Haedal - haevault & farming

Security Assessment

CertiK Assessed on Oct 27th, 2025

CertiK Assessed on Oct 27th, 2025

Haedal - haevault & farming

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Vault	Sui (SUI)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Move	Preliminary comments published on 10/03/2025
	Final report published on 10/28/2025

Vulnerability Summary

29	14	0	15	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

1	Centralization	1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	2 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
23	Informational	12 Resolved, 11 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | HAEDAL - HAEVAULT & FARMING

CODEBASE | HAEDAL - HAEVAULT & FARMING

Repository

https://github.com/haedallsd/haevault/tree/606d7f7dc6c1b9edbfb9e1c6dc1ac7171dba6897

https://github.com/haedallsd/farming-contract/tree/1f779684e65233412208cc80c72da575eb054cf6

https://github.com/haedallsd/farming-contract/tree/0dd27c49adc83b31df7566289a5a0c37d226da49

https://github.com/haedallsd/haevault/tree/e867cc394e36d27165c39b305107bd7537c2e254

https://github.com/haedallsd/farming-contract/tree/7835153827bd4ec271a27b099fd1ae53c991cf85

https://github.com/haedallsd/haevault/tree/25c8b59078c8d6f6d424bd58a725f25cb413c74f

https://github.com/haedallsd/haevault/tree/b515cfed342ead873e6a545b972b4e42b3415fd9

https://github.com/haedallsd/haevault/tree/06aa2493c67d864b025f8832007928e6502f7fb8

Commit

606d7f7dc6c1b9edbfb9e1c6dc1ac7171dba6897

1f779684e65233412208cc80c72da575eb054cf6

0dd27c49adc83b31df7566289a5a0c37d226da49

e867cc394e36d27165c39b305107bd7537c2e254

7835153827bd4ec271a27b099fd1ae53c991cf85

25c8b59078c8d6f6d424bd58a725f25cb413c74f

b515cfed342ead873e6a545b972b4e42b3415fd9

06aa2493c67d864b025f8832007928e6502f7fb8

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | HAEDAL - HAEVAULT & FARMING

This audit was conducted for Haedal to evaluate the security and correctness of the smart contracts associated with the Haedal - haevault & farming project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Formal Verification, Manual Review, and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | HAEDAL - HAEVAULT & FARMING

Overview

The **Haedal - haevault & farming** is a vault and farming protocol built on Sui, enabling users to deposit their assets and earn rewards.

External Dependencies

The project is developed using the Move language and running on the top of the Sui blockchain. The vulnerability and the updates of the language/Sui framework may affect the project as a whole. As the Sui network is rapidly evolving, to avoid any potential compatibility issues and take advantage of new features and improvements, the client should upgrade the Sui framework to the most recent version. Additionally, staying informed about any upcoming updates or changes to the language or framework can help ensure the project remains secure and compatible.

Dependency of the **Haedal - haevault**:

```
Sui = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework/packages/sui-framework", rev = "mainnet-v1.42.2", override = true }
IntegerMate = { git = "https://github.com/CetusProtocol/integer-mate.git", subdir = "sui", rev = "mainnet-v1.23.1", override = true }
CetusClmm = { git = "https://github.com/CetusProtocol/cetus-clmm-interface.git", subdir = "sui/cetus_clmm", rev = "mainnet-v1.26.0", override = true }
```

The haevault also relies on the third-party **Pyth** oracle for the full functionalities, which recommends the team keep monitoring the pyth price feed to avoid unexpected price results.

Dependency of the **Haedal - farming**:

```
Sui = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework/packages/sui-framework", rev = "framework/testnet", override = true }
```

The above dependencies are not within the current audit scope and serve as a black box. Modules/Contracts within the module are assumed to be valid and non-vulnerable actors in this audit and implement proper logic to collaborate with the current project and other modules.

Privileged Roles

To set up the project correctly and ensure that the project functions properly, owners of the following objects are able to use privileged functions, more details in **GLOBAL-01: Centralization Related Risks And Upgradability**.

The advantage of the privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to serve the community best. It is also worthy of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the key pairs of privileged accounts are compromised, the project could have devastating consequences.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Furthermore, any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

Upgradeability

Developers working with the Sui blockchain have the ability to upgrade packages based on their software iteration requirements. However, this also means that the `UpgradeCap` and publisher's key store should be handled with caution to prevent any unexpected loss. Additionally, it is important to inform the community about any upgrade plans to address concerns related to centralization and ensure transparency.

Reference:

- Sui Package Upgrades
- Custom Policies

FINDINGS | HAEDAL - HAEVAULT & FARMING

29	0	1	0	0	5	23
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Haedal to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 29 issues were identified. Leveraging a combination of Formal Verification, Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
HH&-14	**Centralization Related Risks And Upgradability**	**Centralization**	**Centralization**	**Acknowledged**
HH&-15	Farming Contract May Cause DoS Due To Insufficient Funds	Denial of Service	Minor	Acknowledged
HH&-16	`flip_boost()` Causes Inconsistent Rewards For Users With Boost	Inconsistency, Design Issue	Minor	Acknowledged
HH&-17	Missing Check Between The Input `clmm_pool` And `pool`	Inconsistency	Minor	Resolved
HH&-18	Inconsistent Flashloan Repay Amount	Volatile Code	Minor	Acknowledged
HHF-01	Inconsistent Code And Error Message In `admin_change_position`	Coding Issue	Minor	Resolved
HH&-02	Discussion On User `set_boost()`	Design Issue	Informational	Acknowledged
HH&-03	Potential Unintended `Cap` Design	Access Control	Informational	Resolved
HH&-04	Discussion On Reward When Paused Pool	Inconsistency	Informational	Acknowledged
HH&-05	Discussion On Potential Incorrect Slippage Application For Repay Amount	Design Issue	Informational	Resolved

ID	Title	Category	Severity	Status
HH&-06	Tick Spacing “Round-To-Zero” May Break Containment Of The Original Range	Design Issue	Informational	Acknowledged
HH&-07	Discussion On Negative Price	Volatile Code	Informational	Resolved
HH&-08	`after_set_boost` Drops Pending Rewards	Volatile Code	Informational	Acknowledged
HH&-09	Discussion On `remove_oracle_info()`	Volatile Code	Informational	Acknowledged
HH&-10	Discussion On External Funding For `update_price_fee`	Logical Issue	Informational	Resolved
HH&-11	Rebalance Lacks Oracle/Pool Price Deviation Cap Used By Add-Liquidity	Volatile Code	Informational	Acknowledged
HH&-12	Discussion On Borrowing Scope In `flash_loan()`	Volatile Code	Informational	Acknowledged
HH&-13	Inconsistent On Assets Between Deposit And Withdraw	Volatile Code	Informational	Acknowledged
HH&-19	Inconsistency Between Code And Comment	Volatile Code	Informational	Resolved
HH&-20	Missing Check On `lower_offset` And `upper_offset`	Volatile Code	Informational	Resolved
HH&-21	Unintended Abort Caused By Missing Oracle Info Validation In TVL Calculation	Volatile Code	Informational	Resolved
HH&-22	DoS Risk Of Using VecMap	Denial of Service	Informational	Acknowledged
HH&-23	Missing Check On `usd_price_age`	Volatile Code	Informational	Resolved
HH&-24	Missing Pool-Deposit Consistency Check In Remaining	Coding Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
HH&-27	Missing Validation On Zero `rebalance_threshold`	Volatile Code	Informational	Acknowledged
HH&-29	Outdated Dependency	Volatile Code	Informational	Resolved
HHF-02	Discussion On `update_package_version()` And `emergency_pause()`	Logical Issue	Informational	Resolved
HHF-03	Discussion On Inconsistent Doc	Volatile Code	Informational	Resolved
HHF-04	Decimal Parity Can Break Round-Trip When P+O Is Odd	Volatile Code	Informational	Resolved

HH&-14 | Centralization Related Risks And Upgradability

Category	Severity	Location	Status
Centralization	**Centralization**		**Acknowledged**

Description

In the module **config**, the role **AdminCap** has authority over the functions:

- set_roles()
- add_role()
- remove_role()
- remove_member()
- update_package_version()
- emergency_unpause()

In the module **config**, the role **ACL_POOL_MANAGER** has authority over the functions:

- update_protocol_fee_rate()
- update_max_price_deviation_bps()
- set_swap_slippage()

In the module **config**, the role **ACL_EMERGENCY_PAUSE** has authority over the functions:

- emergency_pause()

In the module **pool**, the role **ACL_POOL_MANAGER** has authority over the functions:

- create_pool_v2()
- update_protocol_fee()
- update_hard_cap()
- update_liquidity_offset()
- update_rebalance_threshold()
- unpause()
- pause()

In the module **pool**, the role **ACL_REINVEST_MANAGER** and **ACL_REBALANCE_MANAGER** have authority over the functions:

- flash_loan()
- repay_flash_loan()

In the module **pool**, the role **ACL_CLAIM_PROTOCOL_FEE** has authority over the functions:

- claim_protocol_fee()

In the module **pool**, the role **ACL_REBALANCE_MANAGER** has authority over the functions:

- update_liquidity_offset_only()
- rebalance()
- admin_change_position()

In the module **pyth_oracle**, the role **ACL_ORACLE_MANAGER** has authority over the functions:

- add_oracle_info()
- remove_oracle_info()
- update_price_age()

In the module **breaker**, the role **breaker** has authority over the functions:

- funding_bank()

In the module **manage**, the role **AdminCap** has authority over the functions:

- set_operator_cap_to_address()
- share_acl()
- add_minor_signs_to_acl()
- del_minor_signs()
- add_breaker_to_acl()
- del_breaker_to_acl()
- add_robot_to_acl()
- del_robot_to_acl()
- migrate()

In the module **minorsign**, the role **minor_signs** has authority over the functions:

- add_pool()
- set_pool()
- add_reward_config()
- set_reward_config()
- flip_boost()

In the module **operate**, the role **OperatorCap** has authority over the functions:

- add_pool()
- set_pool()
- add_reward_config()

- set_reward_config()
- flip_boost()
- set_boost()
- after_set_boost()
- after_set_boost_batch()
- funding_bank()
- extract_bank()

In the module **robot**, the role **robot** has authority over the functions:

- set_boost()
- after_set_boost()
- after_set_boost_batch()
- settle_batch()

If any of these privileged accounts are compromised, an attacker could leverage their granted permissions to alter protocol states, create new pools, manipulate oracle data and deposit processes, upgrade or pause the contract, or modify operator lists.

In addition, developers working with the Sui blockchain can upgrade packages based on their software iteration requirements. However, this also means that the `UpgradeCap` and deployer's key store should be handled carefully to prevent any unexpected losses. It is important to inform the community about any upgrade plans to address concerns related to centralization and ensure transparency.

More information can be found:

- Sui Package Upgrades
- Third-Party Package upgrades

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Haedal, 10/10/2025]: Yes, it's time to use ACL address lists to strengthen restrictions.

1. Upgrade BreakerCap, MinorSignCap, and RobotCap to ACL address lists.
2. AdminCap and OperatorCap will remain. Thanks to these two caps, we will ensure multi-signature support with at least four administrators.
3. The team will also regularly review the private key security of all addresses associated with centralized roles.

[CertiK, 10/13/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

HH&-15 | Farming Contract May Cause DoS Due To Insufficient Funds

Category	Severity	Location	Status
Denial of Service	Minor	sources/core.move (farming): 696, 709	Acknowledged

Description

The farming-contract currently does not have a self-sustaining economic mechanism to ensure that the banks corresponding to each `RewardConfig` have sufficient funds to pay users. Instead, it relies on external injections into the banks via `funding_bank()`. The project team must ensure that the banks have enough funds to prevent situations where users are unable to claim rewards through `harvest` due to insufficient bank balance.

Recommendation

The project team must ensure that the banks have enough funds to prevent situations where users are unable to claim rewards through `harvest` due to insufficient bank balance.

Alleviation

[Haedal, 10/10/2025]: The team will ensure that there are sufficient funds in the bank, and the off-chain robot will always monitor the funds in the prize pool to ensure that the prize pool is not zero.

HH&-16 `flip_boost()` Causes Inconsistent Rewards For Users With Boost

Category	Severity	Location	Status
Inconsistency, Design Issue	Minor	sources/core.move (farming): 720~735	Acknowledged

Description

The `flip_boost()` function allows privileged roles to toggle the `boost_active` status of a `reward_config` at any time. However, the toggle only modifies the state of the `RewardConfig` and does not update all Deposits or adjust user-specific state in their Deposits. In this case, users' next reward settlement may be inconsistent, potentially resulting in overpayment or underpayment of rewards.

Also, a potential DoS would happen when the current reward is less than the historical debt due to a share change(from boost to non-boost):

```
fun get_reward<DepositCoin>(
    deposit: &Deposit<DepositCoin>,
    reward_config: &RewardConfig,
    coin_type: &TypeName,
): u64 {
    let reward =
        (
            (deposit.share_in_the_reward_config(reward_config) as u128) *
reward_config.per_share / SCALING_FACTOR,
        ) as u64;
    reward - get_debt<DepositCoin>(deposit, coin_type)
}
```

Scenario

For example, suppose the reward is in the state `boost_active == false` during [100, 200) and `boost_active == true` during [200, 300), meaning `flip_boost()` is called at time 200. During [100, 300), `pool.total_share` and `pool.total_boost_share` are both 1,000,000, and the `reward_rate` is 1,000,000, so the per-second `per_rate` increment is 1 (ignoring the SCALING_FACTOR precision). Account A has a Deposit of 10 throughout [100, 300) with a boost of 900, giving `share` and `boost_share` of 10,000 and 9,000, respectively. There are only two reward-related transactions during [100, 300): Account A's `deposit()` at 100 and the `flip_boost()` at 200.

If user A calls `settle()` at 300, they will use the 9,000 `boost_share` to calculate rewards over the 200 seconds, resulting in an 9,000 × 200 × 1 = 1,800,000 reward. However, during [100, 200), before `flip_boost()`, the calculation should have used 10,000 `share`, so the correct reward should have been 10,000 × 100 × 1 + 9,000 × 100 × 1 = 1,900,000. Thus, the user receives 100,000 less reward than intended.

Recommendation

We recommend revisiting the design of `flip_boost()`. For example, it could be restricted so that toggling does not change all user shares (i.e., `boost_share` and `share` remain equal), ensuring that reward calculations for users are not affected before and after the toggle.

Alleviation

[Haedal, 10/16/2025]: We are aware of this issue. Therefore, we won't be executing stop and start. In other words, we won't be actively changing the state of pool.active. Instead, we'll directly execute the sequence of settle -> flip_boost -> after_set_boost . Due to the relative timing difference in execution order, we'll accept any reward mismatches caused by the settle and after_set_boost operations.

1. boost_share and share must be inconsistent, as we need to set different boost multipliers for each user.

2. We won't be frequently executing flip_boost . In fact, flip_boost determines the boost state from the very beginning of the pool's lifecycle.

3. In extreme cases, if the team needs to execute flip_boost to change the pool's state, we'll do so after reward releases have completed (ensuring no rewards are released during the execution).

Thus, while we accept this issue, it's not a core concern and won't impact overall user returns or the normal operation of the protocol. We're not considering redesigning flip_boost for now.

HH&-17 | Missing Check Between The Input `clmm_pool` And `pool`

Category	Severity	Location	Status
Inconsistency	Minor	sources/pool.move (haevault): 690~691, 884~885	Resolved

Description

The `update_liquidity_offset()` function allows the pool manager to modify the pool's liquidity offset and rebalance the position if necessary. However, the function does not enforce a consistency check between the specified `clmm_pool` and the target pool. If a mismatched `clmm_pool` is used, the function may rely on an incorrect price and tick, potentially leading to an inaccurate rebalance operation.

The same issue also exists in the `get_vault_assets()` function.

Recommendation

It is recommended to enforce a consistency check to ensure that the input `clmm_pool` matches the target `pool`.

Alleviation

[Haedal, 10/14/2025]:

The team heeded the advice and resolved the issue by adding validation on clmm pool and the pool record in the vault in commit 25c8b59078c8d6f6d424bd58a725f25cb413c74f.

HH&-18 | Inconsistent Flashloan Repay Amount

Category	Severity	Location	Status
Volatile Code	Minor	sources/pool.move (haevault): 1351~1352	Acknowledged

Description

The `repay_flash_loan()` function transfers the entire balance of `repay_coin` into `pool.buffer_assets`, without considering the expected repayment amount (`repay_amount`).

```
assert!(repay_coin.value() >= repay_amount, error::incorrect_repay());
assert!(object::id(pool) == pool_id, error::incorrect_repay());

let repay_amount = repay_coin.value();
pool.buffer_assets.join(repay_coin.into_balance());
```

As a result:

- Any excess funds beyond the required `repay_amount` are not refunded to the sender, which can cause unintended financial loss for the user.
- The inflated `buffer_assets` value may also distort subsequent AUM (Assets Under Management) calculations, potentially affecting pool accounting and risk metrics.

Recommendation

It is recommended to accept exactly `repay_amount` from `repay_coin`. Any surplus should be explicitly refunded to the sender. This ensures users do not lose funds and prevents inconsistencies in pool asset accounting.

Alleviation

[Haedal, 10/10/2025]:

The `repay_flash_loan()` function is intended to be used by **keepers managing the vault's asset allocations**.

During the repayment process, as long as the required assets are returned, **any excess funds are considered part of the overall vault assets** and are intentionally retained.

This behavior is by design and aligns with the intended asset management logic.

HHF-01 Inconsistent Code And Error Message In `admin_change_position`

Category	Severity	Location	Status
Coding Issue	Minor	sources/pool.move (haevault-1014): 1785	Resolved

Description

In the `admin_change_position` function, the assert statement requires that the pool is not paused. However, the error message incorrectly states the opposite, indicating that the pool should be paused.

```
    config.check_emergency_restore_version();
    config.check_rebalance_role(ctx.sender());
    assert!(!pool.is_pause, error::pool_is_not_paused());
```

Recommendation

We recommend the team double-check the intended design.

Alleviation

[Haedal, 10/20/2025]: The team heeded the advice and resolved the issue by updating the pause validation in commit b515cfed342ead873e6a545b972b4e42b3415fd9.

HH&-02 | Discussion On User `set_boost()`

Category	Severity	Location	Status
Design Issue	Informational	sources/core.move (farming): 760~766, 768~792	Acknowledged

Description

Regarding setting a user's boost while the pool's reward is currently in boost mode, we have the following issues to query and confirm:

1. We would like to know the range of values for the boost setting, for example, whether it can be greater or less than `BOOST_SCALING_FACTOR`.
2. In the current code, the boost takes effect only upon the next update of the user's debt. However, the functions that update a user's debt (`after_set_boost()` as well as the user-invocable functions) are all executed later than `set_boost()`, which generates the time window between `set_boost` and `after_set_boost` would allowing the user to gain more rewards or suffer reward loss.

Recommendation

We would like to confirm with the team whether this is the intended design.

Alleviation

[Haedal, 10/10/2025]: 1. It can be greater or less than BOOST_SCALING_FACTOR, but is usually greater than BOOST_SCALING_FACTOR.

2. As expected

HH&-03 | Potential Unintended `Cap` Design

Category	Severity	Location	Status
Access Control	Informational	sources/operate.move (farming): 10~29	Resolved

Description

`MinorSignCap`, `BreakerCap`, `RobotCap`, and `OperatorCap` have the `key` and `store` abilities, allowing them to be directly `public_transfer`ed by the holding account to other accounts. This means the granting and revocation of these privileged roles are not determined by the `AdminCap` holder but instead by the respective holders themselves. This would result in these roles not being managed or controlled by the `AdminCap` holder. We would like to ask whether this is the intended design.

```
10 /// `OperatorCap` is used by the offchain programs.
11 public struct OperatorCap has key, store {
12     id: UID,
13 }
14
15 // `MinorSignCap` is used by 3 of 2
16 public struct MinorSignCap has store, key {
17     id: UID,
18 }
19
20 // `BreakerCap` is used by the circuit breaker.
21 public struct BreakerCap has store, key {
22     id: UID,
23 }
24
25 // `RobotCap` is used by the robot.
26 public struct RobotCap has key, store {
27     id: UID,
28 }
```

Recommendation

We would like to ask whether this is the intended design.

Alleviation

[Haedal, 10/13/2025]: The team heeded the advice and resolved the issue in commit 1f779684e65233412208cc80c72da575eb054cf6.

HH&-04 | Discussion On Reward When Paused Pool

Category	Severity	Location	Status
Inconsistency	Informational	sources/core.move (farming): 279~281	Acknowledged

Description

When `pool.active == false`, the pool is expected to be inactive. However, due to the way rewards are calculated, rewards continue to accumulate during this period.

Recommendation

We want to confirm with the team whether this is the intended design.

Alleviation

[Haedal, 10/10/2025]:

As expected, the original intention of the active design is to control user access, not to limit rewards

HH&-05 Discussion On Potential Incorrect Slippage Application For Repay Amount

Category	Severity	Location	Status
Design Issue	Informational	sources/pool.move (haevault): 1274~1281	Resolved

Description

The flash loan repayment applies the configured swap slippage in the wrong direction. Instead of increasing the repayment to account for slippage and execution risk ("plus slippage" per the comment), the current code reduces the required repayment by multiplying by `(1 - slippage)`. This allows underpayment relative to the oracle fair price.

```
/// Executes a flash loan operation, allowing borrowing of one asset type against another with oracle price verification.
/// The loan must be repaid in the other asset type based on the current oracle price plus slippage.
...
    let repay_amount = full_math_u64::mul_div_ceil(
        repay_amount,
        (SLIPPAGE_DENOMINATOR - swap_slippage),
        SLIPPAGE_DENOMINATOR,
    );
```

The test code doesn't consider this case. It repays exactly the certificate amount without asserting that slippage increases/decreases the repayment.

Recommendation

Since this operation can only be called by a privileged operator, we recommend the team double-check the intended design.

Alleviation

[Haedal, 10/10/2025]: This behavior is intentional.

The design allows the repayment amount to be slightly **lower** than the value calculated from oracle price, applying slippage as a **discount**, not an addition.

The parameter `swap_slippage` represents a tolerance for reduced repayment, not extra cost.

The comment 'plus slippage' will be corrected to avoid confusion.

HH&-06 Tick Spacing “Round-To-Zero” May Break Containment Of The Original Range

Category	Severity	Location	Status
Design Issue	Informational	sources/clmm_vault.move (haevault): 735~743	Acknowledged

Description

When aligning both bounds with the same “round to zero” helper, the aligned range is not a superset of the raw range. The lower bound can be raised (when negative), which may violating the expected containment property:

- Expected: aligned_lower ≤ raw_lower and aligned_upper ≥ raw_upper.
- Actual: aligned_lower can be greater than raw_lower.

Suppose:

- tick_spacing = 10
- current_tick = -101
- lower_offset = 7
- upper_offset = 9

Raw Range: [-101 - 7, -101 + 9] = [-108, -92]

Aligned Range: [-108 + |-108| % 10 = 8, -92 + |-92| % 10 = 2] = [-100, -90]

Aligned range [-100, -90] does not contain the raw range [-108, -92] because -100 > -108 (lower bound moved up).

Additionally, in some extreme cases, ranges that are close to 0 can collapse such that `tick_lower == tick_upper`.

- tick_spacing = 10
- current_tick = -3
- lower_offset = 5
- upper_offset = 2
- raw range: [-8, -1]
- aligned range: [0, 0]

Recommendation

Perhaps, the team can consider using floor-to-spacing for the lower bound and using ceil-to-spacing for the upper bound.

Alleviation

[Haedal, 10/10/2025]: Thank you for the observation.

The noted behavior (where the aligned range may not fully contain the raw range) is **acceptable** within our design.

This alignment logic is consistent with the keeper's implementation, and maintaining identical rounding behavior across both components is important for consistency.

Therefore, we will **keep the current rounding logic unchanged**.

HH&-07 | Discussion On Negative Price

Category	Severity	Location	Status
Volatile Code	Informational	sources/pyth_oracle.move (haevault): 591~592	Resolved

Description

The function `pyth_price_from_oracle_info()` calls `get_is_negative()` to check whether the current price is negative. However, for crypto spot pairs such as SUI/USDC or USDC/USD, prices are always expected to be non-negative. A negative price in this context should be treated as invalid data and the function should abort, rather than convert it to its absolute value. Simply taking the absolute value could mask anomalies and introduce unintended risks.

Recommendation

We would like to confirm this observation with the team to understand whether this scenario has been considered, and if this behavior is intentional and aligned with the intended business logic.

Alleviation

[Haedal, 10/13/2025]:

The team heeded the advice and resolved the issue by filtering negative prices in commit e867cc394e36d27165c39b305107bd7537c2e254.

HH&-08 | `after_set_boost` Drops Pending Rewards

Category	Severity	Location	Status
Volatile Code	Informational	sources/core.move (farming): 769	Acknowledged

Description

`core::after_set_boost` recalculates share and overwrites `deposit.debts = debts(...)` without first converting pending rewards to credits.

And, the entry `operate::after_set_boost` exposes this to any `Deposit` without verifying an actual boost change.

If the deposit is not settled by the operator/robot/user before the `after_set_boost`, the pending rewards will be ignored.

Recommendation

We would like to check with the team if the scenario has been considered.

Alleviation

[Haedal, 10/10/2025]:

This situation has been taken into account. The original intention of after_set_boost is to overwrite. In extreme cases, unfinished rewards will also be ignored.

HH&-09 | Discussion On `remove_oracle_info()`

Category	Severity	Location	Status
Volatile Code	Informational	sources/pyth_oracle.move (haevault): 314~315	Acknowledged

Description

The `remove_oracle_info()` function allows the oracle manager role to remove oracle data without any restrictions. Depending on what is removed, the impact varies:

- If the removed asset is the pool's `quote_type`: Any operation that requires `calculate_tvl_base_on_quote` (e.g., `calculate_aum`, `deposit_for`) will abort because the latest quote price within the transaction cannot be retrieved.
- If the removed asset is either `A` or `B`:
  - `calculate_aum` may still execute, but the removed token will be ignored, reducing the reported AUM.
  - Subsequent operations such as `deposit_for`, `add_liquidity`, `rebalance`, and `flash_loan` will abort due to missing price data.
- If the removed asset is neither `A` nor `B` (e.g., a reward token in the buffer):
  - `calculate_aum` will ignore the removed asset, again reducing the reported AUM.
  - `deposit_for` can still proceed (since it depends only on the prices of `A` and `B`). However, this creates a governance/strategy risk: with a smaller AUM denominator, new LPs receive disproportionately more LP tokens, effectively diluting the value of existing LPs.

Recommendation

We would like to highlight this observation to remind the team of the potential risks. It is recommended to refactor the `remove_oracle_info()` function to impose restrictions on oracle data removal, ensuring that critical oracle entries cannot be arbitrarily deleted.

Alleviation

[Haedal, 10/10/2025]:

Implementing strict checks on `remove_oracle_info()` is not straightforward. However, this operation is **strictly protected by a multisig**, and cannot be executed arbitrarily.

Given this access control, we consider the current design acceptable and do not plan to make changes.

[CertiK, 10/10/2025]:

As mentioned by the team, the `remove_oracle_info` is restricted to the Oracle manager; however, we would encourage the team double double-check the potential effect before any invocations.

HH&-10 | Discussion On External Funding For `update_price_fee`

Category	Severity	Location	Status
Logical Issue	Informational	sources/pyth_oracle.move (haevault): 204~205	Resolved

Description

The `deposit_fee()` function is public and allows anyone to fund the `pyth_oracle.update_price_fee` balance. This balance is consumed when paying the update fee during calls to `update_price()` via:

```
let fee = pyth_oracle.split_fee(pyth_state.get_base_update_fee()).into_coin(ctx);
```

If no one funds the `pyth_oracle.update_price_fee`, the balance will eventually be insufficient. In this case, `split_fee()` will fail the assertion check, causing `update_price()` to abort.

Given the enforced call sequence within a PTB (`update_price()` -> `calculate_aum()` -> `deposit_for()`), any failure in `update_price()` prevents the subsequent functions from executing, leading to a potential denial of service (DoS).

Recommendation

We recommend the team closely monitor the `pyth_oracle.update_price_fee` balance and ensure that sufficient SUI is always injected to cover update costs. Consider introducing safeguards (e.g., automatic fee replenishment, restricted funding logic, or failover mechanisms) to reduce reliance on manual intervention and mitigate the risk of DoS.

Alleviation

[Haedal, 10/10/2025]:

We acknowledge that maintaining a sufficient `pyth_oracle.update_price_fee` balance is important to prevent failures.

To mitigate this risk, we plan to implement a **monitoring bot** that tracks the balance and automatically alerts or replenishes SUI as needed.

This approach will reduce reliance on manual intervention and help prevent potential DoS issues.

HH&-11 Rebalance Lacks Oracle/Pool Price Deviation Cap Used By Add-Liquidity

Category	Severity	Location	Status
Volatile Code	Informational	sources/pool.move (haevault): 1672	Acknowledged

Description

Add-liquidity blocks actions when the pool price deviates too far from the oracle price (measured in bps). Rebalance does not enforce this cap, yet derives target ticks directly from the oracle price.

If the oracle price is stale or manipulated relative to the CLMM price, the vault can migrate the position to mispriced ticks during rebalance. Add-liquidity would be blocked in the same condition.

Recommendation

We would like to check with the team if the scenario has been considered.

Alleviation

[Haedal, 10/10/2025]:

Our protocol operates under the **assumption that the oracle price is trustworthy**.

The rebalance logic intentionally uses the oracle price as the reference for determining new position ranges.

If the oracle price becomes unreliable, then the pool price itself would also be untrustworthy, so additional checks at this level would not be meaningful.

[CertiK, 10/10/2025]:

We understand that Oracle is considered trustworthy in the current design, however, it is recommended that the team keep monitoring the potential unexpected price closely, also consider multiple Oracle modes in the future to avoid a single point of failure.

HH&-12 | Discussion On Borrowing Scope In `flash_loan()`

Category	Severity	Location	Status
Volatile Code	Informational	sources/pool.move (haevault): 1249~1250	Acknowledged

Description

The `flash_loan()` function allows borrowing one asset type against another with oracle price verification. Based on the function logic:

- **Borrowed asset (`CoinTypeA`)**: Not limited to the pool's A/B assets. As long as the asset is included in `buffer_assets` and has a valid price update within the same transaction, such as the reward obtained from the CLMM pool, it can be borrowed.
- **Repayment asset (`CoinTypeB`)**: Must be one of the pool's A or B assets.

This design effectively broadens the scope of assets that can be borrowed beyond just the pool's A/B pair.

Recommendation

We recommend discussing this finding with the team to confirm whether this extended borrowing capability is intentional. If unintended, restrictions should be considered to limit the borrowable assets to reduce potential risks.

Alleviation

[Haedal, 10/10/2025]:

The flash loan operation is **restricted to privileged keeper operators**, so it cannot be executed arbitrarily.

The design **intentionally allows borrowing of assets in the buffer**, giving the keeper flexibility to manage and allocate assets as needed.

This behavior is by design and aligns with the intended business logic.

HH&-13 | Inconsistent On Assets Between Deposit And Withdraw

Category	Severity	Location	Status
Volatile Code	Informational	sources/pool.move (haevault): 811, 942, 1121	Acknowledged

Description

The `last_aum` only sums assets that have Oracle coverage in `calculate_aum` and other assets in the buffer are ignored. During LP minting uses `user_tvl` / `last_aum`, so the denominator is understated when unpriced assets exist, minting excess LP.

```
        if (!pyth_oracle.contain_oracle_info(ty) || amount == 0) {
            idx = idx + 1;
            continue
        };
```

However, withdrawals distribute all buffer assets (including unpriced ones), letting depositors later claim value not counted when minting.

Recommendation

We would like to check with the team if the scenario has been considered.

Alleviation

[Haedal, 10/10/2025]:

This behavior is **by design**. Assets without oracle prices are intentionally **not included in AUM calculations**, as the system cannot properly value them.

During pool creation, we ensure that all assets included are **mainstream tokens with available oracle prices**, so unpriced assets should not exist in practice.

HH&-19 | Inconsistency Between Code And Comment

Category	Severity	Location	Status
Volatile Code	Informational	sources/config.move (haevault): 694~695; sources/pool.move (haevault): 543, 1023~1024	Resolved

Description

Package Version Validation

The function returns the protocol fee rate without verifying the current package version, which contradicts the comment indicating that the function should abort if the package version is deprecated:

```
/// # Aborts
/// * If package version is deprecated
```

This inconsistency between the code's behavior and its comment can lead to misunderstandings about the intended functionality and may cause issues if the function is expected to enforce version checks.

Mismatched Slippage Definition

The comment stated that old_slippage/new_slippage are `0-255`, but implementation uses basis points with a 10,000 denominator (`0-10,000`).

```
/// Event emitted when a swap slippage configuration is set.
///
/// # Fields
/// * `type_name` - The type name of the coin being configured
/// * `old_slippage` - The old slippage value being set (0-255)
/// * `new_slippage` - The new slippage value being set (0-255)
public struct SetSwapSlippageEvent has copy, drop {
    type_name: TypeName,
    old_slippage: u64,
    new_slippage: u64,
}
```

Mismatched Fee Cap

The comment stated that the cap is 20%, but MAX_PROTOCOL_FEE_RATE is `5,000/10,000 = 50%`.

```
/// Max protocol fee(5000/10000 = 50%)
const MAX_PROTOCOL_FEE_RATE: u64 = 5000;
...
/// The fee rate must be less than or equal to MAX_PROTOCOL_FEE_RATE (20%).
```

Mismatch Initialized Fee

The comment stated that the default protocol fee is 0%, but the initialized value is 1,600 (16%).

```
/// * Creates and transfers an AdminCap to the deployer
/// * Creates a GlobalConfig object with:
///   - Default slippage settings (100/2000 = 5%)
///   - 0% protocol fee rate
...
fun init(ctx: &mut TxContext) {
    ...
    let mut cfg = GlobalConfig {
        id: object::new(ctx),
        swap_slippages: vec_map::empty(),
        protocol_fee_rate: 1600,
        ...
    };
    ...
```

Missing Aum Calculation

The function's comment states that it will abort if the AUM calculation has not been performed:

```
/// # Aborts
/// * If the pool is paused
/// * If AUM calculation is not done
/// * If package version check fails
```

However, the function does not actually implement any check or logic to verify whether the AUM calculation has been completed.

```
    public fun withdraw_buffer_reward<T, CoinTypeC>(
    ...
```

Return Value Mismatch

The `get_liquidity_range` function stated to return 5 values in the comment, which is not consistent with the design:

```
// Returns the min tick, max tick, lower offset, upper offset, and rebalance
threshold.
```

Recommendation

It is recommended to update the function so that its logic aligns with the comment, or revise the comment to accurately reflect the current behavior of the code.

Alleviation

[Haedal, 10/14/2025]:

The team heeded the advice and resolved the issue by updating the comments in the vault in commit 25c8b59078c8d6f6d424bd58a725f25cb413c74f.

HH&-20 | Missing Check On `lower_offset` And `upper_offset`

Category	Severity	Location	Status
Volatile Code	Informational	sources/clmm_vault.move (haevault): 123~125	Resolved

Description

The `create_pool()` function uses `lower_offset` and `upper_offset` to calculate the lower and upper ticks when opening a position. However, the function lacks input validation for these parameters. If inappropriate values are provided, it may abort due to invalid calculations.

Recommendation

It is recommended to add input validation to ensure that the provided values fall within a valid range.

Alleviation

[Haedal, 10/15/2025]: The team heeded the advice and resolved the issue in commit 25c8b59078c8d6f6d424bd58a725f25cb413c74f.

HH&-21 Unintended Abort Caused By Missing Oracle Info Validation In TVL Calculation

Category	Severity	Location	Status
Volatile Code	Informational	sources/pool.move (haevault): 966~969	Resolved

Description

In the `pool.move` module, the `calculate_tvl_base_on_quote` function assumes all assets in the input `VecMap` have corresponding Pyth oracle information. However, not all call sites (e.g., in `deposit` / `deposit_for`) perform explicit checks via `pyth_oracle.contain_oracle_info(ty)`. This can lead to runtime aborts (`error::price_not_exists`) if an asset lacks oracle support, particularly for `CoinTypeA/B` in user deposits.

Additionally, `create_pool` does not validate oracle info for `CoinTypeA/B`, allowing invalid pools to be created.

```
fun calculate_tvl_base_on_quote(
    ...
    assets: &VecMap<TypeName, u64>,
    ...
): u128 {
    ...
    while (i < vec_map::size(assets)) {
        ...
        let base_price = pyth_oracle.get_price_by_type(*ty, clk);
        ...
    };
    total_aum
}
```

```
public fun deposit_for<CoinTypeA, CoinTypeB, T>(
    ...
): Coin<T> {
    ...
    user_assets.insert(type_name::get<CoinTypeA>(), amount_a);
    user_assets.insert(type_name::get<CoinTypeB>(), amount_b);

    let user_tvl = calculate_tvl_base_on_quote(
        pyth_oracle,
        &user_assets,
        pool.quote_type,
        clk
    );
    ...
```

Recommendation

Recommend adding explicit checks for `CoinTypeA/B` in the `create_pool`.

Alleviation

[Haedal, 10/13/2025]: The team heeded the advice and resolved the issue by adding a check in `create_pool` in commit b4604d914137dc9bf76e659bc7d465272e0f1e73.

HH&-22 | DoS Risk Of Using VecMap

Category	Severity	Location	Status
Denial of Service	Informational	sources/core.move (farming): 36, 46, 74~75	Acknowledged

Description

We have observed that some fields in `Setting`, `Pool`, and `Deposit` make use of `VecMap`:

`core.move`

```
 33  public struct Setting has key {
 34      id: UID,
 35      version: u64,
 36  @>  pools: VecMap<TypeName, address>, // Deposit coin type -> Pool address
 37  @>  boosts: VecMap<address, u64>, // User address -> Boost
 38  }
 39
 40  public struct Pool<phantom DepositCoin> has key, store {
 41      id: UID,
 42      active: bool,
 43      balance: Balance<DepositCoin>,
 44      total_share: u128,
 45      total_boost_share: u128,
 46  @>  reward_configs: VecMap<TypeName, RewardConfig>,
// Reward coins this pool support
 47      users: Table<address, address>, // User -> Deposit to this pool
 48  }
```

`core.move`

```
 68  public struct Deposit<phantom DepositCoin> has key {
 69      id: UID,
 70      pool: address,
 71      amount: u64,
 72      share: u128,
 73      boost_share: u128,
 74  @>  debts: VecMap<TypeName, u64>,
// Reward coin type -> Debt, debt is the reward that has been already harvested
 75  @>  credits: VecMap<TypeName, u64>,
// Reward coin type -> Credit, credit is the reward that has not been harvested
 76      owner: address,
 77  }
```

Since operations on `VecMap` involve traversing the entire `VecMap`, it is more prone to DoS compared to structures like `Table`. For example, in the `settle()` function, each `RewardConfig` is iterated over, and for each `RewardConfig`, debt and credit operations are performed, resulting in `O(n^2)` time complexity, where `n` is the number of `RewardConfig`

entries in the `Pool`. Although `VecMap` has a limit of 1000 entries, it could still exceed Sui's single-transaction limit and cause DoS.

Recommendation

We would like to highlight this DoS risk to the team, recommend exercising caution when using this data structure, and suggest strengthening monitoring for the affected parts to prevent potential DoS occurrences.

Alleviation

[Haedal, 10/10/2025]:

We will strengthen the monitoring and operation sensitivity of VecMap size. The current maximum case of the protocol will not appear in docs. But we will still record and monitor it in the log.

HH&-23 | Missing Check On `usd_price_age`

Category	Severity	Location	Status
Volatile Code	Informational	sources/pyth_oracle.move (haevault): 275~276, 352~353	Resolved

Description

The `add_oracle_info()` function allows the oracle manager to update oracle information. This value is used to determine price freshness: a price is considered fresh and valid only if the difference between the current time and the price update time is less than this threshold.

However, there is no input validation on the `usd_price_age` parameter. For example, this value should always be greater than zero. If it is set to zero or another unreasonable value, prices would never be recognized as fresh, preventing updates from being accepted as expected.

Recommendation

It is recommended to add input validation on `usd_price_age` within the `update_price_age()` and `add_oracle_info()` functions.

Alleviation

[Haedal, 10/13/2025]: The team heeded the advice and resolved the issue by adding a zero check for `usd_price_age` in commit b4604d914137dc9bf76e659bc7d465272e0f1e73

HH&-24 | Missing Pool-Deposit Consistency Check In Remaining

Category	Severity	Location	Status
Coding Issue	Informational	sources/core.move (farming): 769, 933	Acknowledged

Description

Calling remaining with a deposit that belongs to a different pool returns meaningless numbers (computed against the wrong reward_config set). This won't corrupt the state, but can mislead callers and tests.

A similar issue is also valid for `after_set_boost` function.

Recommendation

Recommend adding pool validation to the remaining function.

Alleviation

[Haedal, 10/11/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

HH&-27 | Missing Validation On Zero `rebalance_threshold`

Category	Severity	Location	Status
Volatile Code	Informational	sources/clmm_vault.move (haevault): 181	Acknowledged

Description

The pool's rebalance reads the `threshold` and calls `check_need_rebalance`.

```
    let (need_rebalance, tick_lower, tick_upper) = check_need_rebalance(
        pool,
        clmm_pool.tick_spacing(),
        sqrt_price,
        rebalance_threshold
    );
    assert!(need_rebalance, error::pool_not_need_rebalance());
```

`check_need_rebalance` compares the distance (in ticks) between the expected new range and the current position's range to the threshold.

When setting `rebalance_threshold` to 0 makes the condition trivially true: any non-negative delta satisfies the check. When deltas are both 0 (no price movement and expected range equals the current range), `0 >= 0` still returns true, so `check_need_rebalance` signals rebalancing is needed.

Recommendation

Recommend adding 0 validation `rebalance_threshold` if the scenario is not intended.

Alleviation

[Haedal, 10/10/2025]:

Based on our practical experience, we no longer perform the 'need rebalance' check within the contract.

Rebalance decisions are fully handled by the keeper bot according to more advanced vault management strategies.

As a result, currently the rebalance_threshold is all set to zero. this check is no longer needed

HH&-29 | Outdated Dependency

Category	Severity	Location	Status
Volatile Code	Informational	Move.toml (haevault): 8	Resolved

Description

The given dependency `integer-mate` is outdated.

Recommendation

Recommend upgrading to a newer version to catch up latest update and fix.

Alleviation

[Haedal, 10/15/2025]: The team heeded the advice and resolved the issue in commit 25c8b59078c8d6f6d424bd58a725f25cb413c74f.

HHF-02 Discussion On `update_package_version()` And `emergency_pause()`

Category	Severity	Location	Status
Logical Issue	Informational	sources/config.move (haevault-1014): 391~392, 658~659	Resolved

Description

The `emergency_pause()` function allows entities with the `ACL_EMERGENCY_PAUSE` role to set `config.package_version` to `EMERGENCY_PAUSE_VERSION` . However, the `update_package_version()` function also permits the admin to update the package version, as long as the new version is greater than the existing one:

```
assert!(version > old_version, error::wrong_package_version());
```

This means the admin could also update the version to `EMERGENCY_PAUSE_VERSION` , resulting in overlapping functionality between the two functions and potential ambiguity in their intended roles.

Recommendation

We would like to discuss this finding with the team to confirm whether this behavior is consistent with the intended business logic and access control design.

Alleviation

[Haedal, 10/21/2025]:

Thanks for pointing this out. The current design is intentional:

- `emergency_pause()` can be executed by multiple addresses that have been granted the `ACL_EMERGENCY_PAUSE` role, allowing them to quickly pause the system in emergency situations.
- `update_package_version()` can only be called by the `AdminCap` owner, which ensures that only the main admin can perform version upgrades.

Although both functions can set `package_version` to `EMERGENCY_PAUSE_VERSION` , their access control scopes and intended purposes are different . one is for emergency response (multi-party), and the other is for administrative version management (single owner). We believe this behavior aligns with the intended business logic.

HHF-03 | Discussion On Inconsistent Doc

Category	Severity	Location	Status
Volatile Code	Informational	sources/clmm_vault.move (haevault-1020): 658	Resolved

Description

The codebase consistently treats CLMM sqrt prices as Q64 (scaled by 2^64), but a comment refers to “sqrt price X96.” The math and call sites align with Q64; the X96 mention appears to be a doc inconsistency.

Recommendation

We would like to check with the team if compatible to X96 is needed.

Alleviation

[Haedal, 10/21/2025]: The team heeded the advice and resolved the issue in commit 06aa2493c67d864b025f8832007928e6502f7fb8.

HHF-04 | Decimal Parity Can Break Round-Trip When P+O Is Odd

Category	Severity	Location	Status
Volatile Code	Informational	sources/utils.move (haevault-1013): 136~144	Resolved

Description

The conversion pair is only strictly invertible when `PRICE_MULTIPER_DECIMAL (P) + oracle_price_multiplier_decimal (O)` is even. If P+O is odd, the forward step divides by `10^floor((P+O)/2)` while the math expects `10^((P+O)/2)`, effectively dropping a sqrt(10) factor. After squaring on the way back, this manifests as an approximate 10× drift, beyond normal rounding.

Proof of Concept

The following test is trying to describe the scenario that odd `PRICE_MULTIPER_DECIMAL + oracle_price_multiplier_decimal` would cause ~10x drift

```
#[test]
fun test_roundtrip_parity_cases() {
    // Odd parity: P=10, O=9 -> deterministic ~10x drift
    // Choose price so that sqrt(price * 10^P) is an integer to avoid rounding noise
    let o_odd: u8 = 9;
    let price_in_odd = 10000000000; // 10^10
    let s_odd = price_to_sqrt_price(price_in_odd, o_odd);
    let price_out_odd = sqrt_price_to_price(s_odd, 9, 9, o_odd);
    assert!(price_out_odd == ((price_in_odd as u128) * 10), 1);
}
```

Recommendation

We would like to check with the team if the scenario has been considered, otherwise, validations would needed to avoid odd P+O.

Alleviation

[Haedal, 10/21/2025]: In our use scenario, both PRICE_MULTIPER_DECIMAL and oracle_price_multiplier_decimal is 10, so P + Q is even always.

OPTIMIZATIONS | HAEDAL - HAEVAULT & FARMING

ID	Title	Category	Severity	Status
HH&-01	Missing Event Emission	Volatile Code	Optimization	Acknowledged

HH&-01 | Missing Event Emission

Category	Severity	Location	Status
Volatile Code	Optimization	sources/pool.move (haevault): 1621~1622	Acknowledged

Description

The `deposit_for()` function allows a user to deposit coin assets into the CLMM vault and receive minted LP coins in return. Internally, the pool invokes the `add_liquidity()` function to add the newly deposited assets into the target pool.

However, if the price deviation between the oracle price and the current pool price exceeds the predefined threshold (`config.get_max_price_deviation_bps()`), the function simply returns without any event emission.

This silent return behavior may lead to user confusion and reduces the transparency and auditability of on-chain operations.

Recommendation

It is recommended to emit an event to notify users that their assets were not added to the target pool due to excessive price deviation.

Alleviation

[Haedal, 10/10/2025]:

Users do not need to be concerned about whether the deposited assets are immediately added to the position.

The **keeper bot** continuously monitors and handles liquidity management, ensuring assets are added to the pool as needed.

Whether `add_liquidity` is executed during `deposit_for()` does **not affect the user's LP token amount**, and performing it at deposit time is **not mandatory** in our design.

APPENDIX | HAEDAL - HAEVAULT & FARMING

Audit Scope

haedallsd/haevault
sources/clmm_vault.move
sources/pool.move
sources/pyth_oracle.move
sources/config.move
sources/utils.move
sources/acl.move
sources/balance_bag.move
sources/error.move
sources/utils.move
sources/acl.move
sources/balance_bag.move
sources/clmm_vault.move
sources/config.move
sources/error.move
sources/pool.move
sources/pyth_oracle.move

haedallsd/farming-contract
sources/core.move
sources/operate.move

haedallsd/farming-contract
sources/breaker.move
sources/interface.move
sources/manage.move
sources/minorsign.move
sources/robot.move
sources/breaker.move
sources/core.move
sources/interface.move
sources/manage.move
sources/minorsign.move
sources/operate.move
sources/robot.move

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.

Categories	Description
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.