Hawal

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Mar 14 2025

Hawal 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 to contribute to governance and decentralisation of the Sui blockchain
Type	Staking
Auditors	MoveBit
Timeline	Mon Mar 03 2025 - Fri Mar 14 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/haedallsd/hawal-contract
Commits	c74a124c38756ef217dd676febb3a8ad30155623 d3d51621591c1a2bd7a5553f126361653d861b86 b34346e9b2dc99cf4cf3dc8bac8c516ffaf0de35

1.2 Files in Scope

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

ID	File	SHA-1 Hash
INT	sources/interface.move	b9781ff39233065869bef2b7e3cf6d487776672a
HAW	sources/hawal.move	2655277d45f22eb65a6304f12ac9bc3613814a20
CON	sources/config.move	8c6d51ca7b2146ded6b224e630aee41fe18c8ed9
VAU	sources/vault.move	87748894f1494a8a576b29cf428a607e8f9d4ce9
MAN	sources/manage.move	c15ad07cd53162feced4127601cee5c96fb577fc
MOV4	Move.toml	ef47dd699c6ef4ab0579aad7d4f1f7e529557af8
WAL1	sources/walstaking.move	4ad79eb100134b6fd67ab590d402e3f8a3a423e7
OPE	sources/operate.move	95a757c5e383ac25d4675efe6a3d971548d9bcb2

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	11	1
Informational	5	4	1
Minor	2	2	0
Medium	1	1	0
Major	4	4	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(Optional)

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 Protocol to identify any potential issues and vulnerabilities in the source code of the Hawal 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 12 issues of varying severity, listed below.

ID	Title	Severity	Status
WAL-1	Limiting Epoch Incorrect Judgment	Major	Fixed
WAL-2	Incorrect Implementation of `get_min_total_validator`	Major	Fixed
WAL-3	Incorrect Stake Reward Calculation	Major	Fixed
WAL-4	The User can Withdraw more wal amount than Expected	Major	Fixed
WAL-5	Claim Epoch Conflict	Medium	Fixed
WAL-6	Ambiguous Error Handling in Check for Staking	Minor	Fixed
WAL-7	Unused Parameters and Redundant Mutability	Minor	Fixed
WAL-8	Unused `user_selected_validator_bals` Field	Informational	Fixed
WAL-9	Non-Standard Comments	Informational	Fixed
WAL-10	Hardcoded Values in Initialization	Informational	Fixed

WAL-11	Unused `unstaked_amount` Calculation	Informational	Fixed
WAL-12	Dependence on Centralized Storage Service for image URL	Informational	Acknowledged

3 Participant Process

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

- `set_xxxx_fee` : Configure fee rates for specific operations.
- `set_withdraw_time_limit` : Set the time limit for withdrawals.
- `set_validator_count` : Adjust the number of network validators.
- `sort_validators` : Reorder validator list.
- `set_active_validators` : Activatespecific validators.
- `set_walrus_epoch_start` : Define the starting epoch.
- `migrate` : Migrate the data version.
- `collect_rewards_fee` : Collect rewards fee.
- `toggle_xxxx` : Adjust operations' status.
- `update_validator_rewards` : Update reward of each node.

User

- `request_stake` : Stake WAL tokens to mint HAWAL.
- `request_unstake_delay` : Burn HAWAL to queue WAL unlock, generating an UnstakeTicket for future claiming.
- `claim` : Redeem an approved UnstakeTicket to transfer unlocked WAL tokens to the caller's address.

4 Findings

WAL-1 Limiting Epoch Incorrect Judgment

Severity: Major

Status: Fixed

Code Location:

sources/walstaking.move#378-379

Descriptions:

In the `do_validator_request_withdraw` function, the following judgment exists:

```
let key = keys[i];
// Limiting epoch
if( current_epoch > key ) continue;
```

Here, `current_epoch` represents `claim_epoch`, and `key` represents the `Activation Epoch` of `StakedWal`. Only `Activation Epoch` less than `claim_epoch` should be allowed for the `withdraw` operation, but the current condition is obviously reversed.

Suggestion:

Adjust the `Limiting epoch` condition judgment accordingly.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-2 Incorrect Implementation of `get_min_total_validator`

Severity: Major

Status: Fixed

Code Location:

sources/walstaking.move#677-698

Descriptions:

In the `get_min_total_validator` function, `min_total` is initialized to `0`. If `pools` contain all `active_validators`, the following condition will be entered:

```
if (table::contains(&staking.pools, validator)) {
    let pool = table::borrow(&staking.pools, validator);
    if (pool.total_staked < min_total) {
        min_id = validator;
        min_total = pool.total_staked;
    };
};
```

However, since `total_staked` is always greater than or equal to `0`, the condition `pool.total_staked < min_total` will never be satisfied. This makes it impossible to obtain the validator with the smallest staked amount.

Suggestion:

Set `min_total` to the `total_staked` of the first validator to ensure the comparison works correctly.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-3 Incorrect Stake Reward Calculation

Severity: Major

Status: Fixed

Code Location:

sources/walstaking.move#513;

sources/walstaking.move#652

Descriptions:

In the `calculate_validator_pool_rewards_increase` function, all `StakedWal` rewards are calculated, including the rewards for `withdrawing`, as shown below:

```
let keys = pool.withdrawing.keys();
let mut i = 0;
let length = keys.length();
while (i < length) {
    let key = keys[i];
    let staked_wal_ref = pool.withdrawing.get(&key);
    pool_rewards = pool_rewards + calculate_staked_wal_rewards(wal_staking,
staked_wal_ref, current_epoch);
    i = i + 1;
};
```

Here, the rewards are calculated based on the exchange rate of `current_epoch`, but since these `StakedWal` are already in the `Withdrawing` state, the rewards should be calculated using the `withdraw_epoch`. Otherwise, the calculated rewards will be higher than expected.

The same issue exists in the `do_validator_withdraw` function:

```
let withdraw_principal = withdraw.value();
let withdraw_rewards = calculate_staked_wal_rewards(wal_staking, &withdraw,
current_epoch);
```

Suggestion:

Modify the epoch parameter used for reward calculation to `withdraw_epoch`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-4 The User can Withdraw more wal amount than Expected

Severity: Major

Status: Fixed

Code Location:

sources/walstaking.move#347-409

Descriptions:

In the `request_withdraw_stake()` function, the protocol performs the following actions:

1. Creates a ticket to request more WAL.
2. Burns the user's haWAL tokens.

```
// create a ticket to request more wal, the sender need to claim after approved
    let ut = UnstakeTicket {
        id: object::new(ctx),
        unstake_timestamp_ms: now_ms,
        hawal_amount: input_hawal_amount,
        wal_amount: max_exchange_wal_amount,
        claim_epoch: claim_epoch,
        claim_timestamp_ms: claim_timestamp_ms,
    };
    let sender = tx_context::sender(ctx);
    transfer::transfer(ut, sender);

    // burn all the haWAL
    coin::burn(&mut staking.hawal_treasury_cap, input);
```

However, the protocol **does not update** the following two variables:

- `staking.total_unstaked`
- `staking.hawal_supply`

Later, when the request is approved, the sender calls the `withdraw_stake()` function to claim and return WAL. At this point, the protocol updates both `staking.total_unstaked` and `staking.hawal_supply`.

```
    // update the counters
    staking.total_unstaked = staking.total_unstaked + wal_amount;
    staking.hawal_supply = coin::total_supply(&staking.hawal_treasury_cap);
```

The Issue:

- The `total_unstaked` variable is only increased by the current user's requested amount.
- The `hawal_supply` variable, however, is updated to the **latest value**.

This creates a **mismatch** between `total_unstaked` and `hawal_supply`. As a result, users requesting withdrawals could receive more WAL than they should during the exchange, leading to an incorrect or unfair distribution of funds.

```
public fun get_wal_by_hawal(staking: &Staking, hawal_amount: u64): u64 {
    let total_wal_amount_snapshot = get_total_wal(staking);
    if (total_wal_amount_snapshot == 0 || staking.hawal_supply == 0) {
        return hawal_amount
    };

    let res = (total_wal_amount_snapshot as u128)
            * (hawal_amount as u128)
            / (staking.hawal_supply as u128);
    (res as u64)
}
```

Suggestion:

It is recommended to keep `total_unstaked` and `hawal_supply` updated to the latest value.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-5 Claim Epoch Conflict

Severity: Medium

Status: Fixed

Code Location:

sources/walstaking.move#313-320

Descriptions:

In the `request_withdraw_stake` function, `claim_epoch` is set to `walrus_current_epoch + 1`, which means the claim can be made after the next epoch. However, `claim_timestamp_ms` is set to `current_epoch_end + walrus_epoch_duration`, which means the claim can only be made after the next epoch ends. This may cause a conflict.

```
// n+1
let mut claim_epoch = walrus_current_epoch + 1;
let mut claim_timestamp_ms = current_epoch_end + walrus_epoch_duration;
if (now_ms > mid_epoch_time) {
    // n+2
    claim_epoch = claim_epoch + 1;
    claim_timestamp_ms = claim_timestamp_ms + walrus_epoch_duration
};
```

Suggestion:

It is recommended to adjust to the correct claim epoch and time.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-6 Ambiguous Error Handling in Check for Staking

Severity: Minor

Status: Fixed

Code Location:

sources/walstaking.move#299

Descriptions:

The combined `assert!` check for `max_exchange_wal_amount` bounds uses a single error code (`EUnstakeExceedMaxWalAmount`), obscuring whether the failure is due to exceeding the upper limit or falling below the minimum threshold.

```
assert!(max_exchange_wal_amount <= get_total_wal(staking) &&
max_exchange_wal_amount >= MIN_STAKING_THRESHOLD,
EUnstakeExceedMaxWalAmount);
```

Suggestion:

Split the assertion into two separate checks.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-7 Unused Parameters and Redundant Mutability

Severity: Minor

Status: Fixed

Code Location:

sources/walstaking.move#720;

sources/walstaking.move#578

Descriptions:

Parameters declared in function signatures but not utilized in logic. Variables marked as `mut` without subsequent modification weaken Move's safety guarantee. These increases maintenance complexity and can lead to unexpected behavior.

```
let mut withdraw = if ( // ... )
```

```
fun calculate_validator_pool_rewards_increase(
    system: &System,
    // ...
)
```

Suggestion:

Remove unused parameters and `mut` declarations to to improve code safety and reduce cognitive overhead.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-8 Unused `user_selected_validator_bals` Field

Severity: Informational

Status: Fixed

Code Location:

sources/walstaking.move#82

Descriptions:

The `Staking` object contains the `user_selected_validator_bals` field:

```
public struct Staking has key {
    id: UID,
    /// used to control the package upgrade
    version: u64,
    /// configuration for the protocol
    config: StakingConfig,
    /// keep user's staked wal in current epoch, will be staked in validators at the end of
current epoch
    wal_vault: Vault<WAL>,
    /// keep the protocol fee
    protocol_wal_vault: Vault<WAL>,
    /// TreasuryCap of the wrapped token
    hawal_treasury_cap: TreasuryCap<HAWAL>,
    /// total staked wal amount in history
    total_staked: u64,
    /// total unstaked wal amount in history
    total_unstaked: u64,
    /// total rewards in history
    total_rewards: u64,
    /// total protocol fees in history
    total_protocol_fees: u64,
    /// uncollected protocol fees
    uncollected_protocol_fees: u64,
    /// the number value for haWAL supply, convenient for computing and querying
    hawal_supply: u64,
    pause_stake: bool,
    pause_unstake: bool,
    /// active validators in current epoch, updated every epoch.
    active_validators: vector<ID>,
    /// validators that have stakes, could be ordered by APY
```

```
    validators: vector<ID>,
    /// pools for validators
    pools: Table<ID, PoolInfo>,
    user_selected_validator_bals: VecMap<ID, Balance<WAL>>,
    rewards_last_updated_epoch: u64
}
```

However, the current protocol logic does not use this field.

Suggestion:

If this field is unnecessary, it is recommended to remove it to optimize storage and reduce complexity. If this field is intended to implement certain functionality, it is recommended to add the corresponding logic to avoid misleading design and ensure consistency in the protocol implementation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-9 Non-Standard Comments

Severity: Informational

Status: Fixed

Code Location:

sources/walstaking.move#613-631

Descriptions:

Most of the protocol uses English comments, but some sections still contain Chinese comments.

```
//
let epoch_diff = (current_epoch - walrus_start_epoch) as u64;
let current_epoch_start = walrus_start_timestamp_ms + (epoch_diff *
walrus_epoch_duration);
//
let current_epoch_end = current_epoch_start + walrus_epoch_duration;
//
let mid_epoch_time = current_epoch_start + (walrus_epoch_duration / 2);
```

Suggestion:

Standardize comments to ensure consistency throughout the codebase.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-10 Hardcoded Values in Initialization

Severity: Informational

Status: Fixed

Code Location:

sources/walstaking.move#169-172

Descriptions:

The `initialize` function call uses magic numbers `0` and `100000` instead of named constants, reducing code clarity and maintainability.

```
let sc = config::new(DEFAULT_DEPOSIT_FEE_RATE,
        DEFAULT_REWARD_FEE_RATE, DEFAULT_VALIDATOR_REWARD_FEE_RATE,
         DEFAULT_SERVICE_FEE_RATE, EPOCH_FIRST_TWENTY_HOURS_MILI,
DEFAULT_VALIDATOR_COUNT,0,100000,EPOCH_DURATION);
```

Suggestion:

Declare descriptive constants for these values.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-11 Unused `unstaked_amount` Calculation

Severity: Informational

Status: Fixed

Code Location:

sources/walstaking.move#402;

sources/walstaking.move#532

Descriptions:

In the `do_validator_request_withdraw` and `do_validator_withdraw` functions, the `unstaked_amount` is calculated as follows:

```
unstaked_amount = unstaked_amount + withdraw_principal + withdraw_rewards;
```

However, the calculated value is not used in any subsequent logic.

Suggestion:

It is recommended to remove this calculation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

WAL-12 Dependence on Centralized Storage Service for image URL

Severity: Informational

Status: Acknowledged

Code Location:

sources/walstaking.move#207

Descriptions:

The current implementation uses a centralized storage service for the image URL:

```
fun init(otw: WALSTAKING, ctx: &mut TxContext) {
        let publisher = package::claim(otw, ctx);
        let keys = vector[utf8(b"name"), utf8(b"image_url"), utf8(b"description")];

        let values = vector[
            // Let's add a demo name for our `DemoBear`
            utf8(b"Hawal-UnstakeTicket"),
            // Adding a happy bear image.
            utf8(
                b"https://www.haedal.xyz/hawal.svg",
            ),
            // Description is static for all bears out there.
            utf8(b"To Claim your WAL"),
        ];
```

Suggestion:

It is redommended to use decentralized storage solutions such as IPFS or Arweave.

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.