Cetus

Audit

Presented by:

OtterSec contact@osec.io

Michal Bochnak embe221ed@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Cetus Protocol engaged OtterSec to perform an assessment of the `cetus-clmm` program version written in Sui. This assessment was conducted between March 30th and May 3rd, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 9 findings total.

In particular, we found an issue regarding an unchecked `Position<CoinA, CoinB>` in `add_liquidity` (OS-CTS-ADV-00).

Additionally, we made recommendations to improve code quality by replacing the current casting function to handle negative tick indexes (OS-CTS-SUG-00) and changing the behaviour of a function to better reflect the Aptos version (OS-CTS-SUG-01).

02 | Scope

The source code was delivered to us in a git repository at github.com/CetusProtocol/cetus-clmm-sui. This audit was performed against commit 11d65f5.

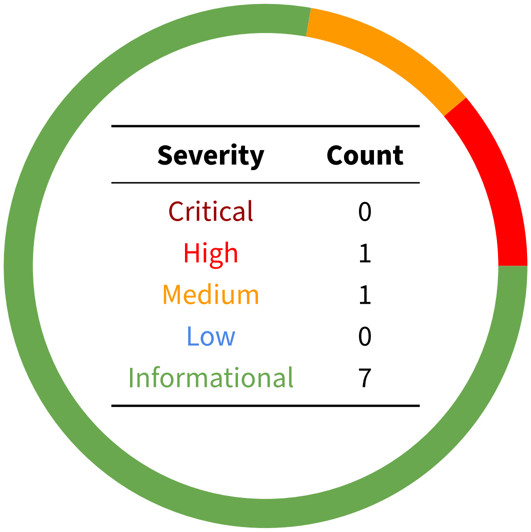
A brief description of the programs is as follows.

Name	Description
cetus-clmm	A concentrated liquidity market maker program built on Sui.

03 | Findings

Overall, we reported 9 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	1
Medium	1
Low	0
Informational	7

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-CTS-ADV-00	High	Resolved	add_liquidity does not validate the passed Position<CoinA, CoinB> against the relevant Pool<CoinA, CoinB> to which the liquidity will be added.
OS-CTS-ADV-01	Medium	Resolved	update_emission enables the calculation of rewards for CoinType with a balance of zero if emission_per_day is equal to zero.

OS-CTS-ADV-00 [high] | Possible Position Confusion In Add Liquidity

Description

add_liquidity_internal does not check if the provided Position<CoinA, CoinB> belongs to the same Pool<CoinA, CoinB> as the one passed as an argument. Therefore, it is possible to pass a position from a pool created for the same pair of coins but with a different tick_spacing.

sui/clmmpool/sources/pool.move RUST

```
fun add_liquidity_internal<CoinTypeA, CoinTypeB>(
    pool: &mut Pool<CoinTypeA, CoinTypeB>,
    position: &mut Position<CoinTypeA, CoinTypeB>,
    by_amount: bool,
    liquidity: u128,
    amount: u64,
    fix_amount_a: bool,
    ctx: &mut TxContext
): AddLiquidityReceipt<CoinTypeA, CoinTypeB> {
    // 1. Check position and pool
    assert_status(pool);
    // ...
}
```

We have prepared a test case that demonstrates a potential attack scenario that exploits the lack of validation in add_liquidity_internal.

RUST

```
#[test]
fun test_add_liquidity_wrong() {
    use std::debug;
    let ctx = &mut tx_context::dummy();
    let (tick_spacing, fee_rate, init_sqrt_price) = (50, 2000,
    ↪  2264044300179098811);

    let items = vector::empty<PositionItem>();
    vector::push_back(&mut items, PositionItem {
        liquidity: 640335940,
        tick_lower: i32::as_u32(i32::neg_from(42000)),
        tick_upper: i32::as_u32(i32::neg_from(40000)),
        amount_a: 16752440,
        amount_b: 0
    });
    vector::push_back(&mut items, PositionItem {
        liquidity: 2317299527,
        tick_lower: i32::as_u32(i32::neg_from(33150)),
        tick_upper: i32::as_u32(i32::neg_from(33050)),
        amount_a: 61541268,
        amount_b: 0
    });

    let pool0 = new_pool_for_testing<CoinA, CoinB>(ctx, tick_spacing,
    ↪   fee_rate, init_sqrt_price, 0);
    let pool1 = new_pool_for_testing<CoinA, CoinB>(ctx, 20, fee_rate,
    ↪   init_sqrt_price, 0);

    // valid
    let item = *vector::borrow(&items, 0);
    let position = open_position(&mut pool0, item.tick_lower,
    ↪   item.tick_upper, ctx);
    let receipt = add_liquidity(&mut pool0, &mut position, item.liquidity,
    ↪   ctx);
    let coin_a = coin::mint_for_testing<CoinA>(receipt.amount_a, ctx);
    let coin_b = coin::mint_for_testing<CoinB>(receipt.amount_b, ctx);
    repay_add_liquidity(&mut pool0, coin::into_balance(coin_a),
    ↪   coin::into_balance(coin_b), receipt);

    // invalid
    let item = *vector::borrow(&items, 1);
    let receipt = add_liquidity(&mut pool1, &mut position, item.liquidity,
    ↪   ctx);
    let coin_a = coin::mint_for_testing<CoinA>(receipt.amount_a, ctx);
    let coin_b = coin::mint_for_testing<CoinB>(receipt.amount_b, ctx);
    repay_add_liquidity(&mut pool1, coin::into_balance(coin_a),
    ↪   coin::into_balance(coin_b), receipt);

    debug::print(&pool0.liquidity);
    debug::print(&pool1.liquidity);

    let (bal0, bal1) = remove_liquidity(&mut pool0, &mut position,
    ↪   640335940, 0, 0, ctx);
    let coin_a_holder = balance::zero<CoinA>();
    let coin_b_holder = balance::zero<CoinB>();

    balance::join(&mut coin_a_holder, bal0);
    balance::join(&mut coin_b_holder, bal1);
    transfer::transfer(coin::from_balance(coin_a_holder, ctx),
    ↪   tx_context::sender(ctx));
    transfer::transfer(coin::from_balance(coin_b_holder, ctx),
     ↪  tx_context::sender(ctx));
    transfer::transfer(position, tx_context::sender(ctx));

    debug::print(&pool0.liquidity);
    debug::print(&pool1.liquidity);

    transfer::transfer(pool0, tx_context::sender(ctx));
    transfer::transfer(pool1, tx_context::sender(ctx));
}
```

Remediation

Ensure that `add_liquidity_internal` validates that the `Position<CoinA, CoinB>` passed as an argument was created for the `Pool<CoinA, CoinB>` provided to the function.

Patch

Fixed in ec89b4f.

OS-CTS-ADV-01 [med] | Incorrect Balance Validation

Description

update_emission calculates emission_per_day with emission_per_second. Passing emission_per_second $\leq (1 << 64)/$DAYS_IN_SECONDS results in emission_per_day being equal to zero.

Therefore, it is possible to bypass the assert statement that checks rewarder_balance >= emission_per_day even if the balance is zero. As a result, emission_per_second is used to calculate the rewards for the CoinType with a balance of zero, which leads to an ENotEnough error.

sui/clmmpool/sources/rewarder.move RUST

```rust
public(friend) fun update_emission<CoinType>(
    vault: &RewarderGlobalVault,
    manager: &mut RewarderManager,
    liquidity: u128,
    emissions_per_second: u128,
    timestamp: u64,
) {
    settle(manager, liquidity, timestamp);
    let emission_per_day = full_math_u128::mul_shr(DAYS_IN_SECONDS,
    ↪   emissions_per_second, 64);
    let rewarder = borrow_mut_rewarder<CoinType>(manager);
    let reward_type = type_name::get<CoinType>();
    assert!(bag::contains(&vault.balances, reward_type),
    ↪   ERewardAmountInsufficient);
    let rewarder_balance = bag::borrow<TypeName, Balance<CoinType>>(
        &vault.balances,
        reward_type
    );
    assert!(balance::value<CoinType>(rewarder_balance) >=
    ↪   (emission_per_day as u64), ERewardAmountInsufficient);
    rewarder.emissions_per_second = emissions_per_second;
}
```

Remediation

Tighten balance validation to prevent bypassing the assert statement when emission_per_day equals zero.

Patch

Fixed in 3406829

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-CTS-SUG-00	fetch_ticks cannot handle negative tick indexes due to an incorrect casting function being used.
OS-CTS-SUG-01	The Sui version of the code does not include the tick at the index that is equal to the tick_idx passed as a parameter, while the Aptos version does.
OS-CTS-SUG-02	flash_swap_with_partner does not validate if the pool is paused.
OS-CTS-SUG-03	deposit_reward allows any type of coin to be deposited, resulting in blocking certain funds.
OS-CTS-SUG-04	Casting the return value of u256 to u64 in the compute_swap_step function may cause arithmetic errors if the returned value exceeds MAX_U64.
OS-CTS-SUG-05	Getter functions do not perform version checks.
OS-CTS-SUG-06	A swap operation may be performed with either amount_out or amount_in set to zero.

OS-CTS-SUG-00 | Incorrect Tick Cast Function Used

Description

fetch_ticks is currently utilizing i32::from to convert u32 into I32. However, i32::from does not allow for values greater than U32_MAX, and therefore, cannot generate negative values from the conversion.

integer-mate/sui/sources/i32.move RUST

```
const MAX_AS_U32: u32 = 0x7fffffff;
---
public fun from(v: u32): I32 {
    assert!(v <= MAX_AS_U32, EOverflow);
    I32 {
        bits: v
    }
}
```

Remediation

Utilize i32::from_u32.

DIFF

```
diff --git a/sui/clmmpool/sources/tick.move
    ↪  b/sui/clmmpool/sources/tick.move
index 60a94e4..4a0766b 100644
--- a/sui/clmmpool/sources/tick.move
+++ b/sui/clmmpool/sources/tick.move
@@ -490,7 +490,7 @@ module cetus_clmm::tick {
         let opt_next_score = if (vector::is_empty(&start)) {
             skip_list::head(&manager.ticks)
         } else {
-            let score = tick_score(i32::from(*vector::borrow(&start,
    ↪  0)));
+            let score = tick_score(i32::from_u32(*vector::borrow(&start,
    ↪  0)));
             skip_list::find_next(&manager.ticks, score, false)
         };

@@ -554,4 +554,4 @@ module cetus_clmm::tick {
         }
     }
```

Patch

Fixed in 03c36bc.

OS-CTS-SUG-01 | Greater Or Equal Ticks Returned In Aptos

Description

There is a discrepancy between the Aptos and Sui versions regarding the output of `fetch_ticks`. The Aptos version includes the tick at the index equal to the `tick_idx` parameter.

aptos/clmmpool/sources/pool.move RUST

```rust
public fun fetch_ticks<CoinTypeA, CoinTypeB>(
    pool_address: address, index: u64, offset: u64, limit: u64
): (u64, u64, vector<Tick>) acquires Pool {
    let pool = borrow_global_mut<Pool<CoinTypeA,
    ↪  CoinTypeB>>(pool_address);
    let tick_spacing = pool.tick_spacing;
    let max_indexes_index = tick_indexes_max(tick_spacing);
    let search_indexes_index = index;
    let ticks = vector::empty<Tick>();
    let offset = offset;
    let count = 0;
    while ((search_indexes_index >= 0) && (search_indexes_index <=
    ↪  max_indexes_index)) {
        if (table::contains(&pool.tick_indexes, search_indexes_index)) {
            let indexes = table::borrow(&pool.tick_indexes,
    ↪  search_indexes_index);
            while ((offset >= 0) && (offset < TICK_INDEXES_LENGTH)) {
                if (bit_vector::is_index_set(indexes, offset)) {
                    let tick_idx = i64::sub(
                        i64::from((TICK_INDEXES_LENGTH *
    ↪  search_indexes_index + offset) * tick_spacing),
                        tick_max(tick_spacing)
                    );
                    let tick = table::borrow(&pool.ticks, tick_idx);
                    count = count + 1;
                    vector::push_back(&mut ticks, *tick);
                    if (count == limit) {
                        return (search_indexes_index, offset, ticks)
                    }
                };
                offset = offset + 1;
            };
            offset = 0;
        };
        search_indexes_index = search_indexes_index + 1;
    };
    (search_indexes_index, offset, ticks)
}
```

However, the Sui version does not.

sui/clmmpool/sources/tick.move RUST

```
public fun fetch_ticks(
    manager: &TickManager,
    start: vector<u32>,
    limit: u64
): vector<Tick> {
    let ticks = vector::empty<Tick>();
    let opt_next_score = if (vector::is_empty(&start)) {
        skip_list::head(&manager.ticks)
    } else {
        let score = tick_score(i32::from_u32(*vector::borrow(&start,
     ↪  0)));
        skip_list::find_next(&manager.ticks, score, false)
    };

    let count = 0;
    while (option_u64::is_some(&opt_next_score)) {
        let node = skip_list::borrow_node(&manager.ticks,
     ↪  option_u64::borrow(&opt_next_score));
        vector::push_back(&mut ticks, *skip_list::borrow_value(node));
        opt_next_score = skip_list::next_score(node);
        count = count + 1;
        if (count == limit) {
            break
        }
    };
    ticks
}
```

Remediation

Change the `include` value from `false` to `true` to include the tick at the index passed to `find_next`.

OS-CTS-SUG-02 | Add Pause Check

Description

The `assert!(!pool.is_pause, EPoolIsPaused)` check was added to `flash_swap`. A similar check should be added to `flash_swap_with_partner`.

Remediation

Add a `assert!(!pool.is_pause, EPoolIsPaused)` line at the beginning of `flash_swap_with_partner` as it was executed for `flash_swap`.

OS-CTS-SUG-03 | Coin Locked In Vault

Description

`deposit_reward` allows any type of coin to be deposited in `RewarderGlobalVault`. However, it is impossible for the coins to exit the vault unassisted, as they have been locked within, and the vault may be filled with coins of any arbitrary type.

Remediation

Abort the execution if an out-of-bound condition occurs.

Patch

The Cetus team voiced that it is impossible to anticipate which rewards will be required in advance, and therefore, the `emergent_withdraw` admin function is utilized to permit the withdrawal of funds.

OS-CTS-SUG-04 | Possible Arithmetic Errors

Description

In compute_swap_step, get_delta_down_from_output and get_delta_up_from_input return values of type u256, which are then cast to u64. If they exceed MAX_U64, arithmetic errors may occur.

Remediation

Check if the return values exceed MAX_U64.

Patch

The Cetus team responded that it is highly unlikely for this scenario to occur and will not implement a patch.

OS-CTS-SUG-05 | Lack Of Version Checks In Getter Functions

Description

`checked_package_version` is employed to ensure that the latest contract version is being invoked. However, this protection mechanism is not utilized by all functions. In the case of getter functions that may return objects, it would be prudent to call `checked_package_version` for added safety.

Remediation

Add a `checked_package_version` function call to getter functions.

OS-CTS-SUG-06 | Add Zero Amount Check

Description

Performing a swap operation that sets either `amount_in` or `amount_out` to zero can lead to the swap operation being executed in the pool with zero liquidity, allowing the `current_sqrt_price` to be set to a semi-arbitrary value.

Proof of Concept

sui/clmmpool/sources/tests/pool_tests.move RUST

```rust
#[test]
fun test_swap_to_target_price() {
    let ctx = &mut tx_context::dummy();
    let (clock, admin_cap, config) = init_test(ctx);
    let pool = pool::new_for_test<CoinA, CoinB>(
        60,
        get_sqrt_price_at_tick(i32::from(0)),
        2000,
        string::utf8(b""),
        0,
        &clock,
        ctx
    );
    let target_sqrt_price = get_sqrt_price_at_tick(i32::from(420000));
    let (recv_amount, pay_amount) = swap(
        &mut pool,
        &config,
        false,
        true,
        1000000000,
        target_sqrt_price,
        &clock,
        ctx
    );
    print(&string::utf8(b"=========== attack ==========="));
    print(&string::utf8(b"recv amount: "));print(&recv_amount);
    print(&string::utf8(b"pay amount: "));print(&pay_amount);
    print(&string::utf8(b"current_sqrt_price:
     ↪  "));print(&pool::current_sqrt_price(&pool));
    print(&string::utf8(b"=============================="));
    print(&string::utf8(b"add_liquidity"));
    add_liquidity_for_swap(&config, &mut pool,  nt(426000), pt(426000),
     ↪  1000000000, &clock, ctx);
    print(&string::utf8(b"=========== victim ==========="));
    let target_sqrt_price = get_sqrt_price_at_tick(i32::from(443580));
    let (recv_amount, pay_amount) = swap(
        &mut pool,
        &config,
        false,
        true,
        100000,
        target_sqrt_price,
        &clock,
        ctx
    );
    print(&string::utf8(b"recv amount: "));print(&recv_amount);
    print(&string::utf8(b"pay amount: "));print(&pay_amount);
    print(&string::utf8(b"current_sqrt_price:
     ↪  "));print(&pool::current_sqrt_price(&pool));
    transfer::public_share_object(pool);
    close_test(admin_cap, config, clock);
}
```

sui/clmmpool/sources/tests/pool_tests.move RUST

```
Running Move unit tests
[debug] "=========== attack ==========="
[debug] "recv amount: "
[debug] 0
[debug] "pay amount: "
[debug] 0
[debug] "current_sqrt_price: "
[debug] 24302327192315584785160762634
[debug] "=============================="
[debug] "add_liquidity"
[debug] "=========== victim ==========="
[debug] "recv amount: "
[debug] 0
[debug] "pay amount: "
[debug] 100000
[debug] "current_sqrt_price: "
[debug] 24302327192317425770219318847
```

Remediation

Abort the execution if either `amount_in` or `amount_out` is zero.

Patch

Fixed in 3406829

A Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities that immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority or access control validation
- Improperly designed economic incentives leading to loss of funds

High

Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities that could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.