Haedal

Audit Report

MOVEBIT

contact@movebit.xyz

https://twitter.com/movebit_

Tue Aug 22 2023

Haedal Audit Report

1 Executive Summary

1.1 Project Information

Description	Haedal is a liquid staking protocol built on Sui that allows anyone to stake their SUI tokens.
Type	Staking
Auditors	MoveBit
Timeline	Wed Aug 02 2023 – Tue Aug 22 2023
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/haedallsd/haedal-protocol
Commits	a08c58e41c7e8031b6a5f057d3bd7d5692e86eec 2d6887866cd7ca7e514b8830dd901c83bad649a6 4b065d0b5011ee43b1d215f22232e5605196de18 48eac9a37c8d7157570cbb798b9df7a9cf53676e 090b1e4a3b2094ade55ab18ec29a2438b7b1e63e

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	Move.toml	c4f05a6e66c296ab9502b2a6831e857b6a23b8bb
INT	sources/interface.move	a7ced125a396a3da112510992a27c88059aa05e4
UTI	sources/util.move	e53382b5a2239fe89c04c3b9876bd6e6474d3e23
CON	sources/config.move	b952d17bdd91864bfdf8da06c620c776b286b756
HAS	sources/hasui.move	043c51935bddf988961f1a5790f253dda55b90f6
VAU	sources/vault.move	cbe4a94055499ed482fe96b5a236f8fb6c82474d
TQU	sources/table_queue.move	02257831926d5787f5a90d68061ec8747c0136fb
OPE	sources/operate.move	b45d0c30557189c1747cdb57e931808ae303d473
MAN	sources/manage.move	b7de3dc17f36bf9c06e7e2980866915d579f11cb
STA	sources/staking.move	e05ac7461bb217467ea5ab87cffd82fb5d84459e

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	8	0
Informational	0	0	0
Minor	5	5	0
Medium	1	1	0
Major	2	2	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Haedal to identify any potential issues and vulnerabilities in the source code of the Haedal smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
MAN–1	Unnecessary Bool Judgment	Minor	Fixed
STA–1	`Pool.rewards` Override	Major	Fixed
STA–2	Precision Issue	Minor	Fixed
STA–3	Unused Constants	Minor	Fixed
STA–4	The Error Codes Are Not Used Anywhere	Minor	Fixed
STA–5	`staking.pending_unstake_amount` Not Updated	Medium	Fixed
STA–6	Miscomputation Issue	Major	Fixed
UTI–1	`calculate_rewards` Can Be Optimized During Boundary Handling	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Haedal Smart Contract:

User

- Users can invoke the `request_stake` function to apply for staking.
- Users can invoke the `request_unstake_instant` function to request an immediate unstaking operation.
- Users can invoke the `request_unstake_delay` function to request a delayed unstaking operation.
- Users can invoke the `claim` function to collect rewards from unstaking.
- Users can invoke the `import_stake_sui_vec` function to import staking information from another system.

Admin

- Admin can initialize contract configurations by calling the `initialize` function.
- Admin can set various configuration parameters related to the staking mechanism, such as deposit fees, reward fees, validator reward fees, service fees, withdrawal time limits, and validator counts.
- Admin can migrate data to a new version of the staking contract, invoked by the new package, using the `migrate` function.
- Admin can collect rewards fees from the staking contract for specific accounts by calling the `collect_rewards_fee` function.
- Admin can collect service fees from the staking contract for specific accounts by calling the `collect_service_fee` function.
- Admin can pause stake and unstake actions by calling the `toggle_stake` and `toggle_unstake` functions.
- Admin can use the `do_stake` function to stake users' SUI tokens to validator nodes for staking rewards.
- Admin can update the haSUI to SUI exchange rate by calling the `update_total_rewards_onchain` function at the beginning of each epoch.
- Admin can initiate unstaking by calling the `do_unstake_onchain` function at the beginning of each epoch, approving the retrieval of SUI tokens by unstaking StakedSui

tokens.

- Admin can initiate unstaking of all StakedSui tokens from risky validators' pools by calling the `unstake_pools` function.

Operator

- Operator can use the `do_stake` function to stake users' SUI tokens to validator nodes for staking rewards.
- Operator can update the haSUI to SUI exchange rate by calling the `update_total_rewards_onchain` function at the beginning of each epoch.
- Operator can initiate unstaking by calling the `do_unstake_onchain` function at the beginning of each epoch, approving the retrieval of SUI tokens by unstaking StakedSui tokens.
- Operator can initiate unstaking of all StakedSui tokens from risky validators' pools by calling the `unstake_pools` function.

4 Findings

MAN-1 Unnecessary Bool Judgment

Severity: Minor

Status: Fixed

Code Location:

sources/manage.move#L41

Descriptions:

The condition `cap.init == false` in the assertion would result in unnecessary gas consumption. It can be replaced with `!cap.init` for better gas efficiency. You can see our test data below:

```
public fun test2() {
        let a = is_even_number(7);
        let i = 0;
        while (i < 1000) {
        assert!(!a, 0);
        i=i+1;
        }
    }

public fun test4() {
        let a = is_even_number(7);
        let i = 0;
        while (i < 1000) {
            assert!(a == false, 0);
            i=i+1;
        }
    }
```

The test2 and test4 cost gas 76 and 111 respectively.So according to the test data, `!a` is indeed more gas efficient than `a == false`.

Suggestion:

It is recommended to replace `cap.init == false` with `!cap.init` for better gas efficiency.

Resolution:

This issue has been fixed. The client modified it to `!cap.init`.

STA-1 `Pool.rewards` Override

Severity: Major

Status: Fixed

Code Location:

sources/staking.move#L337

Descriptions:

There is an issue with importing staking information in the `import_ss` function. Within this function, the rewards are calculated using the `calculate_staked_sui_rewards` function and then directly assigned to the `pool.rewards` variable. This approach is clearly incorrect, as it overwrites the previous value of `pool.rewards`. The correct approach should involve adding the calculated rewards to the previous value.

Incorrectly overwriting the value will lead to severe computational problems. For instance, in the `calculate_validator_pool_rewards_increase` function, the aforementioned issue will cause the calculated reward increase to be excessively high. This, in turn, could result in significant protocol fund losses.

Suggestion:

It's recommended to add the calculated rewards to the existing value of `pool.rewards` instead of directly overwriting it.

Resolution:

This issue has been fixed. The client changed from direct overwrite to addition.

STA-2 Precision Issue

Severity: Minor

Status: Fixed

Code Location:

sources/staking.move#L1037-1044

Descriptions:

While we understand that 10e6 represents the precision for frontend display, there are two scenarios in the `get_exchange_rate` function. In one case, it returns 1, while in the other case, it returns a value of the magnitude of 10e6. The return values of these two scenarios are not within the same order of magnitude, which might not be suitable.

Suggestion:

It is recommended to return values within the same magnitude.

Resolution:

This issue has been fixed. The client has already addressed this accuracy issue.

STA-3 Unused Constants

Severity: Minor

Status: Fixed

Code Location:

sources/staking.move#L8-9

Descriptions:

The two constants, `UNSTAKE_TYPE_NORMAL` and `UNSTAKE_TYPE_INSTANT`, are defined as unsigned 8-bit integers within the contract. However, despite being defined in the contract, it appears that these constants are not used in any functions, processes, or logic within the contract.

Suggestion:

It is recommended to remove these two unnecessary constants.

Resolution:

This issue has been fixed. The client deleted these two constants.

STA-4 The Error Codes Are Not Used Anywhere

Severity: Minor

Status: Fixed

Code Location:

sources/staking.move#L42-55

Descriptions:

The Error codes are not used anywhere, and with further investigation, we found that `TreasuryCap` is not used either, which was meant to handle the mint and burn functions of StSui tokens.

```
const EClaimEpochNotFound: u64 = 2; // @audit not used
const ENoMinStakingThreshhold: u64 = 15; //@audit not used
```

In the current implementation of `get_epoch_claim` function, it is designed that if `epoch` cannot be found, then a new `EpochClaim` will be created and return it. So it's not it seems that the `EClaimEpochNotFound` is safe to be deleted.

For the `ENoMinStakingThreshhold`, since there are assertions to check to make sure if `MIN_STAKING_THRESHOLD` is met, there is no need to have `ENoMinStakingThreshhold`.

Suggestion:

It is recommended to remove those error codes unless defined otherwise.

Resolution:

This issue has been fixed. The client removed the two constant values.

STA-5 `staking.pending_unstake_amount` Not Updated

Severity: Medium

Status: Fixed

Code Location:

sources/staking.move#L331-357

Descriptions:

The `staking.pending_unstake_amount` variable represents the total amount of tokens that are currently scheduled to be unstaked. However, based on the current implementation of the `claim()` function, this amount is not reduced when a user successfully claims their unstaked tokens. This can lead to a discrepancy between the actual tokens that are available for unstaking and the value of `staking.pending_unstake_amount`. In effect, the `staking.pending_unstake_amount` could potentially be overstating the actual amount of tokens that are scheduled to be unstaked.

This could mislead users about the current state of the staking pool and may lead to errors in other parts of the code that rely on `staking.pending_unstake_amount` for calculations or decision-making.

Suggestion:

It's recommended to update the `staking.pending_unstake_amount` variable in the `claim()` function to ensure its accuracy, by subtracting the amount of tokens that have been claimed by the user.

Resolution:

This issue has been fixed. The client followed our suggested fixes.

STA-6 Miscomputation Issue

Severity: Major

Status: Fixed

Code Location:

sources/staking.move#L253

Descriptions:

The function `staking.request_unstake_delay()` allows a user to unstake their staked tokens (stSUI). There is a mistake in the computation `staking.total_staked` + `staking.total_rewards` - `staking.total_staked` present in the code below. Specifically, `staking.total_staked` is added and immediately subtracted, which effectively cancels it out of the equation, leaving just staking.total_rewards. This computation does not to achieve any meaningful calculation or objective. Due to this error in the computation, the variable `max_exchange_sui_amount` can only be less than or equal to staking.total_rewards. This issue poses a significant problem as it restricts users from requesting to unstake (`request_unstake_delay()`) if their staking rewards plus the original staked amount (minus any previous unstakes) exceed the total_rewards. This discrepancy could limit the ability of users to fully interact with the staking protocol, potentially locking their tokens in the system under certain circumstances. The function `request_unstake_instant()` also has the same issue.

Suggestion:

It is recommended that the correct computation should be `staking.total_staked` + `staking.total_rewards` - `staking.total_unstaked`

Resolution:

This issue has been fixed. The client followed our suggested fixes.

```
assert!(max_exchange_sui_amount <= get_total_sui(staking) &&
max_exchange_sui_amount > 0, EUnstakeExceedMaxSuiAmount);
```

UTI-1 `calculate_rewards` Can Be Optimized During Boundary Handling

Severity: Minor

Status: Fixed

Code Location:

sources/util.move#L11-31

Descriptions:

We found that two `if` statements can be optimized to skip some conditions and reduce the heavy gas fees when it's called by both `update_pool_rewards` and `do_pool_unstake`. It's used to calculate the staking reward by retrieve the exchange rate at staked epoch and current epoch, and calculate if the withdrawable SUI is more than staked ones. If so, then it will return `reward_withdraw_amount`, 0 otherwise.

1. If `stake_activation_epoch = current_epoch` then it should also return 0 without going deep below. Let's delve in. Assume that `stake_activation_epoch = current_epoch`, then `exchange_rate_at_staking_epoch = new_epoch_exchange_rate` because they use the same function with same parameters values.Then we convert `staked sui amount` into `token_amount`, then `token_amount` to `total_sui_withdraw_amount`. But since the two exchange rate are the same, it means `staked_sui_amount = total_sui_withdraw_amount`. Then `reward_withdraw_amount` should just be 0. in a high level this makes sense since rewards will only be available after at least 1 epoch, if you stake and unstake at the same epoch, there shouldn't be any reward yet.
2. Obviously `if (total_sui_withdraw_amount = staked_amount)` then you can directly return 0 without further subtraction.

Suggestion:

It is recommended to change the codes below.

`if (stake_activation_epoch >= current_epoch)`

`if (total_sui_withdraw_amount > staked_amount)`

Resolution:

This issue has been fixed. The client made modifications to optimize the process using the recommended method.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.