# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** VOLO
**Date:** 22 Sep, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for VOLO |
| **Approved By** | Luciano Ciattaglia \| Director of Services at Hacken OÜ |
| **Auditor** | Jakub Heba |
| **Auditor** | Vladyslav Khomenko |
| **Type** | Liquid Staking |
| **Platform** | Sui |
| **Language** | Move |
| **Methodology** | Link |
| **Website** | volo.fi |
| **Changelog** | 22.08.2023 - Initial Review<br>22.09.2023 - Second Review |

# Table of Contents

# Introduction

Hacken OÜ (Consultant) was contracted by VOLO (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

# System Overview

The VOLO Liquid Staking is a protocol that allows users to exchange the *SUI* tokens for special *voloSUI* tokens that could be used in VOLO Ecosystem and other DeFi protocols partnered with VOLO Protocol.

The *SUI* tokens are staked using the native *sui_system* module, whose implementation is **out of the audit scope**.

Stakers are able to request exchanging their *voloSUI* for *SUI* with rewards accrued, unstaked funds will be returned in the current or the next epoch.

There are several smart contracts in the audit scope:

- *ownership* – manages owner and operator capabilities.
- *cert* – *voloSUI* token contract, stores and updates *SUI* to *voloSUI* exchange rate.
- *native_pool* – contract allows exchange *SUI* for *voloSUI* and requests to exchange it back at a possibly better rate.
- *math* – math utility contract.
- *validator_set* – a contract that manages validators.
- *unstake_ticket* – a contract that handles tickets, which serve as proof of unstaking, while waiting to exchange tokens.

## Roles

The Owner and Operator are able to transfer their permission independently of each other.

*cert*:

- Owner – Update the contract and migrate the associated objects to it.

*ownership*:

- Owner – Transfer the owner role to another address.
- Operator - Transfer operator role to another address.

*native_pool*:

- Owner - Change min stake amount.
- Owner - Change unstake fee threshold (not more than 100%).
- Owner - Change base unstake fee (not more than 100%).
- Owner - Change base reward fee (not more than 100%).
- Owner - Update rewards threshold.
- Owner - Can withdraw fees from the contract.

- Owner - Can pause or unpause contract. In the paused contract it is not possible for the owner to withdraw fees, sort validators, burn tickets to release unstaking, stake SUI and update rewards.
- Owner — Update the contract and migrate the associated objects to it.
- Operator — Add new validators with specified priorities.
- Operator - Can increase the ratio of reward tokens.

# Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

## Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Public documentation provides the system basics.
- Internal documentation provides valuable insights into the system architecture and general interaction flow.
- The configuration instructions are insufficient.

## Code quality

The total Code Quality score is **9** out of **10**.

- The code is well-written and designed.
- The code contains unused variables.
- Contradiction in variable naming exists.

## Test coverage

The code coverage of the project is **82.16%**.

## Security score

As a result of the audit, the code contains **1** medium, and **1** low severity issues. The security score is **9** out of **10**.

All found issues are displayed in the Findings section of the report.

## Summary

According to the assessment, the Customer's smart contract has the following score: **8.5**.

The system users should acknowledge all the risks summed up in the Risks section of the report.

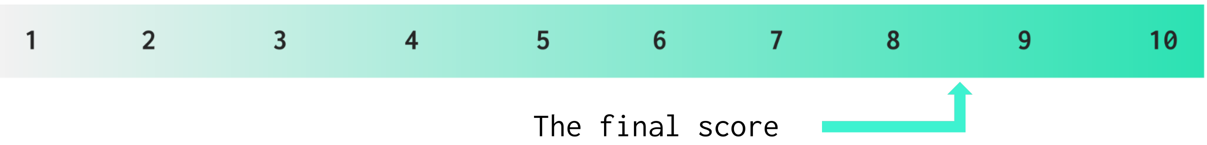

***Table. The distribution of issues during the audit***

| **Review date** | Low | Medium | High | Critical |
|---|---|---|---|---|
| 22 Aug 2023 | 2 | 1 | 2 | 0 |
| 22 Sep 2023 | 1 | 1 | 0 | 0 |

## Risks

- Users are unable to withdraw their funds immediately. They burn their *voloSUI* and register in a queue to get *SUI* in return.
- The smart contracts system is designed to be upgradeable. The owner may change the system logic in the future.
- The owner is able to pause all the pool functionality.
- The owner may set arbitrary minimum thresholds for stake and unstake amounts.

## Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

| Item | Description | Status |
|---|---|---|
| **Integer Overflow and Underflow** | All math operations should be safe from overflows and underflows. | Passed |
| **Outdated Compiler Version** | It is recommended to use a recent version of the Move compiler. | Passed |
| **Access Control & Authorization** | Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users. | Passed |
| **DoS (Denial of Service)** | Execution of the code should never be blocked by a specific contract state unless required. | Passed |
| **Race Conditions** | Race Conditions and Transactions Order Dependency should not be possible. | Passed |
| **Block values as a proxy for time** | Block numbers should not be used for time calculations. | Not Relevant |
| **Signature Reuse** | Signed messages that represent an approval of an action should not be reusable. | Not Relevant |
| **Weak Sources of Randomness** | Random values should never be generated from Chain Attributes or be predictable. | Not Relevant |
| **Calls Only to Trusted Addresses** | All external calls should be performed only to trusted addresses. | Passed |
| **Presence of Unused Variables** | The code should not contain unused variables if this is not justified by design. | Failed (L01) |
| **Assets Integrity** | Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract. | Passed |
| **User Balances Manipulation** | Contract owners or any other third party should not be able to access funds belonging to users. | Passed |
| **Data Consistency** | Smart contract data should be consistent all over the data flow. | Passed |
| **Flashloan Attack** | When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. | Not Relevant |

| | | |
|---|---|---|
| **Token Supply Manipulation** | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer. | Passed |
| **Gas Limit and Loops** | Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit. | Failed<br>(M01) |
| **Compiler Warnings** | The code should not force the compiler to throw warnings. | Passed |
| **Style Guide Violation** | Style guides and best practices should be followed. | Passed |
| **Requirements Compliance** | The code should be compliant with the requirements provided by the Customer. | Passed |
| **Environment Consistency** | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Passed |
| **Secure Oracles Usage** | The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles. | Not Relevant |
| **Tests Coverage** | The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. The usage of contracts by multiple users should be tested. | Failed |
| **Stable Imports** | The code should not reference draft contracts, which may be changed in the future. | Passed |

## Findings

### ■■■■ Critical

No critical severity issues were found.

### ■■■ High

#### H01. Invalid Calculations; Data Consistency

| Impact | High |
|---|---|
| Likelihood | Medium |

During the *remove_stakes* loop, the system is designed to repeatedly process withdrawals from a vault until the *total_withdrawn* meets or exceeds the *requested_amount* or under certain conditions related to the vault's *gap*. However, there is a flaw in the current implementation. The *requested_amount* is not updated to reflect the amount already withdrawn, which leads to potential over-withdrawals or inconsistencies in the withdrawal amounts.

**Path:** ./liquid_staking/sources/validator_set.move: remove_stakes(..)

**Recommendation:** Adjust the *requested_amount* to reflect the new remaining amount that needs to be withdrawn after each successful stake processing.

**Found in:** 6662d76

**Status:** Fixed (Revised commit: d088758)

#### H02. Requirements Violation; Data Consistency

| Impact | High |
|---|---|
| Likelihood | Medium |

The *sort_validators* function is designed to sort validators in descending order based on their priorities. However, the current logic does not consistently achieve this objective. The existing sorting mechanism allows insertion in the middle of the array only when the priorities are not greater than the size of the validator's array. This condition is not guaranteed to be met. Consequently, if the priorities are invariably larger than the size of the validator's array, validators will be consistently added to the initial position, neglecting the intended order based on priorities.

**Path:** ./liquid_staking/sources/validator_set.move: sort_validators(..)

**Recommendation:** Revise the sorting logic to handle all possible ranges of priorities, ensuring they are placed in the correct position regardless of the size of the validator's array.

**Found in:** 6662d76

**Status:** Fixed (Revised commit: d088758)

## ■■ Medium

### M01. Denial Of Service & Inefficient Gas Model

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | Low |

The operator of the *native_pool* has the authority to introduce new validators to the contract for distributing stakes. However, since there is no upper limit on the number of validators, certain contract features (such as sorting validators) might become non-functional due to excessive Gas consumption, especially when iterating through a large list of validators.

**Path:** ./liquid_staking/sources/native_pool.move:
update_validators(..)

**Recommendation:** Set a maximum limit on the total number of supported validators using a constant and restrict the addition of validators beyond this specified number. Currently, there's only restrictions on how many validators can be updated/added in one call.

**Found in:** 6662d76

**Status:** Reported

## ■ Low

### L01. Unused Variables/Structs

Unused variables and structs should be removed from the contracts. Although unused variables and structs are allowed in Move and do not pose a direct security issue, it is best practice to avoid them as they can cause an increase in computation (and unnecessary Gas consumption) and decrease the code readability.

**Paths:**

- ./liquid_staking/sources/native_pool.move: TicketMintedEvent, TicketBurnedEvent;
- ./liquid_staking/sources/validator_set.move: VERSION;

**Recommendation:** Remove unused variables/structs.

**Found in:** 6662d76

**Status:** Reported

### L02. Missing Event Emit

The functions are considered to perform valuable configuration changes, which users should be notified about.

**Path:**

- ./liquid_staking/sources/native_pool.move: update_validators(..)

**Recommendation:** Implement and emit corresponding events to notify users about the changes.

**Found in:** 6662d76

**Status:** Fixed (Revised commit: d088758)

## Informational

### I01. Contradiction

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Low |

The *collect_fee* function incorrectly labels the *OwnerCap* variable as *_operator_cap*, which can cause confusion and misunderstanding.

**Path:** ./liquid_staking/sources/native_pool.move: collect_fee(..);

**Recommendation:** Make sure that function should be available for Owner (not Operator) and rename the variable.

**Found in:** 6662d76

**Status:** Fixed (Revised commit: d088758)

# Disclaimers

## Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

## Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

# Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

| Risk Level | High Impact | Medium Impact | Low Impact |
|---|---|---|---|
| **High Likelihood** | Critical | High | Medium |
| **Medium Likelihood** | High | Medium | Low |
| **Low Likelihood** | Medium | Low | Low |

## Risk Levels

**Critical:** Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

**High:** High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

**Medium:** Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

**Low:** Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

## Impact Levels

**High Impact:** Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

**Medium Impact:** Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

**Low Impact:** Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

## Likelihood Levels

**High Likelihood:** Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

**Medium Likelihood:** Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

**Low Likelihood:** Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

## Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

## Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

### Initial review scope

| | |
|---|---|
| **Repository** | https://github.com/Ankr-network/stakefi-sui-smart-contract |
| **Commit** | 6662d76109670a458cde7f739938b203cf183780 |
| **Requirements** | VOLO Liquid Staking |
| **Contracts** | File: cert.move<br>SHA3: 936fbe01f122ee82e3153cb2163659ac00c4156e9111b7d00ab1cd5963ee8a33<br>File: math.move<br>SHA3: a3b3d370c221acba91d02cd2c83c9e62a88d0f71a9e652d0383fd6a781759c14<br>File: native_pool.move<br>SHA3: b7287d6c7455e7c4923a01bf0842e399a511ad003b461b7e35409251acfec3d1<br>File: ownership.move<br>SHA3: 575331cea032b4fca206721b7bf61c4af7654284b55286385f123c0061d10c82<br>File: unstake_ticket.move<br>SHA3: d7fce315dc1bc5cbe52b2fec5b48dd399e9c75a70ee7994e7136c26e2d5c375b<br>File: validator_set.move<br>SHA3: d0b56628c3cf2f11a91bec5f23fc6037623b1a5284aa40e24fb2bb40c283a469 |

### Second review scope

| | |
|---|---|
| **Repository** | https://github.com/Sui-Volo/volo-liquid-staking-contracts |
| **Commit** | d088758139f34f27a2acf65cdc3e1f89dfcd6596 |
| **Requirements** | VOLO Liquid Staking |
| **Contracts** | File: liquid_staking/sources/cert.move<br>SHA3: e17d707a1dc1edb6e0db8940cf766add7b1621a8a1c0bc0c81a6565e0b8823fa<br>File: liquid_staking/sources/math.move<br>SHA3: a3b3d370c221acba91d02cd2c83c9e62a88d0f71a9e652d0383fd6a781759c14<br>File: liquid_staking/sources/native_pool.move<br>SHA3: 756c319ee895fdf8039d4364184d3fac2a08af8fd4c30a6ef3600ee8dcb24103<br>File: liquid_staking/sources/ownership.move<br>SHA3: 575331cea032b4fca206721b7bf61c4af7654284b55286385f123c0061d10c82<br>File: liquid_staking/sources/unstake_ticket.move<br>SHA3: 2c83a7c9b89e42f2dfc141803846bc9a465fb34847ea2cd39bd33ac5627c449f<br>File: liquid_staking/sources/validator_set.move<br>SHA3: f342b6303c17a9959d3637b22cefa20289e3b5ab25032fc0f0e37cc4525ae712 |