Haedal Hawal

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Oct 13 2025

Haedal Hawal Audit Report

1 Executive Summary

1.1 Project Information

Description	Haedal is a prime liquid staking protocol natively built on Sui. It provides users with robust liquid staking infrastructure, allowing anyone to stake their SUI & WAL tokens to contribute to the governance and decentralization of the network, while earning continual consensus rewards and unleashing LST liquidity to be used in DeFi On top of its liquid staking protocol, Haedal is also building a series of simple yield products including Haedal Market Maker and more, which generate continuous additional on-chain yields for Haedal and its LST ecosystem Haedal serves as a core pillar of the Sui DeFi by merging native liquid staking and yield strategies with user-friendly accessibility. Aim to empower users to maximize capital efficiency through innovative liquid staking and algorithmic DeFi yield solutions, and build Haedal into the ultimate place to stake and earn on Sui
Type	Staking

Auditors	MoveBit
Timeline	Tue Aug 26 2025 - Mon Oct 13 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/haedallsd/hawal-contract
Commits	4629741a9d9a01c47aa6e93718d998e95e274247 4a502e6c095da654e9e610c9fbe4764e345e07e9 8a6bd7d1e942543587b02c55b1f85bba6b05a9a5

1.2 Files in Scope

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

ID	File	SHA-1 Hash
HAW	sources/hawal.move	2655277d45f22eb65a6304f12ac9bc3613814a20
VAU	sources/vault.move	87748894f1494a8a576b29cf428a607e8f9d4ce9
MOV5	Move.toml	64704ac83065c5718222f45a5aa903a71b4c4de2
ROB	sources/robot.move	a75e933195c44afac2f334df8de640021dc0e3c1
BRE	sources/breaker.move	84815240aa2a28a33badbd6e999d563c60fb3303
OPE	sources/operate.move	0896aba4bcdbe13a9fbb6de9eef419bd4e38d873
MAN	sources/manage.move	73c577a02c829b73a3250092b6b53d1712d61396

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	2	1	1
Informational	0	0	0
Minor	1	0	1
Medium	1	1	0
Major	0	0	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 to identify any potential issues and vulnerabilities in the source code of the Haedal 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 2 issues of varying severity, listed below.

ID	Title	Severity	Status
MAN-1	Error Code Missing and Deletion Function Usage Error Code	Medium	Fixed
MAN-2	Centralization Risk	Minor	Acknowledged

3 Participant Process

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

Admin:

- `set_operator_cap_to_address` : Issue an OperatorCap to a specific account, used by off-chain programs.
- `share_acl` : Create and share a new ACL object (containing minor_signs, breakers, and robots).
- `add_minor_signs_to_acl` : Add a new address to the ACL's minor_signs list.
- `del_minor_signs` : Remove an address from the ACL's minor_signs list.
- `add_breaker_to_acl` : Add a new address to the ACL's breakers list.
- `del_breaker_to_acl` : Remove an address from the ACL's breakers list (currently has a bug with the error code).
- `add_robot_to_acl` : Add a new address to the ACL's robots list.
- `del_robot_to_acl` : Remove an address from the ACL's robots list.
- `migrate` : Migrate staking data after a contract upgrade.
- `request_collect_rewards_fee` : Request to collect rewards fee (currently not implemented, always aborts).
- `claim_collect_rewards_fee` : Claim rewards fee on behalf of a user.
- `set_deposit_fee_v2` : Set the deposit fee.
- `set_reward_fee_v2` : Set the reward fee.
- `set_validator_reward_fee_v2` : Set the validator reward fee.
- `set_service_fee_v2` : Set the service fee.
- `claim_collect_rewards_fee_v2` : Claim rewards fee on behalf of a user (v2 interface).

- `claim_collect_protocol_fee_v2` : Claim protocol fee on behalf of a user. toggle_stake: Enable or disable staking functionality globally.
- `toggle_unstake` : Enable or disable unstaking functionality globally.
- `toggle_claim` : Enable or disable reward claiming globally.
- `update_validator_rewards` : Update rewards for a specific validator during epoch transitions.
- `sort_validators` : Sort validators in a specific order (e.g., for ranking or reward distribution).
- `migrate` : Migrate staking data after a contract upgrade.
- `request_collect_rewards_fee` : Request reward fee collection (currently not implemented, always aborts).
- `claim_collect_rewards_fee` : Claim reward fees on behalf of a user.
- `claim_collect_protocol_fee` : Claim protocol fees on behalf of a user.
- `update_validator_rewards_v2` : Update rewards for a specific validator.
- `sort_validators_v2` : Sort validators according to a provided list.
- `validator_offline_v2` : Mark a validator as offline, updating staking state accordingly.
- `set_withdraw_time_limit_v2` : Set the withdrawal time limit for staking.
- `set_validator_count_v2` : Set the maximum number of validators.
- `set_active_validators_v2` : Define the set of active validators.
- `toggle_stake_v2` : Enable or disable staking.

4 Findings

MAN-1 Error Code Missing and Deletion Function Usage Error Code

Severity: Medium

Status: Fixed

Code Location:

sources/manage.move#18,74,87,100

Descriptions:

In the haedal::manage module, only two error codes are defined:

```
const EInitialized: u64 = 1;
const EAccountExist: u64 = 2;
```

However, these error codes are reused in opposite semantic contexts:

Add (add_*) functions: They assert that the account should not already exist:

```
assert!(!vector::contains(&acl.minor_signs, &account), EAccountExist);
```

Here EAccountExist is used to mean already exists, which is acceptable.

Delete (del_*) functions: They assert that the account must exist before removal:

```
assert!(is_exist, EAccountExist);
```

If the account is missing, the function aborts with `EAccountExist`. This incorrectly signals "already exists" when in reality the account was not found.

As a result, the same error code is used to represent two opposite cases (“already exists” vs. “not found”), leading to ambiguity and confusion.

Affected functions include: `del_minor_signs`, `del_breaker_to_acl`, and `del_robot_to_acl`.

Suggestion:

Add error codes and modify the error codes used by the delete function

Resolution:

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

MAN-2 Centralization Risk

Severity: Minor

Status: Acknowledged

Code Location:

sources/manage.move

Descriptions:

The module defines multiple Caps:

- AdminCap
- OperatorCap
- MinorSignCap
- BreakerCap
- RobotCap

Currently, all critical privilege assignments are fully controlled by AdminCap, for example:

```
public entry fun set_operator_cap_to_address(_: &AdminCap, account: address, ctx: &mut TxContext)
public entry fun set_minor_sign_cap_to_address(_: &AdminCap, account: address, ctx: &mut TxContext)
public entry fun set_breaker_cap_to_address(_: &AdminCap, account: address, ctx: &mut TxContext)
public entry fun set_robot_cap_to_address(_: &AdminCap, account: address, ctx: &mut TxContext)
```

This implies:

- All critical capability issuance is centralized in a single administrator.
- If the AdminCap is misused or compromised, an attacker can fully control the system, including issuing arbitrary OperatorCap, BreakerCap, or RobotCap, and performing data migration or upgrade operations.

- The system lacks decentralization or multi-signature protection, creating a single point of failure.

Suggestion:

It is recommended that measures be taken to reduce the risk of centralization, such as a multi-signature mechanism.

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.