



# Auditing Report

Hardening Blockchain Security with Formal Methods

FOR



Volo Staking V2



Veridise Inc.  
May 2, 2025

► **Prepared For:**

Navi Protocol  
<https://www.volosui.com>

► **Prepared By:**

Alberto Gonzalez  
Evgeniy Shishkin  
Aayushman Thapa

► **Contact Us:**

[contact@veridise.com](mailto:contact@veridise.com)

► **Version History:**

May. 2, 2025	V1
Apr. 24, 2025	Initial Draft

# Contents

<b>Contents</b>	<b>iii</b>
<b>1 Executive Summary</b>	<b>1</b>
<b>2 Project Dashboard</b>	<b>3</b>
<b>3 Security Assessment Goals and Scope</b>	<b>4</b>
3.1 Security Assessment Goals . . . . .	4
3.2 Security Assessment Methodology & Scope . . . . .	4
3.3 Classification of Vulnerabilities . . . . .	5
<b>4 Trust Model</b>	<b>6</b>
4.1 Operational Assumptions . . . . .	6
4.2 Privileged Roles . . . . .	6
<b>5 Vulnerability Report</b>	<b>8</b>
5.1 Detailed Description of Issues . . . . .	9
5.1.1 V-VOL-VUL-001: Single inactive validator can lead to permanent denial-of-service . . . . .	9
5.1.2 V-VOL-VUL-002: MigrationCap may be destroyed before migration process completion . . . . .	10
5.1.3 V-VOL-VUL-003: set_validator_weights does not allow for full unstaking	11
5.1.4 V-VOL-VUL-004: Migration flow does not explicitly deactivate V1 protocol	12
5.1.5 V-VOL-VUL-005: Migration setup broken by init behavior . . . . .	13
5.1.6 V-VOL-VUL-006: Outdated exchange rate returned by get_ratio . . . . .	14
5.1.7 V-VOL-VUL-007: Ambiguous names, unused code and other minor issues	15
<b>Glossary</b>	<b>17</b>



From April, 9, 2025 to April, 21, 2025, Navi Protocol engaged Veridise to conduct a security assessment of the Volo Staking V2 project. The security assessment covered the second version of the Volo staking protocol, as well as additional logic implementing the live migration process from version 1 to version 2. Veridise conducted the assessment over 27 person-days, with 3 security analysts reviewing the project over 9 days on commit ff67be4. The review strategy was a comprehensive code review conducted by Veridise security analysts.

**Project Summary.** The Volo Staking protocol is a [Liquid Staking](#) protocol targeting [Sui Network](#) validators. Users who stake SUI tokens receive a corresponding amount of vSUI tokens, which can be freely transferred. The vSUI token is yield-generating; its price is expected to increase over time as rewards are distributed to stakers by the Sui Network. When vSUI tokens are redeemed, the Volo Staking protocol returns the originally staked SUI tokens along with the accumulated rewards.

Volo Staking V2 is a revised version of the original protocol, designed to incorporate the newly implemented SIP-33 Sui Network protocol improvement. The updated protocol includes the following enhancements:

- ▶ **Instant liquidity.** SIP-33 enables instant unstaking of staked SUI. Consequently, Volo Staking V2 supports immediate unstaking, allowing all staked SUI to be redeemed at any time without an "unbonding" period. Prior to SIP-33, staked SUI could not be redeemed until the next epoch.
- ▶ **Validators weight management.** Volo Staking V2 manages staking through a validator weight mechanism. The system delegates SUI to target validators based on pre-configured weights, with the goal of minimizing the redemption of actively staked SUI.
- ▶ **Decentralized reward calculation.** The reward calculation mechanism is implemented entirely on-chain. The previous version relied on an off-chain component to determine reward amounts.
- ▶ **Fee & Reward Redesign.** Version 2 offers more flexible methods for adjusting fees, including staking fees, unstaking fees, and reward fees. Additionally, instead of directing 100% of the fees to the protocol vault, a portion of the unstaking fee can be redistributed back to the pool to generate extra yield for stakers. Furthermore, the operator can also manually deposit SUI into the pool as a boosted reward to increase APY.

**Code Assessment.** The Volo Staking V2 developers provided the source code of the Volo Staking V2 contracts for the code review. The source code appears to be mostly original code written by the Volo Staking V2 developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. To assist Veridise security analysts in understanding the code, the developers also supplied a document outlining the core behavior of both the previous and updated versions of the protocol. This document briefly describes the migration process required to transfer value from V1 to V2.

The source code contained a test suite targeting the most important usage scenarios, as well as a limited form of fuzz-testing.

**Summary of Issues Detected.** The security assessment uncovered 7 issues, 1 of which is assessed to be of critical severity. Specifically, the issue [V-VOL-VUL-001](#) could have led the protocol to be permanently in a state of denial-of-service if any one of the accepted validators switched to an inactive state. The Veridise analysts also identified 5 low-severity issues, and 1 informational finding. These included the inability to unstake from all validators simultaneously without forcing the reallocation of funds to other validators ([V-VOL-VUL-003](#)), a denial-of-service during migration due to the behavior of the init function in Sui package upgrades ([V-VOL-VUL-005](#)), and the possibility for external integrators to access an outdated exchange rate for the vSUI token ([V-VOL-VUL-006](#)).

**Recommendations.** After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Volo Staking V2 protocol.

*Improve testing.* The existing test suite currently includes ten tests (excluding fuzzing scenarios) and covers fundamental behaviors. Expanding this suite to include a broader range of scenarios—such as validator state transitions, validator additions and removals and unstaking of all the funds would help increase confidence in the protocol’s behavior under varied conditions. Additionally, it is recommended to implement negative tests, as the current suite only includes positive test cases. Incorporating negative tests would help verify the protocol’s robustness and error-handling capabilities in the presence of invalid inputs and adverse conditions.

*Migration tests.* During the analysis, attempts to simulate the migration process on the SUI Testnet indicated that certain expected behaviors were not triggered during the upgrade flow, as noted in [V-VOL-VUL-005](#). Incorporating dedicated tests for the migration path, particularly under live or near-live conditions, would help ensure that upgrade mechanisms function as intended.

*Robustness of the refresh function.* As described in [V-VOL-VUL-001](#), failures within the refresh function may render the protocol into a permanent denial of service state. Although no additional issues were identified, subjecting this function to extensive scenario-based testing would provide added assurance that it performs reliably across diverse execution paths and scenarios.

**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.

Table 2.1: Application Summary.

Name	Version	Type	Platform
Volo Staking V2	ff67be4	Move	Sui

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
April, 9–April, 21, 2025	Manual	3	27 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	0	0	0
Low-Severity Issues	5	5	4
Warning-Severity Issues	0	0	0
Informational-Severity Issues	1	1	1
TOTAL	7	7	6

Table 2.4: Category Breakdown.

Name	Number
Logic Error	3
Denial of Service	2
Access Control	1
Maintainability	1



## 3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of Volo Staking V2 smart contracts as well as the migration logic. During the assessment, the security analysts aimed to answer questions such as:

- ▶ Is the exchange rate between vSUI and SUI calculated accurately under all protocol states?
- ▶ Are all SUI and vSUI balances correctly updated during protocol operations such as minting, redeeming and epoch transitions?
- ▶ Is the exchange rate exposed externally in a way that accurately reflects the protocol's internal state?
- ▶ Can the exchange rate be manipulated through edge cases such as rounding issues or donation attacks?
- ▶ Under what conditions can execution of the refresh function fail in a way that leads to a denial-of-service for the protocol operations?
- ▶ Can an incorrect execution of the migration logic during a protocol upgrade result in loss of user or protocol funds?
- ▶ Is the code vulnerable to any commonly occurring Sui-specific security flaws?

## 3.2 Security Assessment Methodology & Scope

**Security Assessment Methodology.** To address the questions outlined above, the security assessment involved a comprehensive review conducted by human experts. Veridise security analysts examined the provided test suite and reviewed the available documentation for Volo Staking V2. The team then proceeded with an in-depth manual analysis of the codebase. Throughout the engagement, the Veridise team maintained regular communication with the Volo Staking V2 developers to clarify the intended behavior of various components and ensure accurate understanding of the protocol's design.

**Scope.** The scope of this security assessment was limited to the below files of the source code provided by the Volo Staking V2 developers:

1. sources/migration/migrate.move
2. sources/volo\_v1/\*.move
3. sources/cert.move
4. sources/fee\_config.move
5. sources/manage.move
6. sources/stake\_pool.move
7. sources/validator\_pool.move

### 3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

**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

The likelihood of a vulnerability is evaluated according to the Table 3.2.

**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

The impact of a vulnerability is evaluated according to the Table 3.3:

**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



## 4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling the security for the Volo Staking V2 protocol:

- ▶ The privileged roles responsible for package upgrades, managing the migration of funds, and other protocol operations are assumed to behave correctly. These roles and their associated permissions are discussed in detail in the following section.
- ▶ Apart from the privileged roles mentioned above, no other actors in the system are assumed to be trusted. Particularly, validators to which the protocol delegates stake are considered potentially malicious.

## 4.2 Privileged Roles

**Roles.** This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensitivity. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive *emergency* roles may be required to perform actions quickly based on real-time monitoring, while *non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assume that the role operators perform their responsibilities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- ▶ **Highly-privileged, non-emergency roles:**
  - **Upgrade Capability Holder:** This role has the authority to perform package upgrades. Possession of this capability enables full control over the contract logic, including the ability to redirect or seize user funds.
  - **Migration Capability Holder:** Temporarily granted during the migration process, this role can reallocate protocol assets. If misused, it could result in loss of funds.
- ▶ **Highly-privileged, emergency roles:**
  - **Stake Pool Admin Capability:** This role can pause protocol operations indefinitely. If triggered, staking and unstaking operations would become unavailable until manually resumed.
- ▶ **Limited-authority roles:**

- **Stake Pool Operator Capability:** This role manages the validator set and assigns validator weights. Misuse of this role could result in suboptimal delegation, potentially degrading protocol performance.

**Operational Recommendations.** The Veridise team recommends separating upgrade authorities from routine protocol management roles in the Volo Staking protocol. Specifically, the entity holding the Upgrade Capability and the temporary Migration Capability should not also operate the protocol's administrative functions, such as managing the validator set or pausing protocol operations. This separation reduces the operational exposure of highly privileged keys and limits the frequency with which they are used.

In particular:

- ▶ The Upgrade Capability should be held by a dedicated entity or multi-signature setup responsible solely for upgrade coordination.
- ▶ The Migration Capability, granted during the migration process, should be revoked immediately after completion, considering the concerns highlighted in issue [V-VOL-VUL-002](#).
- ▶ The Admin and Operator capabilities within the stake pool, which control protocol pausing and validator configuration, should be managed separately from the upgrade and migration authorities.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- ▶ Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- ▶ Using separate keys for each separate function.
- ▶ Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- ▶ Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- ▶ Validating that no party has control over multiple multi-sig keys.
- ▶ Performing regularly scheduled key rotations for high-frequency operations.
- ▶ Securely storing physical, non-digital backups for critical keys.
- ▶ Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- ▶ Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

# 5

## Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

**Table 5.1:** Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-VOL-VUL-001	Single inactive validator can lead to . . .	High	Fixed
V-VOL-VUL-002	MigrationCap may be destroyed before . . .	Low	Fixed
V-VOL-VUL-003	set_validator_weights does not allow for . . .	Low	Acknowledged
V-VOL-VUL-004	Migration flow does not explicitly . . .	Low	Fixed
V-VOL-VUL-005	Migration setup broken by init behavior	Low	Fixed
V-VOL-VUL-006	Outdated exchange rate returned by get_ratio	Low	Fixed
V-VOL-VUL-007	Ambiguous names, unused code and other . . .	Info	Fixed

## 5.1 Detailed Description of Issues

### 5.1.1 V-VOL-VUL-001: Single inactive validator can lead to permanent denial-of-service

Severity	High	Commit	ff67be4
Type	Denial of Service	Status	Fixed
File(s)	sources/validator_pool.move		
Location(s)	refresh()		
Confirmed Fix At	7035a59		

On Sui Network, validator nodes need to maintain a minimum amount of stake in order to process transactions in the next epoch. If their stake falls below the threshold, they are removed from the set of active validators by the protocol and switched to inactive state.

The `validator_pool.refresh()` function is responsible for deleting inactive validators from the internal collection of the protocol. However, there is a flaw in the current implementation: after all of the stakes have been withdrawn from an inactive validator, its weight is not set to zero, which, according to the current implementation, is necessary for the validator’s record to be deleted.

**Impact** Any attempt to stake in an inactive validator will fail. Since this happens almost every time the protocol is called, if there is anything to stake, it will enter a permanent denial of service state without any straightforward path to recovery.

**Recommendation** It is recommended to ensure that the assigned weight for inactive validators is set to 0.

**Developer Response** The developers implemented the suggested fix.

### 5.1.2 V-VOL-VUL-002: MigrationCap may be destroyed before migration process completion

Severity	Low	Commit	ff67be4
Type	Logic Error	Status	Fixed
File(s)	sources/migration/migrate.move		
Location(s)	destroy_migration_cap()		
Confirmed Fix At	1d3adc4		

The Volo V1 to V2 migration consists of the following steps:

1. Create stake pool
2. Export stakes from V1
3. Import stakes to V2
4. Destroy MigrationCap

The MigrationCap object, created in `migrate.init()`, proves the caller's authority to perform the migration. Steps 2 and 3 are handled via `migrate.export_stakes()` and `migrate.import_stakes()`, with MigrationCap destroyed in step 4 via `migrate.destroy_migration_cap()`.

However, the contract does not enforce the order of these steps, allowing MigrationCap to be deleted prematurely, which would result in incomplete migration.

**Impact** Deletion of the MigrationCap object out of order in the migration process would lead to an incomplete migration between the two versions.

**Recommendation** Implement state tracking to enforce the intended migration order, preventing accidental misuse or premature deletion of MigrationCap.

**Developer Response** The developers implemented the suggested fix.

### 5.1.3 V-VOL-VUL-003: `set_validator_weights` does not allow for full unstaking

Severity	Low	Commit	ff67be4
Type	Logic Error	Status	Acknowledged
File(s)	sources/validator_pool.move		
Location(s)	set_validator_weights()		
Confirmed Fix At	N/A		

The current `validator_set::set_validators_weights()` implementation does not allow for full unstaking: once unstaked from the current validator set, the funds must be directed to some other validators in the same call, otherwise it will abort.

**Impact** This protocol behavior may introduce safety concerns. If for some reason, the protocol needs to withdraw all funds in one step, it may not be possible to do so in a timely manner. Possible scenarios include:

- ▶ Discovered security vulnerabilities in the SUI part that puts user funds at risk
- ▶ Unexpected changes to the staking mechanism at the SUI protocol level, making it potentially unsafe

An alternative would be to urgently inform users to redeem all their vSUI coins, but this does not seem possible to do in a timely manner and should not be considered as an alternative to admin-guided unstaking.

**Recommendation** It is recommended to implement this edge case, even if this does not look relevant at the moment.

**Developer Response** The developers acknowledged the issue but do not plan to make any code changes. In the event that unstaking from all validators becomes necessary, such as in response to a severe vulnerability, the team believes the appropriate course of action would be to pause the protocol entirely.

5.1.4 V-VOL-VUL-004: Migration flow does not explicitly deactivate V1 protocol

Severity	Low	Commit	ff67be4
Type	Access Control	Status	Fixed
File(s)	sources/migration/migrate.move		
Location(s)	See description		
Confirmed Fix At	9dd3548		

When upgrading packages in Sui, it’s important to remember that previous versions remain on-chain and callable by users. Projects that support upgrades must account for this and implement logic to properly deactivate the older version. In the current implementation, the migration module in V2 does not explicitly deactivate V1 - either by pausing the protocol or migrating the NativePool object.

**Impact** If the NativePool shared object is not migrated and the V1 package is not paused, both V1 and V2 will remain active. This allows the `cert::Metadata` object to be modified through the `stake` function in both versions, which could lead to inconsistent state or unintended behavior.

**Recommendation** The migration module should explicitly handle the migration of the NativePool object and enforce the pause of the V1 instance.

**Developer Response** The developers implemented the suggested fix.

5.1.5 V-VOL-VUL-005: Migration setup broken by init behavior

Severity	Low	Commit	ff67be4
Type	Denial of Service	Status	Fixed
File(s)	sources/migration/migrate.move		
Location(s)	init()		
Confirmed Fix At	9b3c4c3		

The `migration.move()` module for the V2 upgrade relies on the `init()` function to create both the `MigrationStorage` and `MigrationCap` objects, which are required for migrating funds from the V1 instance to V2. However, according to the Sui documentation:

Any `init` functions you might include in subsequent versions of your package are ignored.

**Impact** The `MigrationStorage` and `MigrationCap` objects will not be created during the upgrade, preventing the migration process entirely. As a result, developers would need to deploy a V3 instance to recover.

**Recommendation** Replace the `init()` logic with a dedicated migration setup function (with proper access control) to create the `MigrationStorage` and `MigrationCap` objects.

**Developer Response** The developers implemented the suggested fix.

### 5.1.6 V-VOL-VUL-006: Outdated exchange rate returned by `get_ratio`

Severity	Low	Commit	ff67be4
Type	Logic Error	Status	Fixed
File(s)	sources/stake_pool.move		
Location(s)	get_ratio()		
Confirmed Fix At	62debde		

The `get_ratio()` function in the `stake_pool.move()` module is responsible for calculating the exchange rate between vSUI and SUI. However, this function may return an outdated value if the `refresh()` function has not been called during the current epoch. This is because `refresh` is responsible for updating the state of the `StakePool` object, including the total supply of SUI, which directly affects the exchange rate calculation.

**Impact** The outdated exchange rate can lead to incorrect valuations of vSUI tokens. This is particularly problematic for external integrators, such as lending protocols, that rely on the value of vSUI tokens as collateral. If the exchange rate is not accurate, it could result in mispricing of collateral.

**Recommendation** Clearly document the behavior of the `get_ratio()` function, emphasizing that it may return outdated values if `refresh` has not been called. This will help external integrators understand the potential risks and take necessary precautions.

**Developer Response** The developers implemented the suggested fix.

### 5.1.7 V-VOL-VUL-007: Ambiguous names, unused code and other minor issues

Severity	Info	Commit	ff67be4
Type	Maintainability	Status	Fixed
File(s)	sources/fee_config.move, sources/validator_pool, sources/migration/migrate.move		
Location(s)	See issue description		
Confirmed Fix At	b3ef7f2		

#### Description

- The following program constructs are currently unused:
  - sources/fee\_config.move:
    - FeeUpdateEvent
  - sources/migration/migrate.move :
    - migration\_storage.exported\_count
  - sources/validator\_pool.move:
    - validatorInfo.last\_refresh\_epoch
- Some variable and function names are unclear or misrepresent their actual purpose, which could lead to confusion or incorrect assumptions:
  - sources/stake\_pool.move:
    - const DECIMALS : This implies it holds a decimal count, but it actually represents a full token amount with decimals included. This is particularly misleading given the presence of another constant named DECIMALS (in the cert module) with a value of 9.
  - sources/manage.move:
    - manage:check\_paused(): Despite its name, this function actually verifies that the system is **not paused**. A clearer name would be check\_not\_paused() or ensure\_not\_paused() .
- The code base has some hardcoded values that should be declared as constants:
  - fee\_config.move:
    - Both calculate\_stake\_fee() and calculate\_unstake\_fee() use the hardcoded value of 9999 which actually depends on the BPS\_MULTIPLIER value. This value should be moved into a special constant computed as:  $BPS\_MULTIPLIER - 1$ .
- Unreachable code paths:
  - The join\_stake function handles two cases for the passed StakedSui object:
    - If the stake is already activated, it converts it to FungibleStakedSui.
    - If the activation epoch is still in the future, it stores it in inactive\_stake.

However, join\_stake is only invoked via increase\_validator\_stake, which always passes a freshly created StakedSui object with an activation epoch set to  $current\_epoch + 1$ . As a result, the first code path (a) is effectively unreachable.

**Impact** The identified issues - including unused constructs, unclear or misleading naming, hardcoded values, and unreachable code paths - all reduce the overall maintainability and clarity of the codebase.

**Recommendation** To improve the maintainability of the codebase:

- ▶ Remove unused constructs to prevent potential inconsistencies and reduce maintenance overhead.
- ▶ Refactor misleading names to accurately reflect their purpose and behavior, reducing the risk of misunderstandings.
- ▶ Extract hardcoded values into clearly named constants to improve readability and make future adjustments less error-prone.
- ▶ Eliminate or review unreachable code paths to ensure the codebase remains clean and only contains logic that can be executed in practice.

**Developer Response** The developers implemented the suggested fixes.



## Glossary

**Liquid Staking** A tokenized contract representing shares of a staked native currency. See <https://chain.link/education-hub/liquid-staking> to learn more. 1

**Sui Network** Sui Network is a Layer 1 blockchain designed for near-instant and low-cost transactions. Sui's main innovation lies in its unique object-centric data model and the use of the Move programming language. See the official website at <https://sui.io>. 1

**vSUI** A token used to represent staked Sui tokens in the Volo protocol. For more information, see <https://www.volosui.com>. 1