Interest Protocol

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Jul 09 2024

Interest Protocol Audit Report

1 Executive Summary

1.1 Project Information

Description	Interest Protocol CLAMM is A Decentralized Exchange
Type	CLAMM
Auditors	MoveBit
Timeline	Tue Jun 04 2024 - Tue Jul 09 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/interest-protocol/clamm
Commits	8bc50340b2a680e6f8fc78cb950d81e41927e641 b3cc49dd0e85ce02f538190b0421aac769176d50 b84fd0bcf5d291e1e1128ebbb03f539f9be94809 44d9d511c6c7328222fd66c73f43e72969021040

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	contracts/Move.toml	3cf2b353a767b616d74ac5c960ac3f12a4d3bf3e
EVE	contracts/sources/events.move	3c883212229c3c8f810785c390cb11b71cf4cd81
POO1	contracts/sources/pool.move	aba41f9c98e5b4a10648497937454a492b6f7ea4
PAD	contracts/sources/pool_admin.move	366e46d8e49565bb69ec2cc396e130a438e8b71e
CUR	contracts/sources/invariants/curves.move	2c9f618d4d4d31ad98493645d776b7b99e7dc4ea
SMA1	contracts/sources/invariants/stable_math.move	3ea81ec3316d10fbc79122ed061d52d4e2487478
VMA	contracts/sources/invariants/volatile_math.move	3c51b6d6654d1207495ee2159c490aaeb7b91c09
UTI1	contracts/sources/utils.move	1091cc9eb2063541e9af338fdbf403088b17d041
ERR	contracts/sources/errors.move	07bacea856f1ac6c5d1e288c932a2948ca4f40ca
SFE1	contracts/sources/stable_fees.move	2ca02bff8c0361ea643141cc8eaff0ae650032f8
STA	contracts/sources/stable.move	fce08ef6abb2437ad2771ffc32d4f29a26075335

VOL	contracts/sources/volatile.move	065ff06ad8fae15644b5d75f19aab5f08ce637e4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	7	1
Informational	0	0	0
Minor	1	1	0
Medium	3	3	0
Major	4	3	1
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

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 Interest Protocol to identify any potential issues and vulnerabilities in the source code of the Interest Protocol - CLAMM 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 8 issues of varying severity, listed below.

ID	Title	Severity	Status
STA-1	Missing Fees for Unbalanced Liquidity	Major	Fixed
STA-2	Dual Fee Structure for Swap Operations	Major	Fixed
STA-3	Users can Avoid the Fees during the Exchange	Major	Fixed
STA-4	Centralization Risk	Major	Acknowledged
STA-5	The First User cannot immediately Remove Liquidity after Adding it	Medium	Fixed
STA-6	Missing Emergency Pause Functionality	Medium	Fixed
VOL-1	Lack of Timelock for Critical Admin Operations	Medium	Fixed
VOL-2	The Commit Operation Lacks Parameter Validation	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Interest Protocol - CLAMM Smart Contract :

Admin

- The admin can call the `pause` function to suspend the protocol.
- The admin can call the `unpause` function to resume the protocol.
- The admin can invoke `ramp` to adjust the pool's amplification factor over time.
- The admin can invoke `stop_ramp` to halt the ramping process and set the amplification factor to its current value.
- The admin can invoke `commit_fee` to set the future fee and admin fee for the pool.
- The admin can invoke `update_fee` to apply the committed fees to the pool.
- The admin can invoke `take_fees` to withdraw accumulated fees from the pool.
- The admin can invoke the `claim_admin_fees` function to claim accumulated admin fees from the pool.
- The admin can invoke the `commit_parameters` function to set future pool parameters with a delay.
- The admin can invoke the `update_parameters` function to apply the committed future parameters to the pool.

User

- The user can call the `new_2_pool` function to create a new 2-coin pool for the protocol.
- The user can call the `new_3_pool` function to create a new 3-coin pool for the protocol.
- The user can call the `new_4_pool` function to create a new 4-coin pool for the protocol.
- The user can call the `new_5_pool` function to create a new 5-coin pool for the protocol.
- The user can call the `swap` function to swap one type of coin for another within the protocol.

- The user can call the `add_liquidity_2_pool` function to add liquidity to a 2-coin pool within the protocol.
- The user can call the `add_liquidity_3_pool` function to add liquidity to a 3-coin pool within the protocol.
- The user can call the `add_liquidity_4_pool` function to add liquidity to a 4-coin pool within the protocol.
- The user can call the `add_liquidity_5_pool` function to add liquidity to a 5-coin pool within the protocol.
- The user can call the `donate` function to donate coins to the protocol.
- The user can call the `remove_liquidity_2_pool` function to remove liquidity from a 2-coin pool within the protocol.
- The user can call the `remove_liquidity_3_pool` function to remove liquidity from a 3-coin pool within the protocol.
- The user can call the `remove_liquidity_4_pool` function to remove liquidity from a 4-coin pool within the protocol.
- The user can call the `remove_liquidity_5_pool` function to remove liquidity from a 5-coin pool within the protocol.
- The user can call the `remove_liquidity_one_coin` function to remove liquidity from the pool in the form of a single coin within the protocol.
- The user can call the `new_2_pool_with_hooks` function to create a new 2-coin pool with hooks within the protocol.
- The user can call the `new_3_pool_with_hooks` function to create a new 3-coin pool with hooks within the protocol.
- The user can call the `new_4_pool_with_hooks` function to create a new 4-coin pool with hooks within the protocol.
- The user can call the `new_5_pool_with_hooks` function to create a new 5-coin pool with hooks within the protocol.
- The user can call the `swap_with_hooks` function to swap one type of coin for another within the protocol, utilizing hooks.
- The user can call `add_liquidity_2_pool_with_hooks` to add liquidity using hooks.

- The user can call `add_liquidity_3_pool_with_hooks` to add liquidity using hooks.
- The user can call `add_liquidity_4_pool_with_hooks` to add liquidity using hooks.
- The user can call `add_liquidity_5_pool_with_hooks` to add liquidity using hooks.
- The user can call `donate_with_hooks` to donate a specified coin to a pool with optional hooks.
- The user can invoke `remove_liquidity_2_pool_with_hooks` to remove liquidity with optional hooks.
- The user can invoke `remove_liquidity_3_pool_with_hooks` to remove liquidity with optional hooks.
- The user can invoke `remove_liquidity_4_pool_with_hooks` to remove liquidity with optional hooks.
- The user can invoke `remove_liquidity_5_pool_with_hooks` to remove liquidity with optional hooks.
- The user can invoke `remove_liquidity_one_coin_with_hooks` to remove liquidity from a pool with one coin, utilizing hooks for customization.

4 Findings

STA-1 Missing Fees for Unbalanced Liquidity

Severity: Major

Status: Fixed

Code Location:

contracts/sources/stable.move#1309-1346

Descriptions:

The current contract's implementation of adding liquidity, such as in the `add_liquidity_2_pool_impl` function, does not charge any fees. When users add liquidity in an unbalanced manner, a fee should be applied. Adding unbalanced liquidity is effectively equivalent to adding balanced liquidity followed by a swap. If fees are not charged for adding unbalanced liquidity, users can exploit this by adding unbalanced liquidity and then removing it to achieve the effect of a swap, thereby evading swap fees. The aforementioned issue also applies to the `remove_liquidity_one_coin_impl` function.

```
 fun add_liquidity_2_pool_impl<CoinA, CoinB, LpCoin>(
   pool: &mut InterestPool<Stable>,
   clock: &Clock,
   coin_a: Coin<CoinA>,
   coin_b: Coin<CoinB>,
   lp_coin_min_amount: u64,
   ctx: &mut TxContext
 ): Coin<LpCoin> {
   assert!(pool.are_coins_ordered(vector[type_name::get<CoinA>(),
type_name::get<CoinB>()]), errors::coins_must_be_in_order());

   let pool_address = pool.addy();
   let coins = pool.coins();
   let state = load_mut<LpCoin>(pool.state_mut());
   let prev_invariant = virtual_price_impl(state, clock);
   let amp = get_a(state.initial_a, state.initial_a_time, state.future_a, state.future_a_time,
clock);
   let prev_k = invariant_(amp, state.balances);
   let coin_a_value = deposit_coin<CoinA, LpCoin>(state, coin_a);
   let coin_b_value = deposit_coin<CoinB, LpCoin>(state, coin_b);
```

```
    let mint_amount = calculate_mint_amount(state, amp, prev_k, lp_coin_min_amount);
    events::add_liquidity(
      pool_address,
      coins,
      vector[coin_a_value, coin_b_value],
      mint_amount
    );
    let lp_coin = state.lp_coin_supply.increase_supply(mint_amount).to_coin(ctx);
    assert!(virtual_price_impl(load<LpCoin>(pool.state_mut()), clock) >= prev_invariant,
errors::invalid_invariant());
    lp_coin
  }
```

Suggestion:

It is recommended to introduce a fee when adding unbalanced liquidity. A reference implementation can be seen in Curve's StableSwap. The aforementioned issue also applies to the `remove_liquidity_one_coin_impl` function.

Resolution:

This issue has been fixed. The client has introduced a fee when adding and removing unbalanced liquidity.

STA-2 Dual Fee Structure for Swap Operations

Severity: Major

Status: Fixed

Code Location:

contracts/sources/stable.move#1227-1307

Descriptions:

In the Curve StableSwap contract, fees are only charged on `dy` (equivalent to `amount_out` in the current contract) during the `exchange` function. However, the current contract charges fees on both `amount_out` and `coin_in_value` during a swap, referred to as `fee_out` and `fee_in`.

```
let fee_in = stable_fees::calculate_fee_in_amount(&state.fees, coin_in_value);
    let  admin_fee_in = stable_fees::calculate_admin_amount(&state.fees, fee_in);
    let admin_coin_in = coin_in.split(admin_fee_in, ctx);
```

```
    let new_out_balance = y(
      amp,
      coin_in_index.to_u256(),
      coin_out_index.to_u256(),
      state.balances[coin_in_index] + normalized_value,
      state.balances
    );

    let normalized_amount_out = state.balances[coin_out_index] - new_out_balance;
    let amount_out = (normalized_amount_out * coin_out_decimals / PRECISION).to_u64();

    let fee_out = stable_fees::calculate_fee_out_amount(&state.fees, amount_out);
    let admin_fee_out = stable_fees::calculate_admin_amount(&state.fees, fee_out);

    let amount_out = amount_out - fee_out - admin_fee_out;
```

Suggestion:

It is recommended to consider revising the `swap` function to only charge fees on `amount_out`, similar to Curve's approach.

Resolution:

This issue has been fixed. The client has revised the `swap` function to only charge fees on `amount_out`, similar to Curve's approach.

STA-3 Users can Avoid the Fees during the Exchange

Severity: Major

Status: Fixed

Code Location:

contracts/sources/stable.move#1245,1246

Descriptions:

In the swap_impl.swap() function, the protocol charges a fee and an admin fee. The fee is calculated as follows `fee = x*percent/WAD` where `WAD = 1e18` :

```
let fee_in = stable_fees::calculate_fee_in_amount(&state.fees, coin_in_value);
let  admin_fee_in = stable_fees::calculate_admin_amount(&state.fees, fee_in);
```

When `x * percent< WAD` , the calculated fee will be 0. This allows users to avoid paying the fee and the admin fee during the exchange.

Suggestion:

It is recommended to check that `fee_in` and `admin_fee_in` are greater than 0.

Resolution:

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

STA-4 Centralization Risk

Severity: Major

Status: Acknowledged

Code Location:

contracts/sources/stable.move;

contracts/sources/volatile.move

Descriptions:

In the current system, administrators have the following primary operational privileges:

1. Adjusting the amplification coefficient of the liquidity pool (`ramp`)
2. Stopping the adjustment of the amplification coefficient of the liquidity pool (`stop_ramp`)
3. Updating transaction fees (`update_fee`)

These operations have significant impacts on the system, and the centralized management of these privileges can lead to the following issues:

1. Single Point of Failure: If the administrator's account is compromised or an erroneous operation occurs, it could have a substantial impact on the entire liquidity pool.
2. Transparency and Trust: Users need to have sufficient trust in the decision-making process and motivations of the administrator. If the administrator makes decisions that are not in the users' best interests, it could damage the system's reputation and user confidence.

Suggestion:

It is recommended to use the multi-sig wallets to mitigate the centralized risk.

STA-5 The First User cannot immediately Remove Liquidity after Adding it

Severity: Medium

Status: Fixed

Code Location:

contracts/sources/stable.move#1580

Descriptions:

After the first user adds liquidity, they cannot immediately remove liquidity because of the check `virtual_price_impl(load<LpCoin>(pool.state_mut()), clock) >= prev_invariant`.

```
    state.lp_coin_supply.decrease_supply(lp_coin.into_balance());

    events::remove_liquidity(
      pool_address,
      coins,
      vector[coin_a.value(), coin_b.value()],
      lp_coin_value
    );

    assert!(virtual_price_impl(load<LpCoin>(pool.state_mut()), clock) >= prev_invariant,
errors::invalid_invariant());

    (coin_a, coin_b)
```

PoC:

```
 #[test]
  fun remove_liquidityAll() {
    let mut scenario = scenario();
    let (alice, _) = people();

    let test = &mut scenario;

    setup_2pool(test, 900, 1000);

    next_tx(test, alice);
    {
```

```
    let mut pool = test::take_shared<InterestPool<Stable>>(test);

    let supply = interest_clamm_stable::lp_coin_supply<LP_COIN>(&mut pool);

    let c = clock::create_for_testing(ctx(test));

    let(coin_usdc, coin_usdt) = interest_clamm_stable::remove_liquidity_2_pool<USDC,
USDT, LP_COIN>(
      &mut pool,
      &c,
      mint1<LP_COIN>(supply, ctx(test)),
      vector[0, 0, 0],
      ctx(test)
    );

     debug::print(&coin_usdc);
     debug::print(&coin_usdt);
    let balances_2 = interest_clamm_stable::balances<LP_COIN>(&mut pool);
    let supply_2 = interest_clamm_stable::lp_coin_supply<LP_COIN>(&mut pool);


     let value1 = burn(coin_usdc);
     let value2 = burn(coin_usdt);


    clock::destroy_for_testing(c);

    test::return_shared(pool);
  };
  test::end(scenario);
 }
```

After running, the protocol will throw an error. The error message is as follows.

```
remove_liquidityAll
error[E11001]: test failure
        ./sources/stable.move:1576:5


1540      fun remove_liquidity_2_pool_impl<CoinA, CoinB, LpCoin>(
            ---------------------------- In this function in 0x0::interest_clamm_stable
    .
```

```
1576        assert!(virtual_price_impl(load<LpCoin>(pool.state_mut()), clock) >= prev_invariant,
errors::invalid_invariant());
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Test was not expected to error, but it
aborted with code 6 originating in the module
0000000000000000000000000000000000000000000000000000000000000000::
interest_clamm_stable rooted here
```

Suggestion:

It is recommended to consider this exceptional situation.

Resolution:

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

STA-6 Missing Emergency Pause Functionality

Severity: Medium

Status: Fixed

Code Location:

contracts/sources/stable.move;

contracts/sources/volatile.move

Descriptions:

The pause functionality is used to temporarily halt the contract's operations in the event of an attack or other emergency situations. This functionality is missing in the current contract, making it impossible to respond promptly and prevent losses in case of potential security threats. A reference implementation can be seen in Curve's code:

```
is_killed: public(bool)
```

```
@external
def kill_me():
    assert msg.sender == self.owner  # dev: only owner
    assert self.kill_deadline > block.timestamp  # dev: deadline has passed
    self.is_killed = True


@external
def unkill_me():
    assert msg.sender == self.owner  # dev: only owner
    self.is_killed = False
```

Suggestion:

It is recommended to introduce the above pause functionality into the existing contract(`add_liquidity` , `remove_liquidity_one_coin` , `swap`).

Resolution:

This issue has been fixed. The client has introduced the pause functionality.

VOL-1 Lack of Timelock for Critical Admin Operations

Severity: Medium

Status: Fixed

Code Location:

contracts/sources/volatile.move#907-941;

contracts/sources/stable_fees.move#32-54

Descriptions:

Critical admin operations, such as changing fee parameters, should incorporate a timelock mechanism. This ensures that changes do not take effect immediately, providing a buffer period for users to respond to potential adverse changes. The current contract lacks this timelock feature, which could lead to sudden and potentially harmful modifications without giving users adequate time to react.

Suggestion:

It is recommended to introduce a timelock mechanism for all critical admin operations to enhance the security and trustworthiness of the contract. A reference implementation can be seen in Curve's code:

```
@external
def commit_new_fee(_new_fee: uint256, _new_admin_fee: uint256):
    assert msg.sender == self.owner  # Only the owner can execute
    assert self.admin_actions_deadline == 0  # No active action
    assert _new_fee <= MAX_FEE  # Fee exceeds maximum
    assert _new_admin_fee <= MAX_ADMIN_FEE  # Admin fee exceeds maximum

    deadline: uint256 = block.timestamp + ADMIN_ACTIONS_DELAY
    self.admin_actions_deadline = deadline
    self.future_fee = _new_fee
    self.future_admin_fee = _new_admin_fee

    log CommitNewFee(deadline, _new_fee, _new_admin_fee)

@external
def apply_new_fee():
    assert msg.sender == self.owner  # Only the owner can execute
```

```
    assert block.timestamp >= self.admin_actions_deadline  # Insufficient time has passed
    assert self.admin_actions_deadline != 0  # No active action

    self.admin_actions_deadline = 0
    fee: uint256 = self.future_fee
    admin_fee: uint256 = self.future_admin_fee
    self.fee = fee
    self.admin_fee = admin_fee

    log NewFee(fee, admin_fee)
```

Resolution:

This issue has been fixed. The client has introduced a timelock mechanism for all critical admin operations.

VOL-2 The Commit Operation Lacks Parameter Validation

Severity: Minor

Status: Fixed

Code Location:

contracts/sources/volatile.move#1007-1043;

contracts/sources/stable.move#912-945

Descriptions:

The `commit_parameters` and `commit_fee` functions lack parameter validation for `update_deadline`. Subsequent commit operations should not be allowed to overwrite previous ones. A check similar to the one in Curve should be added: `assert self.admin_actions_deadline == 0`. Additionally, when performing the update operation, the following checks should be added:

```
assert self.admin_actions_deadline != 0  # dev: no active action
self.admin_actions_deadline = 0
```

Suggestion:

It is recommended to add parameter validation.

Resolution:

This issue has been fixed. The client has added parameter validation.

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.