

Cetus

Audit Report

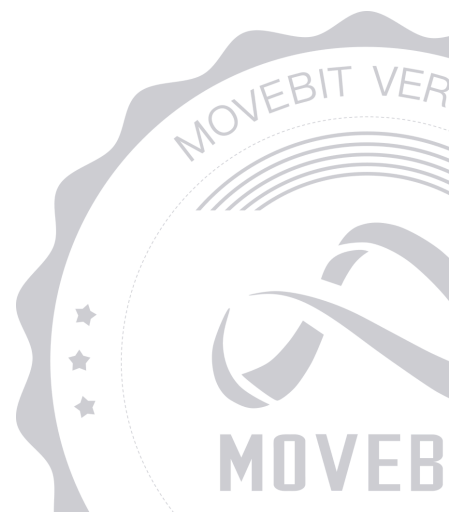


contact@bitslab.xyz



https://twitter.com/movebit_

Thu Jul 23 2026



Cetus Audit Report

1 Executive Summary

1.1 Project Information

Description	This audit covers the changes introduced between commit <code>c4e349bfecfe3852c1a6ab5091c39eac228c5482</code> and commit <code>e8c7f535830032718993da72a71272fc7900d854</code> , corresponding to PR #56 in the Cetus CLMM Sui repository. The review focuses on changes to the vaults package, including the Pyth Pro Compatible oracle upgrade and backward-compatible oracle configuration handling, Vault wrapped-position storage migration to dynamic fields, emergency pause and recovery controls
Type	Dex
Auditors	poetyellow,N0n3,Nebula
Timeline	Tue Jul 21 2026 - Thu Jul 23 2026
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/CetusProtocol/cetus-clmm-sui/ https://github.com/CetusProtocol/cetus-clmm-sui/pull/56
Commits	https://github.com/CetusProtocol/cetus-clmm-sui/:e8c7f535830032718993da72a71272fc7900d854

[a92910d1adcee27e85f84a310017a40571f5ee78](#)

[https://github.com/CetusProtocol/cetus-clmm-sui/pull/56:](https://github.com/CetusProtocol/cetus-clmm-sui/pull/56:temp)
[temp](#)

1.2 Files in Scope

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

ID	File	SHA-1 Hash
ORE	vaults/sources/oracle_registry.move	7cffb3f4d891b47043ab53ff183473f867b9d888
VAU	vaults/sources/vaults.move	8c0f24dc486653e12adac9d1bf34b2a1d641a446

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	3	0
Critical	0	0	0
Major	1	1	0
Medium	0	0	0
Minor	0	0	0
Informational	2	2	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 [Cetus](#) to identify any potential issues and vulnerabilities in the source code of the [Cetus](#) 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 3 issues of varying severity, listed below.

ID	Title	Severity	Status
ORE-4	Positive Pyth Prices Can Normalize to Zero Before Storage	Informational	Fixed
VAU-1	Unbounded Emergency Recovery Can Re-enable Deprecated Package Entrypoints	Major	Fixed
VAU-3	Broad Pause-State Check Accepts Unexpected Version Sentinels	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the [Cetus](#) Smart Contract :
Here is a participant process description for the `vaults` and `oracle_registry` modules.
In this system, a Vault holds a concentrated-liquidity position and mints a vault LP token representing shares in that position. `VaultsManager` maintains ACL-controlled protocol configuration, while `OracleRegistry` stores Pyth price-feed configurations and same-transaction price updates used by Vault operations.

AdminCap Holder

- The `AdminCap` holder can manage ACL membership and permissions through `set_roles()` , `add_role()` , `remove_role()` , and `remove_member()` .
- The `AdminCap` holder can synchronize the manager's package version through `update_package_version()` .
- AdminCap is created during module initialization and transferred to the deployer.

Pool Manager

- A pool-manager-role holder can create Vaults through `create_vault_v2()` .
- A pool-manager-role holder can configure the internal CLMM-pool oracle mapping through `add_oracle_pool()` , `remove_oracle_pool()` , and `update_slippage()` .
- A pool-manager-role holder can update Vault parameters through `update_protocol_fee_rate()` , `update_max_quota()` , and `update_rebalance_threshold()` .
- A pool-manager-role holder can pause or resume an individual Vault through `pause()` and `unpause()` .
- A pool-manager-role holder can begin or finalize a Vault exit through `remove_all_liquidity()` and `finalize_exit()` .

- A pool-manager-role holder can migrate a pre-upgrade Vault's wrapped position from the legacy `positions` vector into the internal `PositionKey` dynamic field through `migrate_position_to_df()` . The migration is one-time and requires a running or rebalancing Vault with no outstanding flash loan.

Oracle Manager

- An oracle-manager-role holder can initialize the shared `OracleRegistry` through `init_oracle_registry()` .
- An oracle-manager-role holder can register Pyth Pro Compatible price-feed metadata through `add_coin_info_pyth2()` or `add_coin_info_use_currency_pyth2()` .
- An oracle-manager-role holder can remove legacy or Pyth Pro Compatible price-feed configuration through `remove_coin_info()` and `remove_coin_info_pyth2()` .
- An oracle-manager-role holder can change a feed's permitted price age and slippage parameters through `update_price_age()` and `update_coin_slippage()` .
- Legacy `add_coin_info()` , `add_coin_info_use_currency()` , and `update_price()` entrypoints are deprecated and abort; operators must use their `_pyth2` counterparts where applicable.

Price Updater / Keeper

- Any caller can submit a current Pyth Pro Compatible `PriceInfoObject` through `update_price_pyth2()` .
- The update is accepted only when the price object matches the registered feed and identifier, is within the configured age limit, has a positive non-zero price, and satisfies the confidence/volatility threshold.
- Consumers retrieve prices through `oracle_registry::get_price()` or `get_price_by_type()` . These functions require the price to have been updated within the same transaction.

Reinvest, Rebalance, and Operations Managers

- A reinvest-manager-role holder can reinvest accumulated Vault assets through `reinvest()` .
- A rebalance-manager-role holder can migrate liquidity and rebalance the Vault position through `migrate_liquidity()` and `rebalance()` .
- A reinvest- or rebalance-manager-role holder can perform operational actions, including `flash_loan()` , `flash_loan_v2()` , `repay_flash_loan()` , `recharge_harvest_asset()` , `collect_fee()` , `harvest()` , and `collect_rewarder()` .
- A protocol-fee-claim-role holder can withdraw accrued protocol fees through `claim_protocol_fee()` .

Emergency Pause Operator

- An emergency-pause-role holder can suspend version-gated Vault operations through `emergency_pause()` .
- The same role can restore a paused manager through `emergency_unpause()` . The restored version must be bounded by the pre-pause version and the current package version.
- During the special emergency-recovery state, authorized operations managers can use `governance_harvest()` and `governance_collect_rewarder()` for paused Vaults.

Vault LP Holder / User

- A user can provide the two underlying assets and receive Vault LP tokens through `deposit_v2()` or `deposit_with_slippage_v2()` .
- A user can withdraw their proportional Vault share through `remove()` or start an exit flow through `exit_vault()` , subject to the Vault's configured state.
- A user can use `get_position_amounts()` to determine the current underlying-token amount represented by a specified LP-token amount.
- The former public `borrow_wrapped_position()` accessor is retained only for upgrade compatibility and always aborts. The wrapped farming/CLMM position is now

accessible only to internal Vault logic through its `PositionKey` dynamic field.

4 Findings

ORE-4 Positive Pyth Prices Can Normalize to Zero Before Storage

Severity: Informational

Status: Fixed

Code Location:

`vaults/sources/oracle_registry.move#331`

Descriptions:

`pyth_price_from_coin_info()` verifies that the raw Pyth mantissa is positive and non-zero. It does not verify that the normalized protocol price remains non-zero after decimal downscaling.

`vaults/sources/oracle_registry.move`

```
let priceu64 = price_i64.get_magnitude_if_positive();
assert!(priceu64 != 0, EInvalidOraclePrice);
// ...
priceu64 / u64::pow(10, price_decimals - price_scale!())
```

When `price_decimals` exceeds `price_scale!()`, the integer division floors the result. Any positive `priceu64` below `10^(price_decimals - price_scale!())` becomes zero. `update_price()` can then persist that zero in the oracle cache, and `get_price()` can return it to downstream flows.

This breaks the invariant that a validated positive oracle input results in a positive protocol price. It can cause incorrect valuation or aborts in accounting paths that assume a cached price is strictly positive. The issue requires a feed that enters the downscaling branch with a positive mantissa smaller than the divisor.

Suggestion:

It is recommended to reject a normalized price of zero after downscaling. Add tests for a positive value that floors to zero, the boundary value that normalizes to one, and ordinary positive prices.

Resolution:

The issue is fixed in commit `3a7294b4eacf77494a4724494eeb65a2d282fb94`. Oracle price normalization now rejects any positive Pyth price that would downscale to zero during decimal normalization.

VAU-1 Unbounded Emergency Recovery Can Re-enable Deprecated Package Entrypoints

Severity: Major

Status: Fixed

Code Location:

`vaults/sources/vaults.move#667`

Descriptions:

`VaultsManager.package_version` is shared on-chain state used by every immutable vaults package. Each package evaluates this shared value against its own compiled `VERSION`.

`vaults/sources/vaults.move`

```
public fun checked_package_version(config: &VaultsManager) {
    assert!(VERSION >= config.package_version, EPackageVersionDeprecate);
}
```

This is a one-way lower-bound gate, not an exact current-package lock. If the shared value is restored to `15`, both the current v16 package and the historical v15 package pass their gates because `16 >= 15` and `15 >= 15`. The old package remains immutable and its public entrypoints remain on-chain, so a low shared version makes its old code callable again. The recovery flaw comes from incompatible pause metadata between old and new packages. The old immutable package's `emergency_pause()` only writes the shared pause sentinel. It does not store the pre-pause version and does not create `EMERGENCY_PAUSE_BEFORE_VERSION`.

`historical vaults/sources/vaults.move`

```
public fun emergency_pause(vaults_m: &mut VaultsManager, ctx: &TxContext) {
    check_emergency_pause_role(vaults_m, tx_context::sender(ctx));
    vaults_m.package_version = EMERGENCY_PAUSE_VERSION;
}
```

The new package stores `old_version` in the dynamic field when its own `emergency_pause()` is used. To recover a pause made by the old package, the new `emergency_unpause()` deliberately accepts a missing dynamic field. In that no-stash branch, it sets `before_version` to `0` and accepts every requested `version` in the range `0..=VERSION`.

vaults/sources/vaults.move

```
let before_version = if (...) {  
    dynamic_field::remove<vector<u8>, u64>(...)  
} else {  
    0  
};  
assert!(version >= before_version && version <= package_version(),  
EWrongPackageVersion);  
vaults_m.package_version = version;
```

The no-stash branch has no record of the version that was active before the old pause and imposes no replacement lower bound. An emergency-pause operator can therefore recover the shared manager to `15` or another historical version.

This creates a two-stage attack surface. The recovery call itself requires the emergency-pause role. However, after an operator mistakenly restores `package_version` to `15`, ordinary users can call historical v15 public entrypoints that have no such role check. A concrete v15 entrypoint is `borrow_wrapped_position()`. It returns `&WrappedPositionNFT` from the public `vault.positions` vector without an ACL, pause, or other access check.

historical v15 vaults/sources/vaults.move

```
public fun borrow_wrapped_position<T>(vault: &Vault<T>): &WrappedPositionNFT {  
    vector::borrow(&vault.positions, 0)  
}
```

The current package fixes this by moving the position to a private dynamic field and making its own `borrow_wrapped_position()` abort. That new implementation does not remove the

v15 immutable package. Once the shared version is restored to 15, the historical getter can again read the legacy vector of any vault that has not completed `migrate_position_to_df()`. An arbitrary user can pass the returned reference to `stable_farming::router::collect_fee()`. This public entrypoint accepts `&WrappedPositionNFT`, collects CLMM fees for the wrapped position, and transfers both resulting coins to `tx_context::sender(ctx)`. The same reference can also reach the public farming-reward and CLMM-reward collection paths. Therefore, after an erroneous privileged rollback, an unprivileged attacker can withdraw accrued fees or rewards from every eligible, unmigrated legacy vault.

stable_farming/sources/router.move

```
public entry fun collect_fee<CoinA, CoinB>(
  global_config: &GlobalConfig,
  clmm_global_config: &CLMMGlobalConfig,
  clmm_pool: &mut CLMMPool<CoinA, CoinB>,
  wrapped_position: &WrappedPositionNFT,
  ctx: &mut TxContext,
){
  let (balance_a, balance_b) = pool::collect_fee(..., wrapped_position, true);
  send_coin<CoinA>(coin::from_balance(balance_a, ctx), tx_context::sender(ctx));
  send_coin<CoinB>(coin::from_balance(balance_b, ctx), tx_context::sender(ctx));
}
```

The exposure is limited to vaults whose positions still remain in the legacy `positions` vector. For migrated vaults, the vector is empty and the historical getter aborts. The existing regression test reconstructs the old-package pause state but recovers only to `package_version()`. It does not test that a lower version is rejected in the same no-stash state.

Suggestion:

It is recommended to allow the no-stash recovery path to restore only `VERSION`, or to record an immutable recovery target and require an exact match. Add regression tests that reject low recovery versions and confirm that historical package entrypoints remain disabled.

Resolution:

The issue is fixed in commit `3a7294b4eacf77494a4724494eeb65a2d282fb94` . The legacy no-stash recovery branch in `emergency_unpause` now restores `package_version` to `16` , which forces the shared manager back onto the current package.

VAU-3 Broad Pause-State Check Accepts Unexpected Version Sentinels

Severity: Informational

Status: Fixed

Code Location:

`vaults/sources/vaults.move#667`

Descriptions:

`emergency_unpause()` decides whether the manager is paused by checking whether `package_version` is greater than `VERSION`.

`vaults/sources/vaults.move`

```
assert!(vaults_m.package_version > VERSION, EProtocolNotEmergencyPause);
```

The only pause value written by `emergency_pause()` is `EMERGENCY_PAUSE_VERSION`. The broad comparison accepts every unexpected version value above `VERSION` as an emergency-pause state and allows the recovery flow to continue. It does not bind recovery to the protocol's defined pause sentinel.

This weakens the version state machine and could treat a future sentinel, migration error, or other unexpected high version as a valid emergency pause. No public path was identified that directly creates such a state, so this issue is informational.

Suggestion:

It is recommended to require `vaults_m.package_version == EMERGENCY_PAUSE_VERSION` and add a regression test that rejects all other high version values.

Resolution:

This issue has been fixed in the commit `3a7294b4eacf77494a4724494eeb65a2d282fb94`.

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 pose any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

