Haedal Protocol - audit

Security Assessment

CertiK Assessed on Oct 23rd, 2025

CertiK Assessed on Oct 23rd, 2025

Haedal Protocol - audit

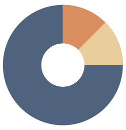
The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Sui (SUI)	Manual Review, Static Analysis

LANGUAGE	TIMELINE
Move	Preliminary comments published on 03/13/2025 Final report published on 10/23/2025

Vulnerability Summary



8	1	0	7	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

🟧 1	Centralization	1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
🟨 0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 1	Minor	1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 6	Informational	1 Resolved, 5 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | HAEDAL PROTOCOL - AUDIT

CODEBASE | HAEDAL PROTOCOL - AUDIT

Repository

base

Commit

79b5265f2dc302821541e87ff4287d7a2209711f

AUDIT SCOPE | HAEDAL PROTOCOL - AUDIT

haedallsd/haedal-protocol
sources/config.move
sources/manage.move
sources/operate.move
sources/staking.move
sources/hasui.move
sources/interface.move
sources/table_queue.move
sources/util.move
sources/vault.move

APPROACH & METHODS | HAEDAL PROTOCOL - AUDIT

This report has been prepared for Haedal to discover issues and vulnerabilities in the source code of the Haedal Protocol - audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | HAEDAL PROTOCOL - AUDIT

Overview

The **Haedal Protocol** is a liquid staking protocol built on Sui that allows anyone to stake their SUI tokens to contribute to governance and decentralisation of the Sui blockchain.

External Dependencies

The project is developed using the Move language and running on the top of the Sui blockchain. The vulnerability and the updates of the language/Sui framework may affect the project as a whole. As the Sui network is rapidly evolving, to avoid any potential compatibility issues and take advantage of new features and improvements, the client should upgrade the Sui framework to the most recent version. Additionally, staying informed about any upcoming updates or changes to the language or framework can help ensure the project remains secure and compatible.

Dependency of the **Haedal Protocol**:

```
Sui = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework/packages/sui-framework", rev = "mainnet" }
SuiSystem = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework/packages/sui-system", rev = "mainnet" }
```

Also, the Haedal Protocol relies on the native staking system in Sui for liquid staking and assumes that the off-chain operations, such as reward update or operator staking, are processed as expected.

The above dependencies are not within the current audit scope and serve as a black box. Modules/Contracts within the module are assumed to be valid and non-vulnerable actors in this audit and implement proper logic to collaborate with the current project and other modules.

Privileged Roles

To set up the project correctly and ensure that the project functions properly, owners of the following objects are able to use privileged functions, more details in **GLOBAL-01: Centralization Related Risks And Upgradability**.

The advantage of the privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to serve the community best. It is also worthy of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the key pairs of privileged accounts are compromised, the project could have devastating consequences.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Furthermore, any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

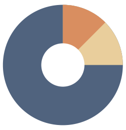
Upgradeability

Developers working with the Sui blockchain have the ability to upgrade packages based on their software iteration requirements. However, this also means that the `UpgradeCap` and publisher's key store should be handled with caution to prevent any unexpected loss. Additionally, it is important to inform the community about any upgrade plans to address concerns related to centralization and ensure transparency.

Reference:

- Sui Package Upgrades
- Custom Policies

FINDINGS | HAEDAL PROTOCOL - AUDIT



Total Findings	Critical	Centralization	Major	Medium	Minor	Informational
8	0	1	0	0	1	6

This report has been prepared for Haedal to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 8 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
SOU-01	**Centralization Related Risks And Upgradability**	**Centralization**	**Centralization**	● **Acknowledged**
STA-06	Discussion On Rewards Update	Design Issue	Minor	● Acknowledged
CON-01	Potentially Unreasonable Fee Settings In `config` Module	Logical Issue	Informational	● Acknowledged
STA-01	Confirmation On Token Source Of `protocol_sui_vault` And `claim_sui_vault`	Design Issue	Informational	● Resolved
STA-02	Discussion On `withdraw_sui`	Volatile Code	Informational	● Acknowledged
STA-04	Discussion On Sui Stake Mechanism	Design Issue	Informational	● Acknowledged
STA-07	Potential Missing Validation On Validators	Volatile Code	Informational	● Acknowledged
STA-08	Potential Manipulation Of SUI And HaSUI Balances	Design Issue	Informational	● Acknowledged

SOU-01 | Centralization Related Risks And Upgradability

Category	Severity	Location	Status
Centralization	● **Centralization**	**sources/manage.move: 49, 56, 61, 66, 71, 76, 81, 86, 92, 96, 100, 105, 110, 115, 120, 128~134, 141, 147, 154~160, 166~172, 181, 188~195; sources/operate.move: 9, 14, 19, 26~32, 39, 45, 52~58, 64~70, 79, 86~93, 98**	● **Acknowledged**

Description

In the module **manage**, the role **AdminCap** has authority over the functions:

- initialize(): initialize staking contract;
- set_deposit_fee(): manage deposit fee;
- set_reward_fee(): manage reward fee;
- set_validator_reward_fee(): manage validator reward fee;
- set_service_fee(): manage service fee;
- set_withdraw_time_limit(): manage `withdraw_time_limit` ;
- set_validator_count(): manage `validator_coun` ;
- sort_validators(): resort the order of validators;
- migrate(): migrate to a new staking version;
- collect_rewards_fee(): abandoned;
- collect_rewards_fee_v2(): collect the reward fee in the staking contract;
- collect_service_fee(): collect the service fee in the staking contract;
- toggle_stake(): pause/unpause the stake operation;
- toggle_unstake(): pause/unpause the unstake operation;
- toggle_claim(): pause/unpause the claim operation;
- do_stake(): stake SUI to validators;
- update_total_rewards_onchain(): update the staking rewards;
- unstake_inactive_validators(): unstake SUI from inactive validators;
- do_unstake_onchain(): unstake SUI from inactive validators, the input `validators` is unused;
- unstake_pools(): unstake SUI from the input `validators` ;
- update_validator_rewards(): update staking rewards of the input `validator` ;
- unstake_from_validator(): unstake from the input `validator` ;

Any compromise to the **AdminCap** account may allow a hacker to take advantage of this authority, upgrade protocol, pause/unpause the contract, and manipulate the operator role list.

In the module **operate**, the role **OperatorCap** has authority over the functions:

- toggle_stake(): pause/unpause the stake operation;
- toggle_unstake(): pause/unpause the unstake operation;
- toggle_claim(): pause/unpause the claim operation;
- do_stake(): stake SUI to validators;
- update_total_rewards_onchain(): update the staking rewards;
- unstake_inactive_validators(): unstake SUI from inactive validators;
- do_unstake_onchain(): unstake SUI from inactive validators, the input `validators` is unused;
- unstake_pools(): unstake SUI from the input `validators` ;
- update_validator_rewards(): update staking rewards of the input `validator` ;
- unstake_from_validator(): unstake from the input `validator` ;
- sort_validators(): resort the order of validators;

Any compromise to the **OperatorCap** account may allow a hacker to take advantage of this authority, create new strategies, update the fee of strategies and withdraw fee of strategies.

In addition, developers working with the Sui blockchain can upgrade packages based on their software iteration requirements. However, this also means that the `UpgradeCap` and deployer's key store should be handled carefully to prevent any unexpected losses. It is important to inform the community about any upgrade plans to address concerns related to centralization and ensure transparency.

More information can be found:

- Sui Package Upgrades
- Third-Party Package upgrades

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Haedal, 07/22/2025]: We use multi-signature contracts to manage the operations of various privileged roles, and have introduced Sui and Mysten Foundation to hold multi-signatures.

[CertiK, 07/22/2025]: The finding will be updated when corresponding multi-sig information is provided. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

STA-06 | Discussion On Rewards Update

Category	Severity	Location	Status
Design Issue	● Minor	sources/staking.move: 571, 608	● Acknowledged

Description

The liquidity staking protocol has two functions for updating potential rewards from native staking:

- `update_validator_rewards` : Updates rewards for a single validator.
- `update_total_rewards_onchain` : Updates rewards for all validators.

Currently, these functions are managed by a privileged role, either a manager or operator.

Concerns raised include:

1. If the privileged roles fail to execute these functions, the staking or unstaking results may be inaccurate.
2. If rewards are updated for only part of validators using `update_validator_rewards` , the staking or unstaking results may also be inaccurate.
3. The user may experience a race condition if they stake/unstake before/after the reward update.

Also, if the `unstake_inactive_validators` not invoked frequently, the `staking.validators` may be fully occupied due to the size limit on vector, ref: https://docs.sui.io/guides/developer/dev-cheat-sheet

Recommendation

We would like to check with the team about more insight into the reward update workflow.

Alleviation

[Haedal, 07/22/2025]: The rewards update flow is:

1. at the end of every epoch, call `do_stake` to deposit all the SUI collected to Sui validators

2. at the begining of next epoch, call `update_validator_rewards` for all the validators, these will be finished in several minutes.

3. call `unstake_inactive_validators` to withdraw SUI from inactive validators, and wait to call next `do_stake` to deposit to other active ones.

Haedal will retry several times when calling `update_validator_rewards` fails. Even if some of the `update_validator_rewards` still fail, the rewards may be inaccurated in the current epoch, but will be recovered by manual or wait to the next epoch begins calling again. The impact on the user is very low.

[CertiK, 08/26/2025]: According to on-chain activities from the admin address:

https://suiscan.xyz/mainnet/account/0x5b476896be81dc47a0599e8920ced58ebe16ed6210b73e04f71db0f164d5ab41/tx-blocks

We observed that `update_validator_rewards()` is not invoked while the protocol is in a paused state, despite the intended process described in the comment:

```
    /// At the begining of every epoch, do below:
    /// 1. pause claim/stake/unstake
    /// 2. call `update_validator_rewards` for every validator separately(to avoid
abort for update all the validators at a time like update_total_rewards_onchain)
    /// 3. resume claim/stake/unstake
```

This raises several concerns if users perform stake/unstake/claim operations between the start of a new epoch and the reward updates:

1. Stale Reward Accounting Enables Excessive Minting

The `request_stake_coin` function calculates the amount of `haSUI` to mint based on the outdated `staking.total_rewards` value. This value is only refreshed when `update_total_rewards_onchain()` and `update_validator_rewards()` are called. If these update functions are not invoked in a timely manner, it can result in over-minting.

As a result, staking operations performed before reward updates will allow new users to receive inflated `haSUI` amounts, effectively diluting the share of existing holders.

2. Protocol Fee Bypass Through Premature Unstaking

The `claim_coin_v2()` function allows validator unstaking without updating rewards or uncollected protocol fees, effectively bypassing fee accounting.

If unstaking operations are executed before reward updates in the current epoch, users can bypass the configured reward fee rates, leading to a permanent loss of protocol revenue.

3. Inaccurate Reward Update

The `update_total_rewards_onchain()` and `update_validator_rewards()` functions call `calculate_validator_pool_rewards_increase()` to obtain `total_rewards_increased`, which is then used to update both `staking.uncollected_protocol_fees` and `staking.total_rewards`.

Within `calculate_validator_pool_rewards_increase()`, the return value `pool_rewards_increased` is derived from the mutable `pool.rewards` baseline:

```
if (pool_rewards > pool.rewards) {
    pool_rewards_increased = pool_rewards - pool.rewards;
    pool.rewards = pool_rewards;
};
```

However, the `pool.rewards` value can be reduced by unstaking operations through `do_validator_unstake()`. When `do_validator_unstake()` is invoked before rewards are updated for the epoch, a portion of the rewards may be distributed or withdrawn without these amounts being added to `staking.total_rewards` while unstake from native staking system, resulting in subsequent users receiving more minted tokens than intended due to an inaccurate exchange rate.

If rewards have not been updated for a long time, such as multiple epochs, and users withdraw before the update, the protocol can both double-count rewards for remaining stakes and fail to account for withdrawn rewards.

- On withdraw: rewards are paid out but not added to `staking.total_rewards`. `pool.rewards` is reduced by `withdraw_rewards` (or set to 0 if insufficient).
- On subsequent update: `increment = pool_rewards - pool.rewards`. With `pool.rewards == 0`, the update re-credits all remaining stake's historical + current rewards, including portions already credited in past updates -> double-count for remaining stakes.

4. Unaccounted Reward

Unstaking operations capture the current rewards but do not add them to `staking.total_rewards`.

The `staking.total_rewards` value is only updated through the following call chain: `update_validator_rewards()` / `update_total_rewards_onchain()` → `calculate_validator_pool_rewards_increase()` → `calculate_staked_sui_rewards()`

In contrast, `do_validator_unstake()` calls `withdraw_staked_sui()`, which also invokes `calculate_staked_sui_rewards()` to compute staked rewards. However, this computed value is not used to update `staking.total_rewards`.

As a result, the protocol's total value is underestimated, leading to an artificially deflated `haSUI` exchange rate.

Root Cause

All vulnerabilities stem from **asynchronous reward update mechanisms** where `pool.rewards` and `staking.total_rewards` serves as stale caches that fail to accurately reflect real-time validator pool states. This creates systemic flaws across reward distribution, fee collection, and exchange rate calculations, allowing users to exploit timing windows between reward accrual and protocol accounting updates.

Enforce a strict pause on all user operations between epoch start and reward updates, or add validation logic (e.g., checking `staking.rewards_last_updated_epoch` against the current epoch in each operation) to ensure reward data is up to date before proceeding.

[Haedal, 09/02/2025]:

During our development process, we have carefully considered these questions, and we believe that none of them would affect the correct update of the protocol's rewards or exchange rate.

Below is our explanation. Please feel free to verify based on this.

1. This is not an issue.

Although a new epoch has started, any staking that occurred before the reward and exchange rate update in **Haedal** can still be considered as part of the previous epoch. These operations will be processed using the exchange rate from the **previous epoch**, so **over-minting does not occur**.

2. This is not an issue.

During the new epoch, before the rewards and exchange rates are updated, if a user triggers the call chain

`claim_coin_v2 --> withdraw_sui --> do_validator_unstake --> withdraw_staked_sui` ,

rewards will be withdrawn from the validator. However, **this does not affect the exchange rate update** in any way.

Let's assume the following example for clarity:

- There is only **one validator** node.
- Only **1 StakedSui** exists, the principal is **1000**.
- Rewards accrued **up to yesterday** are **200**.
- Rewards from **yesterday to today** are **100**.
- So, at the start of the new epoch, the total value is:

`1000 (principal) + 200 (past rewards) + 100 (new rewards) = 1300` .

Now, **before** the exchange rate is updated, suppose a user calls `claim_coin_v2` to withdraw **400**.

In the function `get_split_amount` , the principal to withdraw is calculated as:

```
let withdraw_principal = util::mul_div(need_amount, principal,
staked_sui_with_rewards) + 1;
```

Which gives: `withdraw_principal = 400 * 1000 / 1300 = 307.69230769`

`left_principal = 1000 - withdraw_principal = 692.30769231`

In `do_validator_unstake` , the rewards associated with the withdrawn principal are calculated:

```
let (withdraw_principal, withdraw_rewards) = withdraw_staked_sui(wrapper, withdraw,
unstaked_bal, ctx);
```

Then: `withdraw_rewards = 307.69230769 / 1000 * 100 = 30.76923077`

These rewards are then deducted from the pool (both past rewards and yet-to-be-updated rewards), so that in the next exchange update, the system can properly calculate **incremental rewards** based on the reduced principal and remaining rewards.

```
pool.total_staked = pool.total_staked - withdraw_principal;
if (pool.rewards >= withdraw_rewards) {
pool.rewards = pool.rewards - withdraw_rewards;
} else {

pool.rewards = 0;
}
```

> Note: The value deducted from `pool.rewards` includes both past and current (unrealized) rewards.

> In rare edge cases where `pool.rewards` hasn't been updated for a long time and is very outdated,

> `withdraw_rewards` might exceed `pool.rewards` . However, **this scenario is nearly impossible in practice**.

Finally, when updating rewards, we follow the reward and exchange rate update call chain: `update_validator_rewards --> calculate_validator_pool_rewards_increase --> calculate_staked_sui_rewards`

In the `calculate_validator_pool_rewards_increase` function, the `pool_rewards` accumulated by each staking validator are calculated based on **all the StakedSui rewards accrued from the time of staking until now**.

Since the function `do_validator_unstake` (called during the `claim_coin_v2` process) has already **split the StakedSui and reduced `pool.rewards`** , at this point:

- Both `pool_rewards` and `pool.rewards` will be **smaller** than the values during the `claim_coin_v2` call.
- However, `pool_rewards` will always remain **greater than or equal to** `pool.rewards` .

This discrepancy is **expected and properly handled** by the system.

At this point:

- The total remaining principal is: `692.30769231`
- The incremental rewards from yesterday to today are:

`692.30769231 / 1000 * 100 = 69.23076923`

This will be processed correctly during the reward update.

3. Not an issue either, same principle as point 2.

4. Also not an issue, same principle as point 2.

CON-01 | Potentially Unreasonable Fee Settings In `config` Module

Category	Severity	Location	Status
Logical Issue	● Informational	sources/config.move: 65, 77, 89, 100	● Acknowledged

Description

The `config` module of the contract allows the management of various fee settings, including `deposit_fee` , `reward_fee` , `service_fee` , and `validator_reward_fee` . It has been observed that `deposit_fee` , `reward_fee` , and `service_fee` can be set as high as 100%. Additionally, `validator_reward_fee` is not capped, allowing it to be set to an unlimited value. This configuration poses a risk of setting fees to unreasonable levels, potentially impacting user trust and contract functionality.

Recommendation

We recommend adding a reasonable cap for all the fees.

Alleviation

[Haedal, 07/22/2025]: The current agreement does not charge `deposit_fee` and `validator_reward_fee` , while `reward_fee` and `service_fee` are very low. Currently, they are adjusted offline based on market conditions.

STA-01 | Confirmation On Token Source Of `protocol_sui_vault` And `claim_sui_vault`

Category	Severity	Location	Status
Design Issue	● Informational	sources/staking.move: 472, 979	● Resolved

Description

In the `collect_rewards_fee_v2` function, the contract initiates a withdrawal from the `protocol_sui_vault` before attempting to cover `staking.uncollected_protocol_fees` . If the vault's balance is insufficient, the contract calls the `withdraw_sui` function, which first withdraws from `sui_vault` and then unstakes from validators if necessary. However, it appears that no tokens are currently being deposited into the `protocol_sui_vault` within the staking contract, rendering the initial withdrawal operation redundant. The relevant code snippet is as follows:

```
        let (bal, need_amount) = vault::withdraw_max(&mut
staking.protocol_sui_vault, staking.uncollected_protocol_fees);
        balance::join(&mut bal, withdraw_sui(wrapper, staking, need_amount, ctx));
```

The `claim_sui_vault` vault exhibits a similar issue.

Recommendation

This raises questions about whether the current implementation aligns with the intended design and the source of tokens for these vaults.

Alleviation

[Haedal, 07/22/2025]: This is caused by the contract upgrade. Initially, the funds were stored in `protocol_sui_vault` and `claim_sui_vault` . After the protocol upgrade, they were all obtained from `withdraw_sui` .

STA-02 | Discussion On `withdraw_sui`

Category	Severity	Location	Status
Volatile Code	● Informational	sources/staking.move: 487~488	● Acknowledged

Description

This function first attempts to withdraw SUI from the `sui_vault` . If the vault lacks sufficient funds, it will unstake SUI from validators to cover the shortfall.

If claim or unstake operations cannot secure enough SUI from their own vaults, they will invoke the `withdraw_sui` function to compensate for the deficit. Below are the functions that call `withdraw_sui` :

- **`request_unstake_instant_coin`** : Users instantly unstake `HASUI` .
- **`claim_coin_v2`** : Users claim unstaked SUI after calling `request_unstake_delay` .
- **`collect_rewards_fee_v2`** : Admin collects reward fees.

There is no explicit token inflow into `sui_vault` . If a user stakes SUI with inactive validators, the SUI is deposited into the `sui_vault` . If the admin **does not** stake these SUI tokens with validators, they can be transferred to other parties when users invoke the above functions. However, if the `sui_vault` lacks sufficient SUI, users holding `HASUI` may not be able to retrieve their SUI.

Since this mechanism enables the above operations to acquire SUI as needed, it could significantly impact the project's fund flow and potentially result in financial losses.

Recommendation

We would like to check with the team if the scenario has been considered.

Alleviation

[Haedal, 07/22/2025]: First, if the user does not specify the node to be staked or stake to an inactive node, the funds will be injected into `sui_vault` , and then the funds in it will be staked to the active node before the current epoch is about to end.

In the long run, all the funds received by the protocol are basically in `sui_vault` or staked in the nodes.

When other operations need to withdraw money from the protocol, they will first obtain funds from `sui_vault` . If not enough, they will be withdrawn from the node, so there will be no situation where the user cannot withdraw due to insufficient funds.

[CertiK, 07/22/2025]: The current mode highly requires that the admin/operator operations are successfully executed as expected, and recommends that the team carefully monitor the corresponding operations.

STA-04 | Discussion On Sui Stake Mechanism

Category	Severity	Location	Status
Design Issue	● Informational	sources/staking.move: 561~567, 729~730	● Acknowledged

Description

According to the Sui official documentation, if a user decides to stake with an active validator in Epoch E, the staking process will actually commence in Epoch E+1. If this validator becomes inactive in Epoch E+1, the user will only possess stSUI without receiving any rewards. In this project, the strategy involves first retrieving all active validators in the current Epoch, identifying the user's preferred validator from the active set, and staking to them.

For stakes intended for inactive validators, the contract calculates the average value and distributes these stakes to the active validators.

Furthermore, the calculation below will result in dust due to truncation:

```
let avg_amount = need_stake_amount / validator_count;

        let j = 0;
        while (j < validator_count)  {
            // do stake
            let validator_bal = vault::withdraw(&mut staking.sui_vault, avg_amount);
            let validator = *vector::borrow(&validators, j);
            stake_to_validator(validator_bal, staking, wrapper, validator, ctx);
            j = j + 1;
        };
```

This implies that the total staked amount recorded in the Staking struct may not be fully allocated to the validators in a single `do_stake` operation. Consequently, the user's staking rewards could be affected as some dust remains in the contract and is not staked to the validators.

In the `do_stake` function, when staking SUI from `sui_vault` to validators, the variable `validator_count` might be smaller than the actual length of `validators` . This results in staking tokens primarily to the initial validators in the list. In extreme cases, this could lead to an uneven distribution of SUI among validators, potentially causing side effects for the project and the blockchain.

Recommendation

We would like to check with the team if the scenario has been considered.

Alleviation

[Haedal, 07/22/2025]:

1. The dust is very small, it has very limited impact on the rewards, nearly none.

2. We control the number of nodes and regularly change the order of nodes(calling `sort_validators`) to control the staking of SUI to different nodes.

STA-07 | Potential Missing Validation On Validators

Category	Severity	Location	Status
Volatile Code	● Informational	sources/staking.move: 529	● Acknowledged

Description

As noted in the comment, the `do_stake` function is executed by a privileged role at the end of each epoch. This process involves:

1. Calling `stake_user_selected_validators` .
2. If any user-selected validator is inactive, the balance is transferred to `staking.sui_vault` and subsequently staked to the provided `validators` .

However, there is no validation on the input `validators` , which can lead to potential issues:

- Inactive validators may still be present, causing the invocation to fail.
- Duplicate validators in the input list could result in an uneven distribution of stakes.

Recommendation

Recommend adding corresponding validations to avoid unexpected staking results.

Alleviation

[Haedal, 07/22/2025]:

1. Haedal doesn't store the inactive validators on-chain, but prefers to provide a web UI for users to select the active validators.

2. When calling `do_stake` , Haedal selects validators off-chain, and passes them to `do_stake` . We ensure even distribution among nodes off-chain, or by weight.

STA-08 | Potential Manipulation Of SUI And HaSUI Balances

Category	Severity	Location	Status
Design Issue	● Informational	sources/staking.move: 274~275	● Acknowledged

Description

The `inject_rewards` function allows anyone to transfer SUI tokens to the SUI vault and update `staking.total_rewards`.

However, `staking.total_rewards` is used in the `get_total_sui` function, which, in turn, is referenced by `get_stsui_by_sui` to determine the amount of `haSUI` minted when users deposit SUI. Similarly, `get_total_sui` is also used in `get_sui_by_stsui` to calculate the amount of unstaked SUI that can be withdrawn.

If a malicious attacker invokes the `inject_rewards` function, they can manipulate the `staking.total_rewards` value to artificially increase or decrease the output for `haSUI` or SUI amounts.

If the third party relies on the haSui/Sui ratio in the protocol, this could lead to financial losses for both users and the platform.

Recommendation

We would like to check with the team if this scenario has been considered.

Alleviation

[Haedal, 07/22/2025]: It's not an issue. If a user calls inject_rewards, he will deposit the amount SUI into Haedal protocol, and he can only impact the exchange rate of haSUI:SUI to increase. However, he can't earn more haSUI/SUI from the protocol than he deposits.

[CertiK, 07/22/2025]: As mentioned by the Haedal team, the finding is limited due to the cost of the deposit.

OPTIMIZATIONS | HAEDAL PROTOCOL - AUDIT

ID	Title	Category	Severity	Status
STA-05	Unused Field And Functions	Volatile Code	Optimization	● Resolved

STA-05 | Unused Field And Functions

Category	Severity	Location	Status
Volatile Code	● Optimization	sources/staking.move: 85~86	● Resolved

Description

There are several unused fields in the `Staking` and `StakingConfig` structs:

- In the `Staking` struct:
  - `unstake_epochs`

- In the `StakingConfig` struct:
  - `deposit_fee`
  - `validator_reward_fee`

The following functions are currently unused:

- `vault::withdraw_all`
- `util::pool_token_exchange_rate_at_epoch2`

Additionally, the `_validators` input parameter is unused in the `do_unstake_onchain` function. This function has the same implementation as `unstake_inactive_validators`, as both invoke the same internal function, `unstake_inactive_validators`.

Recommendation

We would like to understand the intended purpose of these fields/functions.

Alleviation

[Haedal, 07/22/2025]: The contract has been upgraded several times; the unused fields/functions are kept for compatibility and have no meaningful usage now.

APPENDIX | HAEDAL PROTOCOL - AUDIT

Finding Categories

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.