Cetus

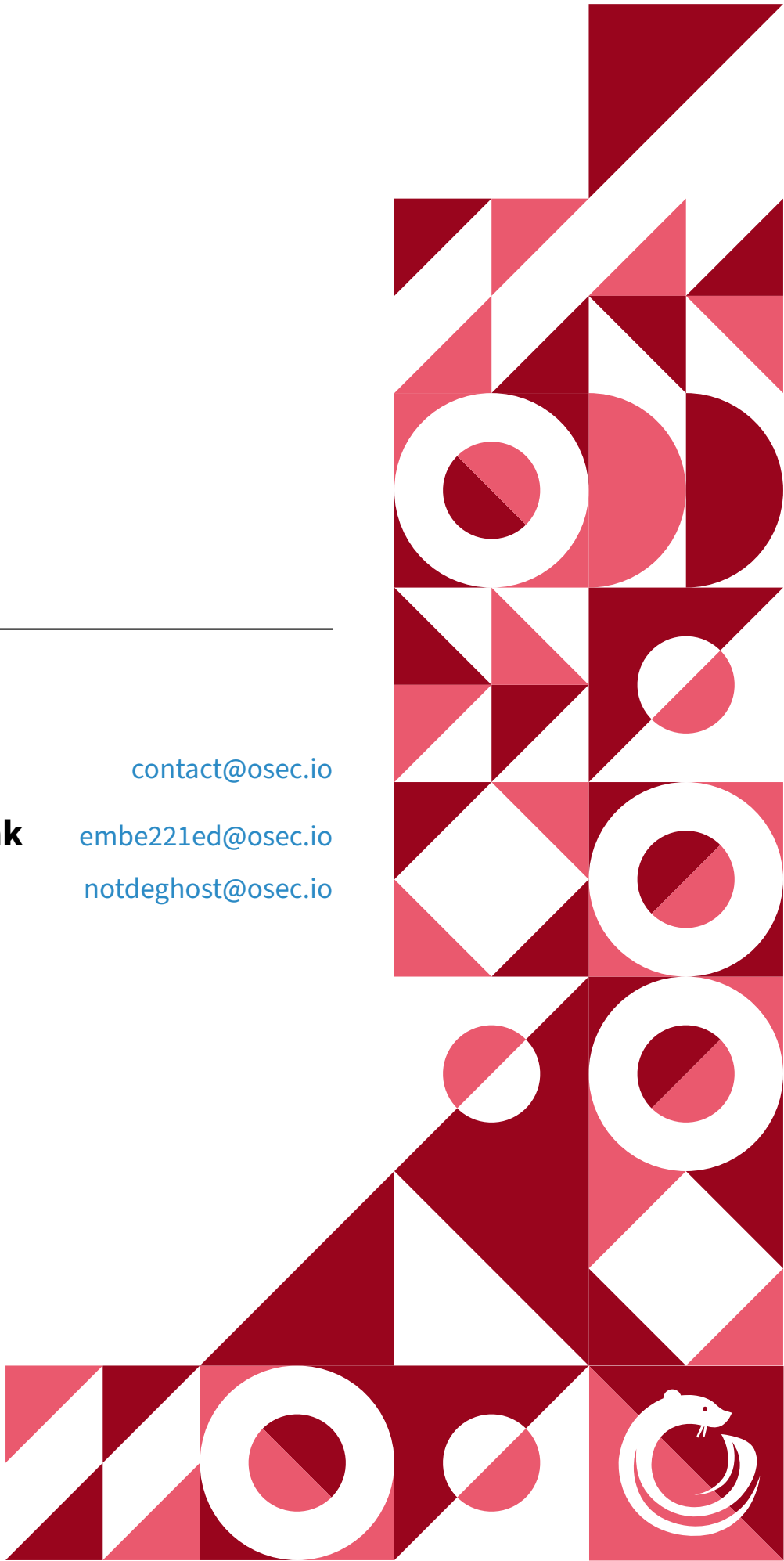
Audit

Presented by:

OtterSec contact@osec.io

Michal Bochnak embe221ed@osec.io

Robert Chen notdeghost@osec.io



Contents

Appendices

01 | Executive Summary

Overview

Cetus Protocol engaged OtterSec to perform an assessment of the `cetus-clmm` program. This assessment was conducted between December 12th, 2022 and January 8th, 2023. For more information on our auditing methodology, see Appendix B.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches January 8th, 2023.

Key Findings

Over the course of this audit engagement, we produced 7 findings total.

In particular, we found an issue with an unchecked overflow in the `get_delta_b` function (OS-CTS-ADV-00).

Additionally, we made recommendations in order to improve the code, such as, removing unnecessary castings(OS-CTS-SUG-02), as well as adding fields that will make calculations of the token amounts needed possible (OS-CTS-SUG-01).

Overall, we commend the Cetus team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/CetusProtocol/cetus-clmm. This audit was performed against commit e56d476.

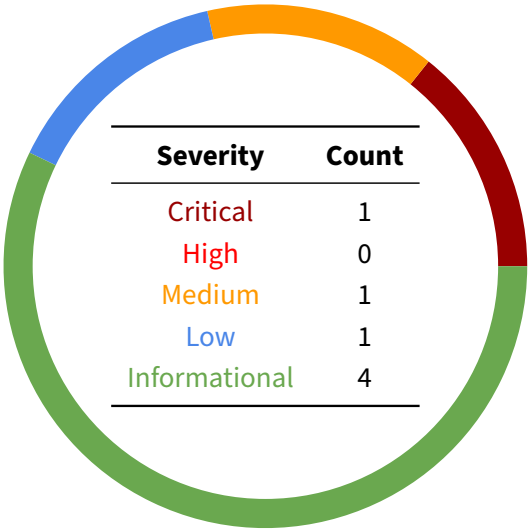
A brief description of the programs is as follows.

name	Description
cetus-clmm	Concentrated liquidity market maker program built on Aptos.

03 | Findings

Overall, we report 7 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	1
High	0
Medium	1
Low	1
Informational	4

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	Critical	Resolved	An unchecked overflow in the `get_delta_b` function.
OS-CTS-ADV-01	Medium	Resolved	An unchecked overflow in the `get_delta_a` function.
OS-CTS-ADV-02	Low	Resolved	Potential overflow in `get_next_sqrt_price_a_up`.

OS-CTS-ADV-00 [crit] [resolved] | Overflow In Calculating Delta B

Description

The function `get_delta_b` is used to calculate the `amount_b` for specified `liquidity`. However, its implementation relies on the assumption that the multiplication of `liquidity` and `sqrt_price_diff` returns the value < `2**128` which does not require to be true.

```
let product = full_math_u128::full_mul(liquidity, sqrt_price_diff);
let should_round_up = (round_up) && (u256::get(&product, 0) > 0);
if (should_round_up) {
  return (u256::get(&product, 1) + 1)
};
(u256::get(&product, 1))
```

Proof of Concept

In order to prepare Proof of Concept, we edited the `test_add_liquidity` case in `pool.move`

```
let pool_address = init_test_pool(clmm, tick_spacing, fee_rate,
    ↪  tick_math::get_sqrt_price_at_tick(i64::from(300100)));
let caps = borrow_global<TestCaps>(signer::address_of(clmm));
let (_amount_a, _amount_b, liquidity) = (0, 0, 112942705988161);

let position_index = open_position<CoinA, CoinB>(
    owner,
    pool_address,
    i64::neg_from(300000),
    i64::from(300000)
);
let receipt = add_liquidity(pool_address, liquidity, position_index);
debug::print(&receipt);
let coin_a = coin::mint(receipt.amount_a, &caps.mint_a);
let coin_b = coin::mint(receipt.amount_b, &caps.mint_b);
repay_add_liquidity(coin_a, coin_b, receipt);
{
  let coin_a = coin::mint(11529215046068469760, &caps.mint_a);
  let coin_b = coin::mint(11529215046068469760, &caps.mint_b);
  let pool = borrow_global_mut<Pool<CoinA, CoinB>>(pool_address);
  coin::merge(&mut pool.coin_a, coin_a);
```

```
  coin::merge(&mut pool.coin_b, coin_b);
};
let coin_b_holder = coin::zero<CoinB>();
let i = 0;
while (i <= 2) {
  let (coin_a, coin_b) = remove_liquidity<CoinA, CoinB>(
      owner,
      pool_address,
      liquidity / 1000,
      position_index
  );
  coin::destroy_zero(coin_a);
  coin::merge(&mut coin_b_holder, coin_b);
  i = i + 1;
};
debug::print(&coin_b_holder);
coin::burn(coin_b_holder, &caps.burn_b);
```

And the result of running this test

```
Running Move unit tests
[debug] 0x1::pool::AddLiquidityReceipt<0x1::pool::CoinA,
    ↪  0x1::pool::CoinB> {
  pool_address: @0x...,
  amount_a: 0,
  amount_b: 4292105107
}
[debug] 0x1::coin::Coin<0x1::pool::CoinB> {
  value: 11068046444433871660
}
[ PASS    ] 0x1::pool::test_add_liquidity
```

Remediation

`product` value should be validated before getting `n1`.

```
+       if (u256::get(&product, 2) != 0 || u256::get(&product, 3) != 0) {
+         abort EMULTIPLICATION_OVERFLOW
+       }
        if (should_round_up) {
```

```
            return (u256::get(&product, 1) + 1)
        };

        (u256::get(&product, 1))
```

Patch

Fixed in c867755.

OS-CTS-ADV-01 [med] [resolved] | Overflow In Calculation Of Delta A

Description

The `numberator` value is not validated before running `u256::shlw` on it. As a result, the non-zero bytes might be removed, which leads to an incorrect calculation of the value.

```
let numberator = u256::shlw(full_math_u128::full_mul(liquidity,
    ↪  sqrt_price_diff));
```

Proof of Concept

```
get_delta_a(tick_math::max_sqrt_price(), tick_math::min_sqrt_price(),
    ↪  56315830353026631512438212669420532741, true);
```

result

```
numberator: [debug] 0x1::u256::U256 {
  n0: 535693272949614043,
  n1: 11711318139720635722,
  n2: 15182355392690797918,
  n3: 710792351
}
numberator >> 64: [debug] 0x1::u256::U256 {
  n0: 0,
  n1: 535693272949614043,
  n2: 11711318139720635722,
  n3: 15182355392690797918
}
```

Remediation

Replace `u256::shlw` usage with `u256::checked_shlw`.

```
-       let numberator = u256::shlw(full_math_u128::full_mul(liquidity,
    ↪   sqrt_price_diff));
+       let (numberator, overflow) =
    ↪   u256::checked_shlw(full_math_u128::full_mul(liquidity,
    ↪   sqrt_price_diff));
+       if (overflow) {
+         abort ERROR;
+       }
```

Patch

Fixed in f6a3888.

OS-CTS-ADV-02 [low] [resolved] | Overflow In Calculation Of Sqrt Price

Description

The `numberator` calculation in `get_next_sqrt_price_a_up` does not check if the `u256::shlw` operation removes the non-zero bytes.

```
let numberator = u256::shlw(full_math_u128::full_mul(sqrt_price,
    ↪  liquidity));
```

There is a possibility that `sqrt_price` multiplied by `liquidity` will return a result large enough that a shift left by 64 bits will remove the non-zero bits.

Proof of Concept

```
#[test]
fun test_get_next_price_a_up() {
    use aptos_std::debug;
    let (sqrt_price, liquidity, amount) = (7922631417992447579055,
    ↪  56315830353026631512438212669420532741, 10000000u64);
    let r1 = get_next_sqrt_price_a_up(sqrt_price, liquidity, amount,
    ↪  true);
    debug::print(&r1);
}
```

Result:

```
numberator: [debug] 0x...::u256::U256 {
  n0: 15300607055598014 51,
  n1: 10270621373656910829,
  n2: 1455684222633421829,
  n3: 71
}
numberator << 64: [debug] 0x...::u256::U256 {
  n0: 0,
  n1: 15300607055598014 51,
  n2: 10270621373656910829,
```

```
  n3: 1455684222633421829
}
r1: [debug] 8795815841869149120
```

Remediation

`u256::checked_shlw` should be used instead

```
-       let numberator = u256::shlw(full_math_u128::full_mul(sqrt_price,
    ↪  liquidity));
+       let (numberator, overflow) =
    ↪  u256::checked_shlw(full_math_u128::full_mul(sqrt_price,
    ↪  liquidity));
+       if (overflow) {
+         abort ERROR;
+       }
```

Patch

Fixed in e7a3c12.

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	It is possible to use an empty string as the partner name.
OS-CTS-SUG-01	Emission reward tokens are potentially incorrectly bounded.
OS-CTS-SUG-02	The `fee_rate` does not require casting.
OS-CTS-SUG-03	The `reset_init_price` function does not validate the given price.

OS-CTS-SUG-00 | Stricter Partner Name Validation

Description

The `create_partner` function allows `name` to be equal to `b""`.

Remediation

The `name` parameter should be compared with empty bytes to check if it contain any bytes.

```
public fun create_partner(
    account: &signer,
    name: String,
    fee_rate: u64,
    receiver: address,
    start_time: u64,
    end_time: u64,
) acquires Partners {
    config::assert_protocol_authority(account);

    assert!(end_time > start_time, error::aborted(EINVALID_TIME));
    assert!(fee_rate < MAX_PARTNER_FEE_RATE,
     ↪  error::invalid_argument(EINVALID_PARTNER_FEE_RATE));
+   assert!(string::is_empty(&name), error::invalid_arument(ERROR_NAME))

    let partners = borrow_global_mut<Partners>(@cetus_clmm);
```

Patch

Fixed in e7a3c12.

OS-CTS-SUG-01 | Incorrectly Calculated Emission

Description

The `pool::update_emission` function incorrectly validates the balance of `reward.authority`.

```
let emission_per_day = full_math_u128::mul_shr(DAYS_IN_SECONDS,
    ↪  emissions_per_second, 64);
// ...
assert!(
    coin::balance<CoinTypeC>(pool_address) >= (emission_per_day as u64),
    error::aborted(EREWARD_AMOUNT_INSUFFICIENT)
);
rewarder.emissions_per_second = emissions_per_second;
// ..
```

The balance of `reward.authority` is compared with `emission_per_day`, meaning that it is expected to hold the amount of tokens needed for one day. However, if the reward lasts for more than one day, the pool might run out of rewards.

Remediation

We suggest adding fields that will track the start and end time of the reward. Then it will be possible to properly bound and calculate the amount of tokens needed.

```
/// The clmmpools's Rewarder for provide additional liquidity incentives.
struct Rewarder has copy, drop, store {
    coin_type: TypeInfo,
    authority: address,
    pending_authority: address,
    emissions_per_second: u128,
    growth_global: u128,
+   start_time: u64,
+   end_time: u64
}
```

```
public fun update_emission<CoinTypeA, CoinTypeB, CoinTypeC>(
    account: &signer,
    pool_address: address,
    index: u8,
    emissions_per_second: u128
) acquires Pool {
    let pool = borrow_global_mut<Pool<CoinTypeA,
    ↪  CoinTypeB>>(pool_address);
    assert_status(pool);
    update_rewarder(pool);

-   let emission_per_day = full_math_u128::mul_shr(DAYS_IN_SECONDS,
    ↪  emissions_per_second, 64);
    assert!((index as u64) < vector::length(&pool.rewarder_infos),
    ↪  error::aborted(EINVALID_REWARD_INDEX));
    let rewarder = vector::borrow_mut(&mut pool.rewarder_infos, (index as
    ↪  u64));
+   let emission_per_time = full_math_u128::mul_shr(rewarder.end_time -
    ↪  rewarder.start_time, emissions_per_second, 64);
    assert!(signer::address_of(account) == rewarder.authority,
    ↪  error::permission_denied(EREWARD_AUTH_ERROR));
    assert!(rewarder.coin_type == type_of<CoinTypeC>(),
    ↪  error::aborted(EREWARD_NOT_MATCH_WITH_INDEX));
    assert!(
-       coin::balance<CoinTypeC>(pool_address) >= (emission_per_day as
    ↪  u64),
+       coin::balance<CoinTypeC>(pool_address) >= (emission_per_time as
    ↪  u64),
        error::aborted(EREWARD_AMOUNT_INSUFFICIENT)
    );
    rewarder.emissions_per_second = emissions_per_second;
    event::emit_event(&mut pool.update_emission_events,
    ↪  UpdateEmissionEvent {
        pool_address,
        index,
        emissions_per_second
    })
}
```

OS-CTS-SUG-02 | Unnecessary Casting Of Fee Rate

Description

In `compute_swap_step` function, the `fee_rate` variable of type `u64` is unnecessarily casted to `u64`:

```
full_math_u64::mul_div_ceil(amount_in, (fee_rate as u64),
    ↪  (FEE_RATE_DENOMINATOR - fee_rate as u64));
```

Remediation

`as u64` part can be removed

```
full_math_u64::mul_div_ceil(amount_in, fee_rate, (FEE_RATE_DENOMINATOR -
    ↪  fee_rate));
```

Patch

Fixed in f6a3888.

OS-CTS-SUG-03 | Reset Init Price Does Not Validate The Price

Description

Currently, the `reset_init_price` function does not validate the `new_initialize_price`. As a result, the price greater than maximum allowed price or lower than minimum allowed price can be set leaving the pool in the incorrect state.

```
public fun reset_init_price<CoinTypeA, CoinTypeB>(pool_address: address,
    ↪  new_initialize_price: u128) acquires Pool {
    let pool = borrow_global_mut<Pool<CoinTypeA,
    ↪  CoinTypeB>>(pool_address);
    assert!(pool.position_index == 1, EPOOL_LIQUIDITY_IS_NOT_ZERO);
    pool.current_sqrt_price = new_initialize_price;
    pool.current_tick_index =
    ↪  tick_math::get_tick_at_sqrt_price(new_initialize_price);
}
```

Remediation

We suggest comparing the `new_initialize_price` with `MAX_SQRT_PRICE_X64` and `MIN_SQRT_PRICE_X64` line in `factory::create_pool` function.

```
assert!(
    new_initialize_price >= tick_math::min_sqrt_price()
      && new_initialize_price <= tick_math::max_sqrt_price(),
    error::aborted(EINVALID_SQRTPRICE)
);
```

Patch

Fixed in e7a3c12.

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.