Haedal - haSUI

Security Assessment

CertiK Assessed on Oct 13th, 2025

CertiK Assessed on Oct 13th, 2025

Haedal - haSUI

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 09/19/2025
	Final report published on 10/13/2025

Vulnerability Summary

9	4	0	5	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.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1	Minor	1 Resolved	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	2 Resolved, 4 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 - HASUI

CODEBASE | HAEDAL - HASUI

Repository

https://github.com/haedallsd/haedal-protocol/tree/audit

Commit

f74661e887bb8d39f8e69b749662b7476498accb

75daf4a7a6e4fc213d9c7ce11ab2fe599194708e

3c502bd8645ea7b4aeb0e63866722911f33fe8f8

510212e965c9fb2b666d5e643b1c28d3b9871450

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | HAEDAL - HASUI

This audit was conducted for Haedal to evaluate the security and correctness of the smart contracts associated with the Haedal - haSUI project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Manual Review and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | HAEDAL - HASUI

Overview

The **Haedal - haSUI** 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 - haSUI**:

```
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 project 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 - HASUI

9	0	1	0	1	1	6
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

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 9 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
HAH-01	**Centralization Related Risks And Upgradability**	**Centralization**	**Centralization**	**Acknowledged**
HAH-02	Incorrect Update On `bal`	Incorrect Calculation	Medium	Resolved
HAH-06	Reward Injection At Zero StSUI Supply Lets First Staker Capture The Entire Injected SUI	Volatile Code	Minor	Resolved
HAH-03	Potential Revert Due To Integer Underflow In Loop Condition	Volatile Code	Informational	Resolved
HAH-05	Outdated Reward Updates Cause Inaccurate Minting And Fee Miscalculation	Logical Issue	Informational	Acknowledged
HAH-07	Discussion On `Cap`	Access Control	Informational	Resolved
HAH-08	`withdraw_staked_sui()` Reward In `withdraw_bal` Taken Via `request_withdraw_stake_non_entry()` May Be Less Than The Value Calculated	Denial of Service	Informational	Acknowledged
HAH-10	Discussion On `get_current_validator_staked_info_detail_single()`	Design Issue	Informational	Acknowledged
HAH-11	`update_validator_rewards()` Does Not Update `staking.rewards_last_updated_epoch` After All Validators Have Been Updated	Logical Issue	Informational	Acknowledged

HAH-01 | Centralization Related Risks And Upgradability

Category	Severity	Location	Status
Centralization	**Centralization**		**Acknowledged**

Description

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

- initialize()
- set_deposit_fee()
- set_reward_fee()
- set_validator_reward_fee()
- set_service_fee()
- set_withdraw_time_limit()
- set_validator_count()
- sort_validators()
- migrate()
- collect_rewards_fee()
- collect_rewards_fee_v2()
- collect_service_fee()
- toggle_stake()
- toggle_unstake()
- toggle_claim()
- do_stake()
- update_total_rewards_onchain()
- unstake_inactive_validators()
- do_unstake_onchain()
- unstake_pools()
- update_validator_rewards()
- unstake_from_validator()
- init_acl()
- add_minor_signs_to_acl()
- del_minor_signs()
- add_breaker_to_acl()
- del_breaker_to_acl()
- add_robot_to_acl()

- del_robot_to_acl()

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

- toggle_stake()
- toggle_unstake()
- toggle_claim()
- do_stake()
- update_total_rewards_onchain()
- unstake_inactive_validators()
- do_unstake_onchain()
- unstake_pools()
- update_validator_rewards()
- unstake_from_validator()
- sort_validators()

In the module **minorsign**, the role **MinorSign** has authority over the functions:

- set_withdraw_time_limit_v2()
- set_validator_count_v2()
- toggle_stake_v2()

In the module **robot**, the role **Robot** has authority over the functions:

- unstake_inactive_validators_v2()
- update_validator_rewards_v2()
- sort_validators_v2()
- do_stake()
- do_unstake_onchain_v2()

In the module **breaker**, the role **breaker** has authority over the functions:

- toggle_unstake_v2()
- toggle_claim_v2()

If any of these privileged accounts are compromised, an attacker could exploit their enabled authorities to alter protocol parameters, manipulate staking and unstaking processes, upgrade or pause the contract, or change operator lists.

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, 10/13/2025]:

All previous privileges and upgrade authorities have been transferred to the main multisig, which manages other permissions via ACL control. The latest code includes ACL adjustments. **AdminCap** and **OperatorCap** privileges have been transferred to the main multisignature wallet. All other Cap roles have been **deprecated** and are no longer in use.

[CertiK, 10/13/2025]:

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

HAH-02 | Incorrect Update On `bal`

Category	Severity	Location	Status
Incorrect Calculation	Medium	staking.move: 534~535	Resolved

Description

In the `if (need_amount > 0)` branch, the code ultimately uses the initial value of `left_amount` to transfer tokens from `unstaked_bal` to the user:

```
balance::join(&mut bal, balance::split(&mut unstaked_bal, left_amount));
```

When `is_instant == true`, the withdrawal logic first deducts any available `user_selected_validator_bals` to help cover `need_amount`, thereby reducing the additional amount required for withdrawal. However, the variable `left_amount` is not updated to reflect this adjustment. As a result, when computing the amount to withdraw from `unstaked_bal`, the code may use an outdated `left_amount`, which could exceed the actual remaining requirement after validators have contributed.

If the remaining deficit is less than `left_amount`, the call to `split(..., left_amount)` may attempt to withdraw more than is available in `unstaked_bal`, leading to a failed or reverted transaction.

Recommendation

It is recommended recommend to update the logic to compute the precise amount needed from the `unstaked_bal`, ensuring that, whether `is_instant` is true or false, the correct amount is split and joined with the user's balance.

Alleviation

[Haedal, 09/19/2025]: The team heeded the advice and resolved the issue in commit 8a235f640ce38d87b5c9f5edc3e56eb7ce0e821f by adding the following code snippet:

```
if (need_amount > 0) {
    left_amount = need_amount;
...
```

HAH-06 Reward Injection At Zero StSUI Supply Lets First Staker Capture The Entire Injected SUI

Category	Severity	Location	Status
Volatile Code	Minor	staking.move: 1063	Resolved

Description

`inject_rewards` deposits the input SUI into `staking.sui_vault` and increments `staking.total_rewards` by the same amount with no access control or precondition on current supply.

When `stsui_supply == 0`, `get_exchange_rate(staking)` (called in this function) returns `EXCHANGE_RATE_PRECISION` (1:1), and `get_stsui_by_sui(staking, sui_amount)` elsewhere also mints haSUI 1:1 regardless of existing SUI backing.

As a result, if rewards are injected while no stSUI exists, the next (first) minter receives stSUI at 1:1 even though the system already holds injected SUI. That first minter can then redeem that stSUI to withdraw essentially the entire injected SUI from the vault. This makes any reward injection made at zero supply fully siphonable by whoever mints first.

Recommendation

Recommend adding an assert in `inject_rewards` which requires the token supply larger than 0.

Alleviation

[Haedal, 09/22/2025]: The team heeded the advice and resolved the issue in commit 75daf4a7a6e4fc213d9c7ce11ab2fe599194708e.

HAH-03 | Potential Revert Due To Integer Underflow In Loop Condition

Category	Severity	Location	Status
Volatile Code	Informational	util.move: 37	Resolved

Description

In the `pool_token_exchange_rate_at_epoch()` function, the loop condition `while (epoch >= 0)` allows execution when `epoch` is 0. Within the loop, `epoch` is decremented:

```
        while (epoch >= 0) {
            ...
            epoch = epoch - 1;
        };
```

When `epoch` reaches 0, subtracting one causes an integer underflow, resulting in an abort. This may disrupt normal execution flow or lead to unexpected behavior if not properly handled.

Recommendation

It is recommended to use `while (epoch > 0)` instead of `while (epoch >= 0)`.

Alleviation

[Haedal, 09/19/2025]: The team heeded the advice and resolved the issue in commit 8a235f640ce38d87b5c9f5edc3e56eb7ce0e821f by using `while (epoch > 0)` instead of `while (epoch >= 0)`.

HAH-05 Outdated Reward Updates Cause Inaccurate Minting And Fee Miscalculation

Category	Severity	Location	Status
Logical Issue	Informational	staking.move: 597~598, 634~635	Acknowledged

Description

The functions `update_total_rewards_onchain()` and `update_validator_rewards()` are designed for the admin to update `staking.total_rewards` and `staking.uncollected_protocol_fees` at the beginning of the epoch, which are essential for determining the exchange rate and the amount of tokens to mint, as well as for protocol fee collection. After these functions are called, the field `staking.rewards_last_updated_epoch` is set to the current epoch to signal that the most recent epoch's rewards and fees have been accounted for:

```
staking.rewards_last_updated_epoch = current_epoch;
```

However, user-facing functions such as `request_stake_coin()`, `request_unstake_instant_coin()`, and `request_unstake_delay()` do not verify whether `staking.rewards_last_updated_epoch` matches the current epoch before proceeding. If a user executes these operations before the rewards for the current epoch are updated, the protocol relies on stale reward data, which introduces several issues:

- The `request_stake_coin()` function mints `haSUI` tokens based on an outdated `staking.total_rewards` value. If new rewards have not yet been added, this can result in over-minting.
- `staking.uncollected_protocol_fees` is updated only by the aforementioned admin-only functions. If users unstake or claim before these updates, their share of rewards may not be properly reflected in `staking.total_rewards`, and protocol fees may be miscalculated, leading to potential loss of fees for the project.
- 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.

Currently, only the explicit reward update functions update `staking.total_rewards` via the following chain: `update_validator_rewards()` / `update_total_rewards_onchain()` ->

`calculate_validator_pool_rewards_increase()` -> `calculate_staked_sui_rewards()`

In contrast, `do_validator_unstake()` also calls `calculate_staked_sui_rewards()`, but its computed rewards do not update `staking.total_rewards`.

This design allows timing gaps in which users can interact with outdated protocol state.

Recommendation

To mitigate these risks, pause all user operations (claim, stake, unstake) between the start of a new epoch and the completion of reward updates, as indicated by the in-code 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
```

Alternatively, it is recommended to verify that `staking.rewards_last_updated_epoch` matches the current epoch before calling `request_stake_coin()`, `request_unstake_instant_coin()`, or `request_unstake_delay()`, to ensure that reward calculations and fee collections are always based on up-to-date data.

Alleviation

[Haedal, 09/19/2025]: We will do this (pause user operations during epoch change).

HAH-07 | Discussion On `Cap`

Category	Severity	Location	Status
Access Control	Informational	sources/manage.move (haedal): 25~40	Resolved

Description

`MinorSignCap`, `BreakerCap`, `RobotCap`, and `OperatorCap` have the `key` and `store` abilities, allowing them to be directly `public_transfer`ed by the holding account to other accounts. This means the granting and revocation of these privileged roles are not determined by the `AdminCap` holder but instead by the respective holders themselves. This would result in these roles not being managed or controlled by the `AdminCap` holder. We would like to ask whether this is the intended design.

```
25      struct MinorSignCap has store, key {
26          id: UID,
27      }
28
29      struct BreakerCap has store, key {
30          id: UID,
31      }
32
33      struct RobotCap has store, key {
34          id: UID,
35      }
36
37      /// `OperatorCap` is used by the offchain programs.
38      struct OperatorCap has store, key {
39          id: UID,
40      }
```

Recommendation

We would like to ask whether this is the intended design.

Alleviation

[Haedal, 09/24/2025]: The team heeded the advice and resolved the issue by using ACL in commit 08f9ea76aeacd32f4c44489e3415be7774a8f01e.

HAH-08 `withdraw_staked_sui()` Reward In `withdraw_bal` Taken Via `request_withdraw_stake_non_entry()` May Be Less Than The Value Calculated

Category	Severity	Location	Status
Denial of Service	Informational	sources/staking.move (haedal): 997~1007	Acknowledged

Description

In `withdraw_staked_sui()`, the principal and rewards are calculated following the approach used in Sui's official `staking_pool.move`, and then the official function `sui_system::request_withdraw_stake_non_entry()` is called to withdraw the corresponding funds.

``staking.move`

```
  997      fun withdraw_staked_sui(wrapper: &mut SuiSystemState, staked_sui:
StakedSui, unstaked_bal: &mut Balance<SUI>, ctx: &mut TxContext):(u64, u64) {
  998          // calculate total value of the staked_sui
  999          let principal = staking_pool::staked_sui_amount(&staked_sui);
 1000  @>      let rewards = calculate_staked_sui_rewards(wrapper, &staked_sui,
tx_context::epoch(ctx));
 1001  @>      let withdraw_bal = sui_system::request_withdraw_stake_non_entry(
wrapper, staked_sui, ctx);
 1002          let withdraw_amount = balance::value(&withdraw_bal);
 1003  @>      assert!(withdraw_amount == principal + rewards,
EStakedSuiRewardsNotMatched);
 1004
 1005          balance::join(unstaked_bal, withdraw_bal);
 1006          (principal, rewards)
 1007      }
```

`util.move`

```
 11      public fun calculate_rewards(wrapper: &mut SuiSystemState, pool_id: ID,
 staked_amount: u64, stake_activation_epoch: u64, current_epoch: u64):u64 {
 12          if (stake_activation_epoch >= current_epoch) {
// no rewards yet, referred sui_system::request_withdraw_stake()
 13              return 0
 14          };
 15
 16          let exchange_rates = sui_system::pool_exchange_rates(wrapper, &pool_id)
;
 17
 18          let pool_token_withdraw_amount = {
 19              let exchange_rate_at_staking_epoch =
pool_token_exchange_rate_at_epoch(exchange_rates, stake_activation_epoch);
 20              get_token_amount(&exchange_rate_at_staking_epoch, staked_amount)
 21          };
 22
 23          let new_epoch_exchange_rate = pool_token_exchange_rate_at_epoch(
exchange_rates, current_epoch);
 24          let total_sui_withdraw_amount = get_sui_amount(&new_epoch_exchange_rate
, pool_token_withdraw_amount);
 25
 26          let reward_withdraw_amount =
 27              if (total_sui_withdraw_amount > staked_amount)
 28                  total_sui_withdraw_amount - staked_amount
 29              else 0;
 30
 31          reward_withdraw_amount
 32      }
```

However, there is one difference: in the official `withdraw_rewards()` function, if the `pool.rewards_pool` is insufficient to pay out the computed reward amount, it takes the smaller value of `pool.rewards_pool.value()` as the reward and returns it.

`staking_pool.move`

```
419  fun withdraw_rewards(
420      pool: &mut StakingPool,
421      principal_withdraw_amount: u64,
422      pool_token_withdraw_amount: u64,
423      epoch: u64,
424  ): Balance<SUI> {
425      let exchange_rate = pool.pool_token_exchange_rate_at_epoch(epoch);
426      let total_sui_withdraw_amount = exchange_rate.get_sui_amount(
pool_token_withdraw_amount);
427      let mut reward_withdraw_amount = if (total_sui_withdraw_amount >=
 principal_withdraw_amount) {
428          total_sui_withdraw_amount - principal_withdraw_amount
429      } else 0;
430
431      // This may happen when we are withdrawing everything from the pool and
432      // the rewards pool balance may be less than reward_withdraw_amount.
433      // TODO: FIGURE OUT EXACTLY WHY THIS CAN HAPPEN.
434      reward_withdraw_amount = reward_withdraw_amount.min(pool.rewards_pool.value
());
435      pool.rewards_pool.split(reward_withdraw_amount)
436  }
```

If such an unexpected situation occurs, the actual rewards withdrawn will be less than the value calculated by `withdraw_staked_sui()`, causing the transaction to abort due to a check statement `assert!(withdraw_amount == principal + rewards, EStakedSuiRewardsNotMatched);`.

Recommendation

We would like to confirm with the Sui team whether this is an intentional design.

Alleviation

[Haedal, 09/30/2025]:

This is intentional.

The official `withdraw_rewards` function returns both the principal and the rewards. However, in extreme cases (currently unknown), the rewards may be insufficient, causing the official protocol to return a smaller value. In this case, we perform strong amount validation and reject the transaction. This is because the reward calculation for hasui-token and the entire system is derived from the calculate_staked_sui_rewards function. Therefore, accepting this extreme case would result in a loss of rewards and an error in the direct ratio between hasui and sui.

In reality, due to the official strict control of rewards, rewards are only generated as epochs progress, so theoretically, insufficient rewards are unlikely.

HAH-10 Discussion On `get_current_validator_staked_info_detail_single()`

Category	Severity	Location	Status
Design Issue	Informational	staking.move: 1287	Acknowledged

Description

Below are some issues and optimizations for the `get_current_validator_staked_info_detail_single()`:

1. `get_current_validator_staked_info_detail_single()` first assigns `ret.pool_id` to the staking's id, and later reassigns it to the `pool_id` of the actual staking pool where the stake was deposited. We would like to clarify whether this field is intended to represent the staking id or the staking pool's `pool_id` for the validator. If it is the staking pool's `pool_id`, it only needs to be assigned once and does not need to be reassigned repeatedly inside the loop.

`staking.move`

```
 1258      public fun get_current_validator_staked_info_detail_single(staking: &
Staking, wrapper: &mut SuiSystemState, validator: address, recalc: bool, ctx: &
TxContext): ValidatorStakedInfoV2 {
 1259          let pool = table::borrow(&staking.pools, validator);
 1260          let ret = ValidatorStakedInfoV2{ validator, total_staked: pool.
total_staked, rewards: pool.rewards, staked_sui_count: table_queue::length(&pool.
staked_suis),
 1261  @>          pool_id: object::id(staking),
 1262              exchange_rates_unexisted_epoches: vector<u64>[],
 1263              stake_activation_epoches: vector<u64>[],
 1264          };
 1265          if (ret.staked_sui_count == 0) {
 1266              return ret
 1267          };
 1268
 1269          if (recalc) {
 1270              ret.total_staked = 0;
 1271              ret.rewards = 0;
 1272          };
 1273
 1274          let epoch_map = vec_map::empty<u64, u64>();
 1275
 1276          let current_epoch = tx_context::epoch(ctx);
 1277          let tail = table_queue::tail(&pool.staked_suis);
 1278          let i = table_queue::head(&pool.staked_suis);
 1279          while (i < tail) {
 1280              let staked_sui_ref = table_queue::borrow(&pool.staked_suis, i);
 1281              if (recalc) {
 1282                  ret.total_staked = ret.total_staked + staking_pool::
staked_sui_amount(staked_sui_ref);
 1283                  ret.rewards = ret.rewards + calculate_staked_sui_rewards(
wrapper, staked_sui_ref, current_epoch);
 1284              };
 1285
 1286              let stake_activation_epoch = staking_pool::stake_activation_epoch
(staked_sui_ref);
 1287  @>          ret.pool_id = staking_pool::pool_id(staked_sui_ref);
```

2. `get_current_validator_staked_info_detail_single()` may insert duplicate `stake_activation_epoch` values into `ret.stake_activation_epoches`, whereas `ret.exchange_rates_unexisted_epoches` only contains distinct epochs. We want to confirm whether this is the intended behavior; if not, `ret.stake_activation_epoches` should insert each epoch only once, similar to `ret.exchange_rates_unexisted_epoches`.

`staking.move`

```
 1295             if (!table::contains(exchange_rates, stake_activation_epoch)) {
 1296                 if (!vec_map::contains(&epoch_map, &stake_activation_epoch))
{
 1297                     vector::push_back(&mut ret.
exchange_rates_unexisted_epoches, stake_activation_epoch);
 1298                     vec_map::insert(&mut epoch_map, stake_activation_epoch, 0
);
 1299                 };
 1300             };
 1301             vector::push_back(&mut ret.stake_activation_epoches,
 stake_activation_epoch);
```

Recommendation

We want to confirm whether this is the intended design.

Alleviation

[Haedal, 09/30/2025]: This is a function used in our protocol to perform some data queries temporarily. We will consider canceling this function in the future.

HAH-11 `update_validator_rewards()` Does Not Update `staking.rewards_last_updated_epoch` After All Validators Have Been Updated

Category	Severity	Location	Status
Logical Issue	Informational	staking.move: 638	Acknowledged

Description

When `update_total_rewards_onchain()` can abort due to high gas usage, `update_validator_rewards()` is used to update each validator's rewards separately to avoid that. However, after using `update_validator_rewards()` to update all validators' rewards, it does **not** update `staking.rewards_last_updated_epoch` the way `update_total_rewards_onchain()` does. This causes `staking.rewards_last_updated_epoch` to become inconsistent with the actual update state.

Recommendation

We recommend implementing a mechanism to track whether each validator has been updated to the current epoch, and to update `staking.rewards_last_updated_epoch` once all validators have been brought up to date for the current epoch.

Alleviation

[Haedal, 09/30/2025]: The latest update_rewards function is updated on a single node by the control center (robot) by iterating through all validators and calling the update_validator_rewards function. This is because we previously aborted the update_total_rewards_onchain function due to excessive gas usage caused by too many validators.

Similarly, we also store updates to rewards_last_updated_epoch in the control center's transaction.

Detailed steps for the off-chain control center: Create a transaction > Check and synchronize the official epoch > Check validators > Pre-transaction > Initiate transactions individually > Check each transaction individually > Update the epoch > Submit the entire transaction.

Therefore, in the latest protocol processing, we will not update rewards_last_updated_epoch in the protocol, but instead handle it off-chain.

[CertiK, 09/30/2025]: It is recommended to update `staking.rewards_last_updated_epoch` within the function once the last validator reward has been updated.

APPENDIX | HAEDAL - HASUI

Audit Scope

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

haedallsd/haedal-protocol
sources/robot.move
sources/staking.move
sources/table_queue.move
sources/util.move
sources/vault.move

Finding Categories

Categories	Description
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
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.