Cetus Farming Smart Contract

Audit Report

contact@movebit.xyz

https://twitter.com/movebit_

Fri Jan 19 2024

Cetus Farming Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	A farming
Type	DeFi
Auditors	MoveBit
Timeline	Fri Dec 29 2023 - Fri Jan 05 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/CetusProtocol/cetus-clmm-sui
Commits	cab10270ca5567772b47bad65bc2db0953bcc55e 8f83fca210929aa86053e2338c00204450170c46 26a210c342f590739f9360d1195d62180c2e4b54 61e18eb5ed3ac7c9bc84811a472fa03725ecb6a8

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV4	sui/stable-farming/Move.toml	4271cf3660b793b951d089ac65c7c6ab623c79f8
ACL1	sui/stable-farming/sources/acl.move	1e9c71ce06bded01c75c8adafe01352d9c72239d
REW2	sui/stable-farming/sources/rewarder.move	1cd3ae49bbd4c51a1f46b9054b21417b7240dadd
ROU2	sui/stable-farming/sources/router.move	5a18b04aa0b79b2ff1772c5adff71eeb0cf379a3
CON2	sui/stable-farming/sources/config.move	cf8e633cb40d101ac3a3684848e696244c397a2f
POO2	sui/stable-farming/sources/pool.move	7645b5da7af569e78e751b67a3568366b56feb86

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	9	9	0
Informational	0	0	0
Minor	5	5	0
Medium	0	0	0
Major	4	4	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

ScaleBit 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
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

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) 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 Farming 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 9 issues of varying severity, listed below.

ID	Title	Severity	Status
CON-1	Security Vulnerability in `add_operator` Function of `config.move` due to Missing Contract Version Check	Major	Fixed
POO-1	`pending_reward` Is Not Compatible	Major	Fixed
REW-1	Variable Return Value in Public Function	Major	Fixed
POO1-1	Reward Distribution Can Be Refactored	Minor	Fixed
POO1-2	Unnecessary Parameter	Minor	Fixed
REW1-1	Precision Loss Results in Rewards being Left in the Contract and Unable to be Withdrawn	Major	Fixed
REW1-2	Incorrect Assert Location	Minor	Fixed
REW1-3	Unused Function	Minor	Fixed
REW1-4	Unused `Function` `borrow_mut_pool_share()` in the	Minor	Fixed

	Contract		

3 Participant Process

Here are the relevant actors with their respective abilities within the Cetus Farming Smart Contract :

Admin

- The Admin can set roles for the member through `set_roles()` .
- The Admin can add a role to the member through `add_role()` .
- The Admin can remove a role from the member through `remove_role()` .
- The Admin can add an operator for the contract through `add_operator()` .
- The Admin can set the package version through `set_package_version()` .
- The Admin can withdraw the reward from the vault through `emergent_withdraw<RewardCoin>()` .

Operator

- The Operator can create a `farming Pool` related to `clmmpool` through `create_pool<CoinA, CoinB>()` .
- The Operator can update the `Pool` effective `Tick` range through `update_effective_tick_range<CoinA, CoinB>()` .
- The Operator can add reward `RewardCoin` for the `clmm_pool` through `add_rewarder<RewarderCoin, CoinA, CoinB>()` .
- The Operator can update the allocated point of the `clmm_pool` through `update_pool_allocate_point<RewardCoin, CoinA, CoinB>()` .
- The Operator can create a `Rewarder` type is `RewardCoin` through `create_rewarder<RewardCoin>()` .
- The Operator can update `Rewarder` emission speed through `update_rewarder<RewardCoin>()` .

User

- The User can deposit the `clmm` position info pool to get the farming rewarder through `deposit<CoinA, CoinB>()` .
- The User can withdraw `clmm` position from the farming pool, before the rewarder should be harvested through `withdraw()` .

- The User can harvest the farming rewarder through `harvest<RewardCoin>()`.
- The User can add liquidity through `add_liquidity<CoinA, CoinB>()`.
- The User can add liquidity by using the fix coin through `add_liquidity_fix_coin<CoinA, CoinB>()`.
- The User can remove liquidity through `remove_liquidity<CoinA, CoinB>()`.
- The User can collect the fee of `clmm_pool` through `collect_fee<CoinA, CoinB>()`.
- The User can collect the `clmm` reward through `collect_clmm_reward<RewardCoin, CoinA, CoinB>()`.
- The User can close position through `close_position<CoinA, CoinB>()`.

4 Findings

CON-1 Security Vulnerability in `add_operator` Function of `config.move` due to Missing Contract Version Check

Severity: Major

Status: Fixed

Code Location:

stable-farming/sources/config.move#110

Descriptions:

The function `add_operator()` in the `config.move` lacks a check for the contract version. If the caller invokes methods from an old version of the contract, and if there happen to be vulnerabilities in the old version, this could potentially lead to serious security incidents.

Suggestion:

It is recommended to add a check for the contract version

Resolution:

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

POO-1 `pending_reward` Is Not Compatible

Severity: Major

Status: Fixed

Code Location:

sui/stable-farming/sources/pool.move#685-686,749-750

Descriptions:

Reward distribution is a crucial part of Cetus stable farming smart contract code. And indeed most of the functions in `pool.move` have a while loop to update rewards.
However, in both `add_liquidity_fix_coin` and `remove_liquidity`, the `pending_reward` is calculated as such :

```
let pending_reward = (accumulated_reward as u128) - pos_info.reward_debt;
```

while in other functions like `add_liquidity`, `harvest`, `withdraw` it is calculated this way:

```
let pending_reward = (accumulated_reward as u128) - pos_info.reward_debt + pos_info.reward;
```

Without the addition of `pos_info.reward`, `pending_reward` would be calculated wrong and cause major problems in reward distribution in `add_liquidity_fix_coin` and `remove_liquidity` functions.

Suggestion:

It is suggested to fix the calculation of `pending_reward` in both `add_liquidity_fix_coin` and `remove_liquidity` functions.

Resolution:

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

REW-1 Variable Return Value in Public Function

Severity: Major

Status: Fixed

Code Location:

sui/clmmpool/sources/rewarder.move#315;

sui/stable-farming/sources/pool.move#594

Descriptions:

The function `borrow_mut_pool_share` returns a mutable reference to a value, which refers to the key `pool` in `manager.pool_shares`. Consequently, the value corresponding to this key- `pool_share`, can be modified by anyone, leading to errors in the contract when calculating `accumulate_pool_reward`. And the function `borrow_mut_clmm_position` has the same issue.

Suggestion:

It is recommended to modify the visibility to `internal` or `friend`.

Resolution:

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

POO1-1 Reward Distribution Can Be Refactored

Severity: Minor

Status: Fixed

Code Location:

sui/integrate/sources/pool.move#605-619

Descriptions:

In `pool.move` several functions use a while loop to update and calculate the pool rewards. For example, the one in `add_liquidity_fix_coin` is the same as the one in `remove_liquidity`.

```
while (idx < vector::length(&pool.rewarders)) {
        let rewarder_coin = *vector::borrow(&pool.rewarders, idx);
        let pool_acc_per_share = rewarder::update_pool_share(
           rewarder_manager,
           rewarder_coin,
           object::id(clmm_pool),
           pool_share,
            clk
        );
        let pos_info = vec_map::get_mut(&mut wrapped_pos_info.rewards,
&rewarder_coin);
        let accumulated_reward = full_math_u128::full_mul(old_share,
pool_acc_per_share);
        let pending_reward = (accumulated_reward as u128) - pos_info.reward_debt;
        pos_info.reward = pos_info.reward + pending_reward;
        pos_info.reward_debt = (full_math_u128::full_mul(share, pool_acc_per_share) as
u128);
        idx = idx + 1;
    }
```

Suggestion:

It is suggested to refactor the code and write a helper function to calculate the rewards to improve the readability and reusability.

Resolution:

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

POO1-2 Unnecessary Parameter

Severity: Minor

Status: Fixed

Code Location:

stable-farming/sources/pool.move#905

Descriptions:

In the `collect_fee` function, the parameters `coin_a` and `coin_b` are redundant. The return value after calling the `clmm_pool::collect_fee` function is the balance type, and there is no need to call the `join` function.

Suggestion:

It is recommended to remove `coin_a` and `coin_b` Parameters.

Resolution:

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

REW1-1 Precision Loss Results in Rewards being Left in the Contract and Unable to be Withdrawn

Severity: Major

Status: Fixed

Code Location:

integrate/sources/rewarder.move#404-407

Descriptions:

In the `accumulate_pool_reward()` function, the protocol first calculates the accumulated rewards over the past time period. Then, based on the proportion of each reward pool, it allocates these rewards to each pool. Finally, when `calculating acc_per_share` , it uses the `pool_acc_reward/total_pool_share` .

```
let acc_reward = full_math_u128::full_mul(
 rewarder.emission_per_second,
 ((current_ts - last_reward_time) as u128)
 );
 let pool_acc_reward = (acc_reward * (pool_rewarder_info.allocate_po
int as u256)) / (rewarder.total_allocate_point as u256);
 pool_rewarder_info.reward_released = pool_rewarder_info.reward_rele
ased + (pool_acc_reward as u128) / REWARD_PRECISION;
 pool_rewarder_info.acc_per_share = pool_rewarder_info.acc_per_shar
e + ((pool_acc_reward as u128) / pool_share);
 pool_rewarder_info.acc_per_share
```

Note that there are two instances of precision loss, once when calculating rewards for each pool based on the proportion and another when calculating `acc_per_share` . This can result in residual rewards in the reward pool that cannot be withdrawn after all users have claimed their rewards.

Suggestion:

It is recommended to implement a method in the reward pool allowing a specific role to withdraw rewards. Additionally, if `deposit_rewarder()` is accidentally called to recharge

rewards, a withdrawal mechanism should be available.

Resolution:

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

REW1-2 Incorrect Assert Location

Severity: Minor

Status: Fixed

Code Location:

stable-farming/sources/rewarder.move#299

Descriptions:

Assert of the `reward_balance` and `amount` quantities are ineffective when placed after the `split` function, they should be placed before.

Suggestion:

It is recommended to assert before the `split` function.

Resolution:

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

REW1-3 Unused Function

Severity: Minor

Status: Fixed

Code Location:

stable-farming/sources/rewarder.move#317

Descriptions:

The function `borrow_mut_pool_share()` is not utilized within the contract. Redundant functions like this may result in higher gas consumption during deployment and can impact the overall readability of the contract.

Suggestion:

It is recommended to remove redundant functions.

Resolution:

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

REW1-4 Unused `Function borrow_mut_pool_share()` in the Contract

Severity: Minor

Status: Fixed

Code Location:

stable-farming/sources/rewarder.move#317

Descriptions:

The function `borrow_mut_pool_share()` is not utilized within the contract. Redundant functions like this may result in higher gas consumption during deployment and can impact the overall readability of the contract.

```
public(friend) fun borrow_mut_pool_share(manager: &mut RewarderManager, poo
l: ID): &mut u128 {
linked_table::borrow_mut(&mut manager.pool_shares, pool) }
```

Suggestion:

It is recommended to remove redundant functions.

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

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.