auditors did identify one medium-severity issues, which corresponds to the lack of staking transparency as staked funds are simply sent to an intermediary. Additionally, several minor issues were identified, including the lack of data validation when a ratio is updated and the potential for unbacked shares in some situations.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Sui Liquid Staking Protocol	`0x6487534`	Sui Move	Sui

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
June 10 - June 13, 2023	Manual & Tools	1	3 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	1	1
Low-Severity Issues	3	3
Warning-Severity Issues	2	2
Informational-Severity Issues	0	0
TOTAL	6	6

Table 2.4: Category Breakdown.

Name	Number
Logic Error	2
Data Validation	2
Shared Ownership	1
Maintainability	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Sui Liquid Staking Protocol's smart contracts. In our audit, we sought to answer the following questions:

- Can the ratio be updated to a value outside the range of [1, `MAX_RATIO`]?
- Can users observe how their funds are being used?
- Can users mint certificate tokens without staking funds?
- Will users be eventually refunded?
- Can ownership capabilities be created by untrusted users?
- Are best practices, as advised by the SUI team, observed?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Property-based Testing.* We also leverage property-based testing to determine if the protocol may deviate from the expected behavior. To do this, we created and modified the tests to ensure that certain properties held in the soruce code.

Scope. The scope of this audit is limited to the `sources` folder of the source code provided by the Sui Liquid Staking Protocol developers, which contains the `pool`, `ownership` and `cert` modules.

Methodology. The Veridise auditors performed a manual audit of the code assisted by property-based testing. We also developed some specifications to be checked by the move prover, however the prover failed to terminate on any of the specifications we developed, so we did not include them in this report.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, xed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
VUL-ASL-001	Unknown Investment Strategy	Medium	Acknowledged
VUL-ASL-002	No Checks on Owner Ratio Update	Low	Fixed
VUL-ASL-003	Potential for Unbacked Shares	Low	Intended Behavior
VUL-ASL-004	Restrict Ratio Updates	Low	Fixed
VUL-ASL-005	Shared Object Reduces Throughput	Warning	Intended Behavior
VUL-ASL-006	Variable Name Matches Type Name	Warning	Fixed

4.1 Detailed Description of Bugs

4.1.1 VUL-ASL-001: Unknown Investment Strategy

Severity	Medium	Commit	1
Type	Logic Error	Status	Acknowledged
Files	pool.move		
Functions	collect		

To produce yields, liquid staking protocols such as this one must invest or stake pooled funds. This protocol does so by allowing an approved operator to collect funds from the pool, invest them and then later distribute rewards to users as they unstake. While this provides the operator greater exibility, as funds can be invested anywhere, it comes at the cost of transparency and security.

```
1  public entry fun collect(pool: &mut Pool, _operatorCap: &OperatorCap, intermediary:
       address, ctx: &mut TxContext) {
2      when_not_paused(freeze(pool));
3      let amount = balance::value(&pool.pending);
4      let coin = coin::take(&mut pool.pending, amount, ctx);
5
6      event::emit(CollectedEvent {
7          to: intermediary,
8          amount,
9      });
10
11     transfer::public_transfer(coin, intermediary);
12 }
```

Snippet 4.1: Function used by an operator to collect funds for investment

Impact Such an investment method reduces transparency as it is likely that users will be unable to determine where their funds were invested to generate yields. Additionally, this approach provides weaker security guarantees as users must trust the operator to eventually distribute funds.

Recommendation Add the investment logic either to known entities or through generics to the contract so that users know how their funds will be invested.

Developer Response The liquid staking pool will stake funds with validators. To increase transparancy, though, we changed the code so that SUI is only sent to the intermediary address identi ed by the pool.

4.1.2 VUL-ASL-002: No Checks on Owner Ratio Update

Severity	Low	Commit	1
Type	Data Validation	Status	Fixed
Files	cert.move		
Functions	force_update_ratio		

The force_update_ratio function can be used by the owner of the pool to update the current value of the ratio. Unlike update_ratio, which can be used by the operator, however, there are no restrictions that are placed on how the ratio may be updated. As the ratio determines how SUI will be converted to xSUI and vice versa, an admin error can have signi cant impacts.

```
1 public entry fun force_update_ratio(metadata: &mut Metadata<CERT>, _operator_cap: &
      OwnerCap, value: u256) {
2     event::emit(RatioUpdatedEvent {
3         prevValue: metadata.ratio,
4         newValue: value
5     });
6     metadata.ratio = value;
7 }
```

Snippet 4.2: Function used by owner to set the ratio

Impact As no validation is performed on the ratio, the owner could accidentally set the ratio to:

1. A value greater than MAX_RATIO which would cause unstakes to return less SUI than what was staked
2. Zero, which would break certain aspects of the protocol until it was updated again
3. A low value which could allow a single user to drain the entire pool.

Recommendation Place restrictions on how the ratio may be updated.

4.1.3 VUL-ASL-003: Potential for Unbacked Shares

Severity	Low	Commit	1
Type	Logic Error	Status	Intended Behavior
Files	cert.move		
Functions	to_shares		

The protocol uses a ratio to determine the number of shares that corresponds to a single SUI token and vice versa. Such a conversion, however, can be subject to rounding errors that may cause users to be underpaid. To avoid the case where a user pays and gets nothing, the developers provide a special case that will round up.

```
public fun to_shares(metadata: &Metadata<CERT>, amount: u64): u64 {
    let sharesBig = (amount as u256) * metadata.ratio / MAX_RATIO;
    let shares = (sharesBig as u64);
    // avoid rounding issue, any amount should be greater than 1 share
    if (amount > 0 && shares == 0) {
        shares = 1;
    };
    shares
}
```

Snippet 4.3: Function that converts SUI to xSUI shares and rounds up if shares are 0

Impact Rounding up has the potential to cause unbacked xSUI tokens as it allows users to underpay for an xSUI. Note, the scale of this depends on the current ratio as the more it deviates from `MAX_RATIO` the more value a user would gain.

Recommendation Rather than rounding up in this case, consider instead making this can an error in `pool::stake` .

Developer Response The stake/unstake logic in the pool will reduce the likelihood of rounding via minimum stake/unstake thresholds.

4.1.4 VUL-ASL-004: Restrict Ratio Updates

Severity	Low	Commit	1
Type	Data Validation	Status	Fixed
Files	cert.move		
Functions	update_ratio		

Unlike when an owner updates the ratio of the liquid staking token, an operator must update the ratio to a non-zero value that is strictly less than the current ratio. While this restriction prevents some of the potential mistakes pointed out in this issue, it does not consider the case where an operator updates the ratio to some value that is very low.

```
1  public entry fun update_ratio(metadata: &mut Metadata<CERT>, _operator_cap: &
       OperatorCap, value: u256) {
2      assert!(value < metadata.ratio, E_RATIO_TOO_BIG);
3      assert!(value > 0, E_RATIO_TOO_SMALL);
4      // TODO: check threshold
5      event::emit(RatioUpdatedEvent {
6          prevValue: metadata.ratio,
7          newValue: value
8      });
9      metadata.ratio = value;
10 }
```

Snippet 4.4: Function used by the operator to update the ratio

Impact If the operator accidentally or maliciously sets the value to a low value, the conversion from shares to SUI can cause large amounts of SUI to be rewarded for a single xSUI.

Recommendation Either place updates to ratio on a timelock so that they may be reviewed by the owner or place additional restrictions so that the operator may only impact the ratio by a certain percentage.

4.1.5 VUL-ASL-005: Shared Object Reduces Throughput

Severity	Warning	**Commit**	1
Type	Shared Ownership	**Status**	Intended Behavior
Files	`pool.move`		
Functions	init		

The SUI ecosystem promises higher throughput by allowing transactions to be processed in parallel. As such, they recommend that whenever possible, developers use owned objects (as they allow for the best throughput) rather than shared objects such as what is used by the liquid staking pool.

```
fun init(ctx: &mut TxContext) {
    transfer::share_object(Pool {
        id: object::new(ctx),
        paused: false,
        pending: balance::zero<SUI>(),
        min_stake: 1_000_000,
        min_unstake: 1_000_000,
        unstake_queue: vector::empty<UnstakeRecord>(),
        total_unstake: 0,
    });
}
```

Snippet 4.5: Initialization of the liquid staking pool, which makes the pool a shared object

Recommendation Consider implementing the pool as an owned object rather than a shared object. Note, this isn't always possible but by implementing the protocol as an owned object, the protocol may benefit from a better user experience.

Developer Response We cannot use an owned object for the pool.

4.1.6 VUL-ASL-006: Variable Name Matches Type Name

Severity	Warning	**Commit**	1
Type	Maintainability	**Status**	Fixed
Files	`pool.move`		
Functions	UnstakeRecord		

Unlike some other ecosystems, SUI move places fewer restrictions on variable names than others. Typically, variable names are restricted so that they don't collide with certain reserved keywords like types. From the developer perspective, this can be helpful as avoiding type names improves readable and clarity.

```
1 struct UnstakeRecord has store, drop, copy {
2     address: address,
3     amount: u64,
4 }
```

Snippet 4.6: The definition of an UnstakeRecord with an element named `address`

Recommendation Do not have variable names that collide with type-names