Zellic

Liquidswap

Smart Contract Security Assessment

November 3, 2022

Prepared for:

Boris Povod and Igor Demko

Pontem Network

Prepared by:

Daniel Lu and Jacob Farrell

Zellic Inc.

Contents

About Zellic 3

1 Executive Summary 4

2 Introduction 6

2.1 About Liquidswap . . . 6

2.2 Methodology . . . 6

2.3 Scope . . . 7

2.4 Project Overview . . . 8

2.5 Project Timeline . . . 8

3 Detailed Findings 10

3.1 Inaccuracy in `liquidswap::stable_curve` computations . . . 10

3.2 Implicit precision loss in `stable_curve::lp_value` . . . 13

3.3 Incorrect rounding behavior in `router::get_coin_in_with_fees` . . . 15

3.4 `lp_account::retrieve_signer_cap` should be a friend to `liquidity_pool` 18

4 Formal Verification 19

4.1 `liquidswap::math` . . . 19

4.2 `liquidswap::emergency` . . . 20

5 Discussion 21

5.1 Test coverage missing for `stable_curve::get_y` . . . 21

5.2 Tests fail after breaking Aptos change . . . 21

5.3 Outdated comments . . . 21

5.4 Unused struct member in `liquidity_pool::Flashloan` . . . 21

5.5 Flashloan implementation depends on Aptos VM correctness 22
5.6 Liquidswap allows any tokens and pools to be made 22

6 Audit Results **23**

6.1 Disclaimers 23

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Executive Summary

Zellic conducted an audit for Pontem Network from October 10th to October 14th, 2022.

Our general overview of the code is that it was very well-organized and structured. The code coverage is high, with tests included for the vast majority of functions.

We applaud Pontem Network for their attention to detail and diligence in maintaining high code quality standards in the development of Liquidswap. The documentation was adequate, although it could be improved.

Zellic thoroughly reviewed the Liquidswap codebase to find protocol-breaking bugs as defined by the documentation and to find any technical issues outlined in the Methodology section (2.2) of this document.

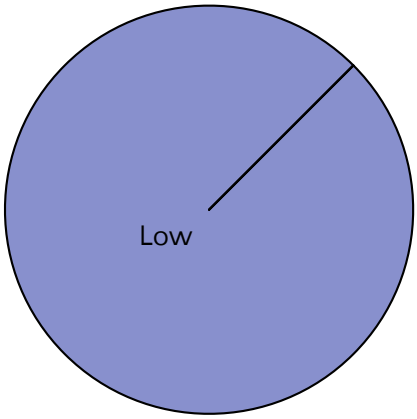
Specifically, taking into account Liquidswap's threat model, we focused heavily on issues that would break core invariants such as the liquidity pool market maker function values.

During our assessment on the scoped Liquidswap contracts, we discovered four findings. Fortunately, no critical issues were found. Of the four findings, all were of low severity.

Additionally, Zellic recorded its notes and observations, as well as sample specifications for Pontem Network's benefit in the Discussion section (5) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	4
Informational	0



2 Introduction

2.1 About Liquidswap

Liquidswap is the first AMM (automated market maker) on the Aptos blockchain, created to enable safe and decentralized token swaps. The protocol uses smart contracts developed by the Pontem Network team, written in the Move language, and published on the Aptos mainnet.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of open-source tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these shallow bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so forth as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, and so forth.

Complex integration risks. Several high-profile exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. We perform a meticulous review of all of the contract's possible external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so forth.

Code maturity. We review for possible improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so forth.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact; we assign it on a case-by-case basis based on our professional judgment and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Similarly, Zellic organizes its reports such that the most important findings come first in the document rather than being ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their importance may differ. This varies based on numerous soft factors, such as our clients' threat models, their business needs, their project timelines, and so forth. We aim to provide useful and actionable advice to our partners that consider their long-term goals rather than simply provide a list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Liquidswap Contracts

Repository https://github.com/pontem-network/liquidswap

Versions `23e5c828c0a34ea9f328e816df2b9c52e187ee1f`

Programs
- ./liquidswap_init/sources/lp_account.move
- ./liquidswap_lp/sources/lp_coin.move
- ./sources/swap/dao_storage.move
- ./sources/swap/router.move
- ./sources/swap/scripts.move
- ./sources/swap/curves.move
- ./sources/swap/liquidity_pool.move
- ./sources/swap/emergency.move
- ./sources/libs/stable_curve.move
- ./sources/libs/math.spec.move
- ./sources/libs/math.move
- ./sources/libs/coin_helper.move

Type Move

Platform Aptos

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-weeks. The assessment was conducted over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-founder
jazzy@zellic.io

Stephen Tong, Co-founder
stephen@zellic.io

The following consultants were engaged to conduct the assessment:

Daniel Lu, Engineer
daniel@zellic.io

Jacob Farrell, Engineer
jacob@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

October 10, 2022	Start of primary review period
October 14, 2022	End of primary review period

3 Detailed Findings

3.1 Inaccuracy in `liquidswap::stable_curve` computations

- **Target**: liquidswap::stable_curve
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

Liquidswap provides peripheral modules for interacting with the protocol. The `liquidswap::stable_curve` module exposes helper functions for computing exchange amounts.

Liquidity pools for correlated coins utilize a different curve. Specifically, if the reserves of two coins are `x` and `y`, then it maintains that `c = x ^ 3 y + y ^ 3 x` must increase across exchanges. To help compute quantities, the internal function `stable_curve::get_y` is used to find `y` given `x` and `c`.

```
fun get_y(x0: U256, xy: U256, y: U256): U256 {
    let i = 0;

    let one_u256 = u256::from_u128(1);

    while (i < 255) {
        let k = f(x0, y);
        let _dy = u256::zero();
        let cmp = u256::compare(&k, &xy);
        if (cmp == 1) {
            _dy = u256::add(
                u256::div(
                    u256::sub(xy, k),
                    d(x0, y),
                ),
                one_u256 // Round up
            );
            y = u256::add(y, _dy);
        } else {
            _dy = u256::div(
                u256::sub(k, xy),
```

```
                d(x0, y),
            );
            y = u256::sub(y, _dy);
        };
        cmp = u256::compare(&_dy, &one_u256);
        if (cmp == 0 || cmp == 1) {
            return y
        };

        i = i + 1;
    };

    y
}
```

This implementation uses Newton's method to iteratively find a `y` value given an initial guess. However, the criteria the function uses to find when it converges will result in slightly unstable outputs: The result of `get_y` differs slightly on different starting conditions. Consider the following:

```
get_y(u256::from_u128(138), u256::from_u128(200000000),
    u256::from_u128(40));
get_y(u256::from_u128(138), u256::from_u128(200000000),
    u256::from_u128(50));
get_y(u256::from_u128(138), u256::from_u128(200000000),
    u256::from_u128(60));
```

While the first and third return `63`, the second returns `64`. The true result should be approximately `62.98`; the initial condition of `50` results in an incorrect result, even if we suppose `get_y` should round upwards.

Impact

The incorrect `get_y` values lead to slightly incorrect calculations by `coin_in` and `coin_out`. The goal of `coin_out` is to return the amount of output a user should receive given reserve states and an input quantity. However, the value it returns can be too low:

```
coin_out(47, 100000000, 100000000, 10000, 15674);
```

This call returns 47, but a user could actually extract 48 coins from the pool while still

increasing the liquidity pool value.

Recommendations

First, the desired behavior of `get_y` should be better documented. At the moment, it is unclear whether it should round up or round down. Based on this decision, the update and stopping criteria for `get_y` should be adjusted. Currently, if the adjustment for `y` is less than or equal to one in a given iteration, the function assumes it has converged and returns.

Remediation

Pontem Network fixed this issue in commit `0b01ed6`

3.2 Implicit precision loss in `stable_curve::lp_value`

- **Target**: liquidswap::stable_curve
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

In `stable_curve::lp_value`, coins with more than eight decimals experience implicit precision loss. The current implementation returns the LP value scaled by `(10 ^ 8) ^ 4` in order to maintain precision across division:

```
public fun lp_value(x_coin: u128, x_scale: u64, y_coin: u128, y_scale:
    u64): U256 {
    let x_u256 = u256::from_u128(x_coin);
    let y_u256 = u256::from_u128(y_coin);
    let u2561e8 = u256::from_u128(ONE_E_8);

    let x_scale_u256 = u256::from_u64(x_scale);
    let y_scale_u256 = u256::from_u64(y_scale);

    let _x = u256::div(
        u256::mul(x_u256, u2561e8),
        x_scale_u256,
    );

    let _y = u256::div(
        u256::mul(y_u256, u2561e8),
        y_scale_u256,
    );

    let _a = u256::mul(_x, _y);

    // ((_x * _x) / 1e18 + (_y * _y) / 1e18)
    let _b = u256::add(
        u256::mul(_x, _x),
        u256::mul(_y, _y),
    );

    u256::mul(_a, _b)
}
```

However, this means that `stable_curve::lp_value` will return inaccurate values when coins have more decimals.

Impact

Loss of precision in LP value calculations can cause fees to be unexpectedly high: Situations where a swap would theoretically increase LP value might fail. This precision loss will also affect the accuracy of router functions.

Recommendations

When coins have more than eight decimals, either rounding should be handled explicitly or they should be disallowed from the protocol.

Another option is to use the numerator `max(x_scale, y_scale)` instead of `10 ^ 8` to mitigate precision loss. Still, coins with unusually high precision would need to be either disallowed or explicitly considered in order to avoid overflow problems.

Remediation

This issue has been acknowledged by Pontem Network.

3.3 Incorrect rounding behavior in `router::get_coin_in_with_fees`

- **Target**: liquidswap::router
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

In the function `router::get_coin_in_with_fees`, the result is rounded up incorrectly for both stable and uncorrelated curves, which can lead to an undue amount being paid in fees.

The formula for rounding up integer division is `(n - 1)/d + 1` for `n > 0`.

```
let coin_in = (stable_curve::coin_in(
    (coin_out as u128),
    scale_out,
    scale_in,
    (reserve_out as u128),
    (reserve_in as u128),
) as u64) + 1;

(coin_in * fee_scale / fee_multiplier) + 1
```

The stable curve branch of `router::get_coin_in_with_fees` does not correctly implement the formula stated above.

```
let coin_in = math::mul_div(
    coin_out, // y
    reserve_in * fee_scale, // rx * 1000
    new_reserves_out // (ry - y) * 997
) + 1;
```

Furthermore, the uncorrelated curve branch also incorrectly implements the formula stated above.

Impact

For certain swap amounts, a user could end up paying more in fees than would be accurate.

Recommendations

In the case of the stable curve branch of `router::get_coin_in_with_fees`, the code should be rewritten to adhere to the rounded up integer division formula.

```
let coin_in = (stable_curve::coin_in(
    (coin_out as u128),
    scale_out,
    scale_in,
    (reserve_out as u128),
    (reserve_in as u128),
) as u64);

let n = coin_in * fee_scale;

if (n > 0) {
    ((n - 1) / fee_multiplier) + 1
} else {
    0
}
```

Likewise, the uncorrelated curve branch also needs a revision.

```
// add to liquidswap::math
public fun mul_div_rounded_up(x: u64, y: u64, z: u64): u64 {
    assert!(z ≠ 0, ERR_DIVIDE_BY_ZERO);
    let n = (x as u128) * (y as u128);
    let r = if (n > 0) {
        ((n - 1) / (z as u128)) + 1
    } else {
        0
    }
    (r as u64)
}

let coin_in = math::mul_div_rounded_up(
    coin_out, // y
    reserve_in * fee_scale, // rx * 1000
    new_reserves_out // (ry - y) * 997
);
```

Remediation

Pontem Network fixed this issue in commit `0b01ed6`

3.4 `lp_account::retrieve_signer_cap` should be a friend to `liquidity_pool`

- **Target**: liquidswap::lp_account
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The function `lp_account::retrieve_signer_cap` can currently be called by any module. If `lp_account::retrieve_signer_cap` is called by a function other than `liquidity_pool::initialize`, then the initialization process of Liquidswap will be unable to move forward.

Impact

The initialization of Liquidswap can be griefed. This will make liquidswap inaccessible to any users.

Recommendations

The function `lp_account::retrieve_signer_cap` needs to be marked as `pub(friend)`, and the module `liquidswap::liquidity_pool` needs to be added as a friend to `liquidswap::lp_account`.

Remediation

This issue has been acknowledged by Pontem Network.

4 Formal Verification

The Move language is designed to support formal verifications against specifications. Currently, there are a number of these written for the `liquidswap::math` module. We encourage further verification of contract functions as well as some improvements to current specifications. Here are some examples.

4.1 `liquidswap::math`

First, the specification for `math::overflow_add` could be improved. The purpose of this function is to add `u128` integers, but allowing for overflow.

```
spec overflow_add {
    ensures result ≤ MAX_U128;
    ensures a + b ≤ MAX_U128 ⟹ result == a + b;
    ensures a + b > MAX_U128 ⟹ result ≠ a + b;
    ensures a + b > MAX_U128 && a < (MAX_U128 - b) ⟹ result == a -
    (MAX_U128 - b) - 1;
    ensures a + b > MAX_U128 && b < (MAX_U128 - a) ⟹ result == b -
    (MAX_U128 - a) - 1;
    ensures a + b ≤ MAX_U128 ⟹ result == a + b;
}
```

However, this does not reflect how the function should work conceptually. Instead, consider the following specification:

```
spec overflow_add {
    /// The function should never abort.
    aborts_if false;

    /// Addition should overflow if the sum exceeds `MAX_U128`
    ensures result == (a + b) % (MAX_U128 + 1);
}
```

This checks that the function cannot abort and makes the desired functionality more clear.

4.2 `liquidswap::emergency`

Another strong application for the prover is in `liquidswap::emergency`. This module provides a way to pause and resume operations as well as a way to disable itself. The `emergency::disable_forever` function is intended to be permanent, and that can actually be proven:

```
spec liquidswap::emergency {
    invariant update old(is_disabled()) ⟹ is_disabled();
}
```

Essentially, this claims that across updates to storage, the emergency module cannot change from disabled to enabled.

5 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

5.1 Test coverage missing for `stable_curve::get_y`

The function `stable_curve::get_y` is currently missing test coverage and is highly susceptible to imprecise behavior because of the nature of its implementation. We recommend tests are added for `stable_curve::get_y` to ensure that its behavior is expected.

5.2 Tests fail after breaking Aptos change

Under Aptos Move version 0.3.5 and the commit hash of this audit, a majority of tests fail to run. However, the tests do pass under an older version of Aptos Move. We recommend making the necessary changes to Liquidswap so that the tests pass on the latest version of Aptos Move.

5.3 Outdated comments

Some comments throughout the project do not reflect the current state of the code.

Comments containing mathematical descriptions such as `// ((_x * _x) / 1e18 + (_y * _y) / 1e18)` in `stable_curve::lp_value` are not accurate and can lead to confusion when reading the project's code.

5.4 Unused struct member in `liquidity_pool::Flashloan`

The `pool_addr: address` member of `liquidity_pool::Flashloan` is unnecessary and unused. This member is likely a remnant of older code, and we recommend it be removed.

5.5 Flashloan implementation depends on Aptos VM correctness

The implementation of flashloans in Liquidswap is correct; however, it depends heavily on the correctness of the Aptos VM and its bytecode verifier.

5.6 Liquidswap allows any tokens and pools to be made

Any user is able to make a pool with any two tokens. Depending on the implementation of the frontend of Liquidswap, this could lead to confusion as pools could be made with fake tokens that share names with real tokens. We recommend an allowlist for pools that are displayed in the frontend.

6 Audit Results

At the time of our audit, the code was not deployed.

During our audit, we discovered four findings. Of these, all were low risk. Pontem Network acknowledged all findings and implemented fixes.

6.1 Disclaimers

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any additional code added to the assessed project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.