Pontem Liquidswap

Audit

Presented by:

OtterSec contact@osec.io

Fineas Silaghi fedex@osec.io

Robert Chen notdeghost@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Pontem engaged OtterSec to perform an assessment of the `liquidswap` program. This assessment was conducted between August 8th and September 2nd, 2022.

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 over to streamline patches and confirm remediation.

We delivered final confirmation of the patches September 5th, 2022.

Key Findings

The following is a summary of the major findings in this audit.

- 8 findings total
- 2 vulnerabilities which could lead to loss of funds
  - OS-PLS-ADV-00: Broken Stable Curve Math
  - OS-PLS-ADV-01: Unstrict Swap Invariant

02 | Scope

The source code was delivered to us in a git repository at github.com/pontem-network/pontem-network-liquidswap. This audit was performed against commit `9cd6904`.

There were a total of one program included in this audit. A brief description of the program is as follows. A full list of program files and hashes can be found in Appendix A.

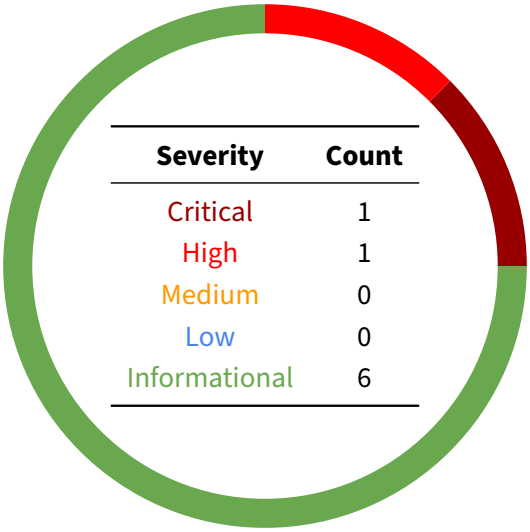
Name	Description
liquidswap	Automated Market Maker protocol that supports both Uncorrelated and Stable curves.

03 | Findings

Overall, we report 8 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.

The below chart displays the findings by severity.



Severity	Count
Critical	1
High	1
Medium	0
Low	0
Informational	6

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.

ID	Severity	Status	Description
OS-PLS-ADV-00	Critical	Resolved	Stable curve math is broken
OS-PLS-ADV-01	High	Resolved	Swaps should use strict comparison for `lp_value`

OS-PLS-ADV-00 [crit] [resolved] | Broken Stable Curve Math

Description

The `liquidity_pool::compute_and_verify_lp_value` function, checks if the lp value is the same before and after a swap. When dealing with a stable curve, the lp value before the swap, is calculated incorrectly.

sources/swap/liquidity_pool.move

```
if (curve_type == STABLE_CURVE) {
    let lp_value_before_swap = stable_curve::lp_value(x_res, x_scale,
    ↪  y_res, y_scale);
    // 100000000 == FEE_SCALE * FEE_SCALE
    lp_value_before_swap = u256::mul(
        lp_value_before_swap,
        u256::from_u128(100000000),
    );
```

The `stable_curve::lp_value` function conducts mathematical operations using decimals with 8 digits precision. Several calculations, contain additional divisions by `10^8`, which are not needed. These unnecessary operations affect the returning value which implicitly affects the returning amount from a swap transaction.

sources/libs/stable_curve.move

```
let u2561e8 = u256::from_u128(ONE_E_8);

[...]

let _b = u256::add(
    u256::div(
        u256::mul(_x, _x),
        u2561e8,
    ),
    u256::div(
        u256::mul(_y, _y),
        u2561e8,
    )
);
```

Proof of Concept

Given the stable curve: `x^3*y + x*y^3`, consider the following test case:

1. x = 500000899318256
2. y = 25000567572582123
3. lp_value = lp_value(x, 1000000, y, 1000000000000)
4. assert!(u256::as_u128(lp_value) == 312508781701599715772530613362069248234)

The returned `lp_value` is incorrect, the correct value should be 312508781701599715772756132553838833260.

Let's also consider the following scenario, derived from an existing test case:

1. A liquidity pool contains 10 BTC and 10000 USDT coins.
2. An attacker swaps a very small amount of btc

Remediation

The issue can be fixed by removing the multiplication between the `lp_value_before_swap` and 100000000 and the unnecessary divisions from the `stable_curve::lp_value`, `stable_curve::d` and `stable_curve::f` functions.

Patch

Resolved in commit 5349aa7 and 637ad72.

OS-PLS-ADV-01 [high] [resolved] | Unstrict Swap Invariant

Description

When dealing with an uncorrelated curve, the program introduced an error by reporting an incorrect swap if the `lp_value` after the swap is strictly smaller than the `lp_value` before the swap. The swap should be valid only when the value after is greater than the value before.

Otherwise, swapping would be able to exploit potential rounding errors, depending on the precision of the relevant curves.

Some napkin math implies that the imprecision is nontrivial. For a token with 8 decimals, the stable swap math would give up to 1,000,000 atomic units of imprecision. This would represent up to 1% of the original token's value, which

sources/swap/liquidity_pool.move

```
assert!(
    lp_value_after_swap_and_fee >= lp_value_before_swap,
    ERR_INCORRECT_SWAP,
);
```

Remediation

The incorrect assert can be fixed by making the condition strictly greater.

sources/libs/stable_curve.move

```
+   let cmp = u256::compare(&lp_value_after_swap_and_fee,
    ↪  &lp_value_before_swap_u256);
+   assert!(cmp == GREATER_THAN, ERR_INCORRECT_SWAP);
```

Patch

Resolved in commit 637ad72.

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 do represent antipatterns and could introduce a vulnerability in the future.

ID	Status	Description
OS-PLS-SUG-00	Resolved	Update event emitted out of sync with the actual update code block.
OS-PLS-SUG-01	Resolved	Deposited event emitted instead of withdrawn event in withdraw function.
OS-PLS-SUG-02	Resolved	Ability to call getter functions in liquidity pool during flashloan lock.

OS-PLS-SUG-00 [resolved] | Unsynchronized Update and Event Emission

Description

The `liquidity_pool::update_oracle` function emits an event of type `OracleUpdatedEvent` once the update is finalized. The problem is that the place from where the event is launched can be reached even when the update has not happened.

sources/libs/stable_curve.move

```
[...]

if (time_elapsed > 0 && x_reserve != 0 && y_reserve != 0) {
    [...]

    pool.last_price_x_cumulative = *&pool.last_price_x_cumulative +
    ↪  last_price_x_cumulative;
    pool.last_price_y_cumulative = *&pool.last_price_y_cumulative +
    ↪  last_price_y_cumulative;
};

pool.last_block_timestamp = block_timestamp;

let events_store = borrow_global_mut<EventsStore<X, Y, LP>>(pool_addr);
event::emit_event(
    &mut events_store.oracle_updated_handle,
    OracleUpdatedEvent<X, Y, LP> {
        last_price_x_cumulative: pool.last_price_x_cumulative,
        last_price_y_cumulative: pool.last_price_y_cumulative,
    });
```

Remediation

Move the code sequence responsible for emitting the event at the end of the if block from above.

Patch

Event emit moved right after the update has happened, resolved in 637ad72.

OS-PLS-SUG-01 [resolved] | Incorrect DaoStorage Event Type

Description

The function `dao_storage::withdraw` emits an incorrect event type, `CoinDepositedEvent`, when the withdraw functionality is completed.

Remediation

Change the event handle to `coin_withdrawn_handle` and the event type to `CoinWithdrawnEvent`.

sources/swap/dao_storage.move

```
    event::emit_event(
+       &mut events_store.coin_withdrawn_handle,
+       CoinWithdrawnEvent<X, Y, LP>{ x_val, y_val }
    );
```

Patch

Issue resolved in 637ad72 by replacing the `CoinDepositedEvent` event type with `CoinWithdrawnEvent` and the handle `coin_deposited_handle` with the proper `coin_withdrawn_handle`.

OS-PLS-SUG-02 [resolved] | Accessible Locked Pool

Description

When a flashloan transaction is performed, the pool that is used is being locked. However, functions such as `get_reserves_size` and `get_cumulative_prices` can still access the pool due to the lack of sufficient checks.

Remediation

Insert at the top of the functions an assert to prevent the functions from being used during flashloans.

sources/swap/liquidity_pool.move

```
public fun get_reserves_size<X, Y, LP>(pool_addr: address): (u64, u64)
acquires LiquidityPool {
    assert_no_emergency();
+   assert_pool_locked<X, Y, LP>(pool_addr);
```

Patch

Resolved in 8eacbb1 by including the suggested `assert` in both functions.

06 | Formal Verification

Here we present recommendations and example specifications for formal verification of contracts.

ID	Description
OS-PLS-VER-00	Recommendations for `liquidity_pool.move`
OS-PLS-VER-01	Recommendations for `emergency.move`
OS-PLS-VER-02	Recommendations for `u256.move`

OS-PLS-VER-00 | Liquidity Pool

Specifications

1. `Flashloan` Data Invariants. Ensure all loans have at least some value.

sources/swap/liquidity_pool.move RUST

```
spec Flashloan {
    invariant x_loan > 0 || y_loan > 0;
}
```

2. No free money theorem. A series of swaps should never arbitrarily increase a pool's token balances. Note that to write a specification for this, you will need to verify the internal math libraries.

sources/swap/liquidity_pool.move RUST

```
spec test_swap {
    ensures !result_1;
}

fun test_swap<X, Y, LP>(
    pool_addr: address,
    x_in: Coin<X>,
    x_out: u64,
    y_in: Coin<Y>,
    y_out: u64,
    x_nxt: u64,
    y_nxt: u64
): (bool, Coin<X>, Coin<Y>) acquires LiquidityPool, EventsStore {
    let x_init = coin::value(&x_in);
    let y_init = coin::value(&y_in);

    let (coin_x, coin_y) = swap<X, Y, LP>(pool_addr, x_in, x_out,
    ↪  y_in, y_out);
    let (coin_x, coin_y) = swap<X, Y, LP>(pool_addr, coin_x, x_nxt,
    ↪  coin_y, y_nxt);

    let free_money = coin::value(&coin_x) >= x_init &&
    ↪  coin::value(&coin_y) > y_init;
    ↪  (free_money, coin_x, coin_y)
}
```

OS-PLS-VER-01 | Emergency

Specifications

1. Explicate when key functions can abort. For example, queries for emergency state should never abort.

sources/swap/emergency.spec.move RUST

```
spec liquidswap::emergency {
  spec is_emergency {
      aborts_if false;
  }

  spec is_disabled {
      aborts_if false;
  }
}
```

It should also be possible to explicate abort conditions of more complex functions. For example, `pause` should abort if and only if there is an emergency, it's already disabled, `Emergency` has already been initialized, or the incorrect signer is passed.

OS-PLS-VER-02 | U256

Specifications

1. Verify that the U256 structs are isomorphic to the natural numbers. Note that because Move Prover numbers are unbounded, it is relatively easy to construct these relationships.

   sources/u256.move RUST

   ```
   spec fun real_val(a: U256): num {
       a.v0
           + (a.v1 << 64)
           + (a.v2 << 128)
           + (a.v3 << 192)
   }

   spec add_test {
       pragma opaque;
       ensures real_val(result) == (real_val(a) + real_val(b));
   }
   ```

   Note how we define a `real_val` function which maps U256 structs onto the native natural number representation. In order for the Move Prover to terminate, you may also need to unroll the loop used in add.

2. Explicate error conditions for arithmetic functions. For example, `add` and `mul` should only abort if the product exceeds the maximum representable U256 value. Similarly, `div` should only abort if the operand is zero.

3. Verify that conversion to and from Move native numeric types with the `as_u64` and `from_u64` functions operate as expected.

A | Program Files

Below are the files in scope for this audit and their corresponding SHA256 hashes.

```
Move.toml                              e976e116aee0da5c319e6186e1406a8c
sources
  libs
    coin_helper.move                   f51ac07f2060348d6878e357cb291d22
    compare.move                       f3d14171fd38329ba1f43dd96b2ca901
    math.move                          e21502bec25aa18ce04dd705fbec3b06
    stable_curve.move                  09b7cc72c287e455c4677d8996197f5f
  swap
    dao_storage.move                   ee17a7d434413253e1b1f7702cc92314
    emergency.move                     094d7f2ba067ec58f013421c9dadd384
    liquidity_pool.move                7125ff40022b3ae57e99c1a0db1561a1
    router.move                        f1b31af7b33fdc16d3af47fe8269fe4f
    scripts.move                       8c4118a6863fb64d61c0a42d8804e99c
  test_helpers
    test_account.move                  576d9389d3459ae9fb0730afb6ee1042
    test_coins.move                    41593b6e772f517887d800390b8ccb79
    test_lp.move                       832bea42230bb33f467ba20acea05087
tests
  coin_helper_tests.move               bd118a7245ededbbcb5434ec76fdd903
  dao_storage_tests.move               200299d1bc5cf114a5950bbb4553986d
  emergency_tests.move                 ca2239bfee7194aef0e9a3317f9da42f
  flashloan_tests.move                 0d1b0a39335f87f5c24aa9f114cf411b
  liquidity_pool_tests.move            53db4a96705ff4f1675e4ca345fdd0eb
  math_tests.move                      92b76a15e2867cc5dfdcde1fabfc014a
  router_tests.move                    7e65feb21f7a503d23fc4fd379e5cc27
  scripts_tests.move                   336b20dc02351843b1909a85cb0f495f
```