Haedal Protocol

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Oct 13 2025

Haedal Protocol 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/haedal-protocol
Commits	24e8e0d5b678c0e4d4b720e6edcfbd1a5984498f 08f9ea76aeacd32f4c44489e3415be7774a8f01e 510212e965c9fb2b666d5e643b1c28d3b9871450

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	Move.toml	84a1e54090ef5b95fcb9b2d473f7b84574397ad9
HAS	sources/hasui.move	043c51935bddf988961f1a5790f253dda55b90f6
VAU	sources/vault.move	c4f6a0535cfb5b57379e78c401b2142831aaddec
ROB	sources/robot.move	6fff8e053e18b50bce3f73948c338f76140183f8
BRE	sources/breaker.move	51f3ab242ccf0d4e08b25b556edafcb2eb38d0e9
OPE	sources/operate.move	b7a654824f55989fa41604db5185b78eda09a068
STA	sources/staking.move	4c20cd95b5faa1586546c5b05caff050e3c310ed
MAN	sources/manage.move	0158552b38611c810e9ae38cd2c64498c6c09abc

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 Protocol 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	Incorrect Use of Assertion Error Codes	Medium	Fixed
MAN-2	Centralization Risk	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Haedal Protocol 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 Incorrect Use of Assertion Error Codes

Severity: Medium

Status: Fixed

Code Location:

sources/manage.move#94

Descriptions:

In `del_breaker_to_acl`, confirm that the account exists before deleting it:

```
let (is_exist, index) = vector::index_of(&acl.breakers, &account);
assert!(is_exist, EAccountExist);
vector::remove(&mut acl.breakers, index);
```

But the error code used by the assertion is `EAccountExist` (literally meaning "account already exists"). When account does not exist, the assertion fails and throws `EAccountExist` - which is the opposite of the actual situation.

Suggestion:

Change assertion error codes to semantically correct error constants.

```
let (is_exist, index) = vector::index_of(&acl.breakers, &account);
assert!(is_exist, EAccountNotExist);
vector::remove(&mut acl.breakers, index);
```

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