Haedal Vault

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Mon Mar 10 2025

Haedal Vault Audit Report

1 Executive Summary

1.1 Project Information

Description	VolatileVault is a concentrated liquidity vault protocol built on Sui that enables automated liquidity management and yield generation
Type	DeFi
Auditors	MoveBit
Timeline	Wed Mar 05 2025 - Mon Mar 10 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/haedallsd/vaults
Commits	bab0890ab983efe86e32f73bfb7a6a7a63b6298b 7a0e9c759da2175c1c942083a75f50409a2d4b4d 70830a62838cc4b3b161ca8819d48a3d4543f42f

1.2 Files in Scope

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

ID	File	SHA-1 Hash
ACL	sources/acl.move	c8282612393b44703bfc065a41b8fd177f510f7b
MOV	Move.toml	f55d64f5f351016f611b8397d6fe6c0f5fdaf783
BBA	sources/balance_bag.move	66aff5736a650dc1b2c952340e011cde8704fb59
CVA	sources/clmm_vault.move	11308d6f08314373f8205b00b829d1a69efb8088
POR	sources/pyth_oracle.move	560210640c28056acba57f4cd47db32c445c021f
CON	sources/config.move	c1bdf69ea86409ac23a3e2ae22611ae2d66ecc3f
ERR	sources/error.move	08957b2bef81b45d06703bb316d6e36e09c5e7d2
UTI	sources/utils.move	d1258bd6382e6485c1e8685214024cda2a45e8bd
POO	sources/pool.move	28ab909b9b821d8b858a58db8ec989b03266db30

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	0	0	0
Minor	1	1	0
Medium	4	4	0
Major	1	1	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 Haedal Vault 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 6 issues of varying severity, listed below.

ID	Title	Severity	Status
BBA-1	Change `public` to `public(package)`	Minor	Fixed
POO-1	Missing Slippage Protection for Adding Liquidity	Major	Fixed
POO-2	Temporarily Disable other Operations when Performing the Flash Loan	Medium	Fixed
POO-3	Missing Check for the `pool.is_pause`	Medium	Fixed
POO-4	Missing Check for whether the Fee and Reward have been Claimed	Medium	Fixed
POO-5	The Protocol does not Use the Updated Price when calling `withdraw()`	Medium	Fixed

3 Participant Process

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

Admin:

- `set_roles()` : Sets all roles for a member in the access control list.
- `add_role()` : Adds a specific role to a member in the access control list.
- `remove_role()` : Revokes a specific role from a member in the access control list.
- `remove_member()` : Removes a member from the access control list.
- `update_package_version()` : Updates the package version number in the GlobalConfig.

Manager:

- `update_protocol_fee_rate()` : Updates the protocol fee rate that is charged on operations.
- `set_swap_slippage` : Sets the swap slippage configuration for a coin type.
- `create_pool()` : Creates a new liquidity pool with the specified parameters.
- `update_protocol_fee()` : Updates the protocol fee rate for a pool.
- `update_hard_cap()` : Updates the hard cap (maximum total value locked) for a pool.
- `update_liquidity_offset()` : Updates the liquidity offset parameters for a pool's CLMM vault and triggers a rebalance.
- `update_rebalance_threshold()` : Updates the rebalance threshold parameter for a pool's CLMM vault.
- `unpause()` : Unpauses a pool, allowing operations to resume.
- `pause()` : Pauses a pool, preventing any operations until unpaused.
- `add_oracle_info()` : Adds oracle information for a specific coin type to the PythOracle.
- `remove_oracle_info()` : Removes oracle information for a specific coin type from the PythOracle.
- `update_price_age()` : Updates the maximum allowed age for price data for a specific coin type in the PythOracle.

- **Rebalance Role**:
- `rebalance()` : Rebalances the pool's concentrated liquidity position based on oracle prices.

User:

- `collect_reward()` : Collects rewards from the CLMM pool.
- `collect_fee()` : Collects accumulated trading fees from the CLMM pool position and distributes them between protocol and buffer.
- `update_price()` : Updates the price feed for a given coin type in the Pyth oracle.
- `calculate_aum()` : Calculates the Assets Under Management (AUM) for a pool.
- `deposit()` : Deposit to the pool.
- `withdraw()` : Withdraw from the pool by burning LP tokens and withdrawing underlying assets.
- `withdraw_buffer_reward()` : Removes a specific asset from the pool based on the user's LP token share.
- `destory_withdraw_cert()` : Destory and finalizes a withdrawal certificate after all assets have been withdrawn.
- `add_liquidity()` : Add liquidity assets from the pool's buffer into the CLMM vault position.
- `deposit_fee()` : Deposits a fee into the PythOracle's fee balance.

4 Findings

BBA-1 Change `public` to `public(package)`

Severity: Minor

Status: Fixed

Code Location:

sources/balance_bag.move#80-89

Descriptions:

The `join()`, `withdraw_all()`, and `split()` functions are currently only called within the pool contract.

```
public fun add_liquidity<CoinTypeA, CoinTypeB, T>(
    pool: &mut Pool<T>,
    config: &GlobalConfig,
    clmm_config: &ClmmConfig,
    clmm_pool: &mut ClmmPool<CoinTypeA, CoinTypeB>,
    clk: &Clock,
) {
    config.checked_package_version();
    let mut balance_a = pool.buffer_assets.withdraw_all();
    let mut balance_b = pool.buffer_assets.withdraw_all();
```

To enhance security and restrict external access, it is recommended to change the visibility of these functions from public to public(package).

Suggestion:

It is recommended to change the visibility of these functions from public to public(package)

Resolution:

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

POO-1 Missing Slippage Protection for Adding Liquidity

Severity: Major

Status: Fixed

Code Location:

sources/pool.move#335

Descriptions:

The `add_liquidity()` function allows users to add liquidity assets from the pool's buffer into the CLMM vault position.

```
public fun add_liquidity<CoinTypeA, CoinTypeB, T>(
    pool: &mut Pool<T>,
    config: &GlobalConfig,
    clmm_config: &ClmmConfig,
    clmm_pool: &mut ClmmPool<CoinTypeA, CoinTypeB>,
    clk: &Clock,
) {
    config.checked_package_version();
    let mut balance_a = pool.buffer_assets.withdraw_all();
    let mut balance_b = pool.buffer_assets.withdraw_all();
    let (amount_a, amount_b, delta_liquidity) = clmm_vault::increase_liquidity(
        &mut pool.clmm_vault,
        clmm_config,
        clmm_pool,
        &mut balance_a,
        &mut balance_b,
        clk,
    );
```

In this function, the protocol uses `current_sqrt_price` to calculate `delta_liquidity`, `need_a`, and `need_b`.

```
let (current_tick_index, current_sqrt_price) = (
        clmm_pool.current_tick_index(),
        clmm_pool.current_sqrt_price(),
    );
    let (tick_lower, tick_upper) =
position::tick_range(clmm_vault.wrapped_position.borrow());
```

```
let (delta_liquidity, need_a, need_b) = clmm_math::get_liquidity_by_amount(
    tick_lower,
    tick_upper,
    current_tick_index,
    current_sqrt_price,
    amount_a,
    true,
);
```

However, since `current_sqrt_price` can be easily manipulated by swapping a large amount of assets, this could result in `delta_liquidity` not matching the user's expectations.

Suggestion:

It is recommended to ensure that `delta_liquidity` is greater than the user's expected minimum value.

Resolution:

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

POO-2 Temporarily Disable other Operations when Performing the Flash Loan

Severity: Medium

Status: Fixed

Code Location:

sources/pool.move#1171

Descriptions:

The `flash_loan()` function enables users to borrow one type of asset against another, with price verification using an oracle.

1. The protocol first checks that `pool.is_pause` is `false` before executing the flash loan logic.

```
public fun flash_loan<CoinTypeA, CoinTypeB, T>(
    pool: &mut Pool<T>,
    config: &GlobalConfig,
    pyth_oracle: &PythOracle,
    loan_amount: u64,
    clk: &Clock,
    ctx: &mut TxContext,
): (Coin<CoinTypeA>, FlashLoanCert) {
    config.checked_package_version();
    config.check_operation_role(ctx.sender());
    assert!(!pool.is_pause, error::pool_is_pause());
    assert!(loan_amount > 0, error::token_amount_is_zero());
```

2. It is recommended to set `pool.is_pause` to `true` when performing the flash loan to temporarily disable other operations.
   For example: https://github.com/pontem-network/liquidswap/blob/main/sources/swap/liquidity_pool.move#L389

3. Once the loan is repaid, `pool.is_pause` should be set back to `false` in the `repay_flash_loan()` function to re-enable normal operations.

This mechanism helps prevent users from performing unintended actions during the flash loan process.

Suggestion:

It is recommended to set pool.is_pause to true when performing the flash loan to temporarily disable other operations.

Resolution:

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

POO-3 Missing Check for the `pool.is_pause`

Severity: Medium

Status: Fixed

Code Location:

sources/pool.move#1499-1527

Descriptions:

The `add_liquidity()` function allows users to add liquidity assets from the pool's buffer into the CLMM vault position. However, there is no `pool.is_pause` check.

```
public fun add_liquidity<CoinTypeA, CoinTypeB, T>(
    pool: &mut Pool<T>,
    config: &GlobalConfig,
    clmm_config: &ClmmConfig,
    clmm_pool: &mut ClmmPool<CoinTypeA, CoinTypeB>,
    clk: &Clock,
) {
    config.checked_package_version();
    let mut balance_a = pool.buffer_assets.withdraw_all();
    let mut balance_b = pool.buffer_assets.withdraw_all();
```

Although it is usually called within the `deposit()` function, the `add_liquidity()` function is public and can be called directly.

Suggestion:

It is recommended to verify whether the `pool.is_pause` is false.

Resolution:

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

POO-4 Missing Check for whether the Fee and Reward have been Claimed

Severity: Medium

Status: Fixed

Code Location:

sources/pool.move#1603-1627

Descriptions:

The `rebalance()` function is used to adjust the pool's concentrated liquidity position based on oracle prices. The function closes the existing liquidity position and opens a new one at an optimized range around the current price. However, the protocol does not verify whether the fees and rewards from the existing position have been claimed before rebalancing.

```
public fun rebalance<CoinTypeA, CoinTypeB, T>(
    pool: &mut Pool<T>,
    config: &GlobalConfig,
    pyth_oracle: &PythOracle,
    clmm_config: &ClmmConfig,
    clmm_pool: &mut ClmmPool<CoinTypeA, CoinTypeB>,
    clk: &Clock,
    ctx: &mut TxContext,
) {
    config.checked_package_version();
    config.check_rebalance_role(ctx.sender());
    assert!(!pool.is_pause, error::pool_is_pause());
    assert!(object::id(clmm_pool) == pool.clmm_vault.pool_id(),
error::clmm_pool_not_match());
```

If the rebalance role forgets to claim these fees and rewards, it could result in financial losses.

Suggestion:

It is recommended to add the assertion `clmm_vault::assert_fee_reward_claimed(clmm_config, clmm_pool, &pool.clmm_vault, clk);` .

Resolution:

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

POO-5 The Protocol does not Use the Updated Price when calling `withdraw()`

Severity: Medium

Status: Fixed

Code Location:

sources/pool.move#1052-1148

Descriptions:

According to the README.md, the withdrawal flow includes an `update_price` process, which requires the protocol to pay a fee when updating the price.

```
#### 2.2 Withdraw Flow
- collect_reward
- collect_fee
- update_price
- withdraw
- withdraw_buffer_reward
- destory_withdraw_cert
```

```
public fun update_price<T>(
    pyth_oracle: &mut PythOracle,
    config: &GlobalConfig,
    pyth_state: &State,
    price_updates: HotPotatoVector<PriceInfo>,
    price_info_obj: &mut PriceInfoObject,
    clk: &Clock,
    ctx: &mut TxContext,
): HotPotatoVector<PriceInfo> {
    config.checked_package_version();

    let fee = pyth_oracle.split_fee(pyth_state.get_base_update_fee()).into_coin(ctx);
    let h = pyth::update_single_price_feed(
        pyth_state,
        price_updates,
        price_info_obj,
        fee,
        clk,
  );
```

However, in the actual implementation of the withdraw() function, the protocol does not use the updated price.

Suggestion:

It is recommended to avoid updating the price.

Resolution:

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

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