Audit Report

Astroport Concentrated Liquidity Pool

v1.0

March 16, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Delphi Labs Ltd. to perform a security audit of the Astroport Concentrated Liquidity Pool.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/astroport-fi/astroport-core
Commit	`042b0768951422099f5d77224c320978cbfa92cc`
Scope	The scope was restricted to - `contracts/pair_concentrated` - `packages/astroport/src/pair_concentrated.rs` - Other imports used from `packages`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Astroport implements an automated, decentralized exchange protocol in the Cosmos Ecosystem. This audit only covers Astroport's concentrated liquidity pool in `contracts/pair_concentrated`.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Low	No technical documentation or math formula explanations were available.
Test coverage	High	`cargo tarpaulin` reports 90.05% code coverage for `pair_concentrated` folder. It is worth to mention that whole `astroport-pair-concentrated` package returns 62.88% code coverage.

Summary of Findings

No	Description	Severity	Status
1	The moving average time limit can be incorrectly set to zero	Minor	Resolved
2	Factory contract's ownership transfer freezes pair contracts	Minor	Resolved
3	Unnecessary nested loops when initializing cumulative prices	Informational	Resolved
4	Incorrect error message for CW20 token swap	Informational	Resolved
5	Unsafe math usage could lead to division by zero error	Informational	Resolved
6	Unclear error message for a single-sided initial deposit	Informational	Resolved
7	Unnecessary tax deductions are performed	Informational	Acknowledged
8	Validation can be performed earlier	Informational	Resolved
9	Mismatch between comments and code	Informational	Resolved
10	Arithmetic operations can be optimized	Informational	Resolved
11	Duplicated fee address checks during `Swap`	Informational	Acknowledged

Detailed Findings

1. The moving average time limit can be incorrectly set to zero

Severity: Minor

In `contracts/pair_concentrated/src/const.rs:49`, the boundaries of the moving average time limit are defined as an including range between zero seconds and a week.

However, since `ma_half_time` is used as the denominator in `contracts/pair_concentrated/src/state.rs:304` and it can be set to zero, it could cause a division by zero error.

Recommendation

We recommend altering the allowed range of the `ma_half_time` parameter to be between one second and one week.

Status: Resolved

2. Factory contract's ownership transfer freezes pair contracts

Severity: Minor

During the handling of `ExecuteMsg`, the `migration_check` function defined in `contracts/pair_concentrated/src/contract.rs:215-218` ensures that there is no pending ownership transfer in the factory contract.

When the factory's new owner accepts their role by executing the `ClaimOwnership` message, all registered pairs are added to the `PAIRS_TO_MIGRATE` vector. When a pair is in this vector, the `migration_check` function returns an error and all the pair operations are disabled. The factory's new owner can then reactivate pairs by executing the `MarkAsMigrated` message.

This implies that all pair operations will fail while even just one pair is not yet migrated. Such a situation may be caused by a migration issue or a compromise of the factory contract.

Recommendation

We recommend segregating pairs and factory ownership to prevent a freeze of a pair's critical operations while contract ownership of the factory is transferred.

Status: Resolved

3. Unnecessary nested loops when initializing cumulative prices

Severity: Informational

When instantiating the contract, the cumulative price vector is initialized with default values in `contracts/pair_concentrated/src/contract.rs:80-84`. Within two nested loops, it is then verified that assets are not duplicated before pushing them to the relevant vector.

As there can only be two assets, the usage of nested loops is unnecessary, increases code complexity, and reduces readability.

Recommendation

We recommend unrolling the nested loops in `contacts/pair_concentrated/src/contracts.rs:80-84` and pushing the values to the vector independently.

Status: Resolved

4. Incorrect error message for CW20 token swap

Severity: Informational

When a `Swap` operation is performed in `contracts/pair_concentrated/src/contract.rs:243-245`, `offer_asset` is validated to not be a CW20 token. If it is, an incorrect `ContractError::Unauthorized` error is returned.

This provides a poor user experience as it may be difficult for users to identify the source of the error when executing a swap transaction with a non-native asset.

Recommendation

We recommend substituting the error with a more appropriate one such as `ContractError::Cw20DirectSwap`.

Status: Resolved

5. Unsafe math usage could lead to division by zero error

Severity: Informational

The `query_compute_d` function is used to compute the current pool `D` value by invoking `Calc_d` and `newton_d` functions.

However, if the pool has no liquidity yet (i. e. before the first use of `provide_liquidity`), a division by zero error can occur in `contracts/pair_concentrated/math/math-decimal.rs:65`.

A test case showcasing this issue is provided in the Appendix.

Recommendation

We recommend using safe math libraries and functions like `checked_sub` in order to handle errors gracefully.

Status: Resolved

6. Unclear error message for a single-sided initial deposit

Severity: Informational

The contract does not allow the first deposit to be a single-sided one by returning an `InvalidZeroAmount` error if only one asset is provided in `contracts/pair_concentrated/src/contract.rs:414-415`.

However, this error message is not self explanatory and may prevent a user from understanding why their liquidity provision is not successful.

Recommendation

We recommend returning a more informationive error message.

Status: Resolved

7. Unnecessary tax deductions are performed

Severity: Informational

When performing native tokens transfers tax deductions are performed in `packages/astroport/src/asset.rs:92`.

However, the function to compute tax always returns zero instead of performing actual calculations.

Recommendation

We recommend removing the tax deductions applied to native token transfers as they are unnecessary.

Status: Acknowledged

8. Validation can be performed earlier

Severity: Informational

In `contracts/pair_concentrated/src/contract.rs:59`, there is a check that prevents the code from running if the amount of assets provided in `msg.msg_info` is different waom two.

Since this information is always available, this check can be done before calling `check_asset_infos` and potentially abort the execution early to save resources.

Similarly, in `contracts/pair_concentrated/src/contract.rs:377`, the check for invalid number of assets can be done before querying the pools and running `check_assets`.

Recommendation

We recommend performing the validation on the number of provided assets at the beginning of each function.

Status: Resolved

9. Mismatch between comments and code

Severity: Informational

In `contracts/pair_concentrated/src/state.rs:261`, the comments for the `get_amp_gamma` function state that the parameters stored in the `future` parameters will be returned in case the block time is greater than the value in `self.future_time`.

However, in `contracts/pair_concentrated/src/state.rs:264`, the future parameters are also returned if the block time is equal to the `self.future_time`.

Recommendation

We recommend correcting the comments to match the actual implementation.

Status: Resolved

10. Arithmetic operations can be optimized

Severity: Informational

In `contracts/pair_concentrated/src/utils.rs:291`, in order to get the index of `ask_ind`, a subtraction is performed. Given that `ask_ind` can be either 0 or 1, this is not the most efficient way to compute a bit flip.

In `contracts/pair_concentrated/src/utils.rs:415`, a sum of two differences is calculated to get to the the `deviation`, which is defined as the sum of the distances between the value of two deposits and the average deposit value. Given that there are only two elements, this computation is not efficient.

Recommendation

We recommend using `xor` for flipping a single bit instead of subtracting. Additionally, we recommend replacing `deposits[0].diff(avg) + deposits[1].diff(avg);` with `2 * deposits[0].diff(avg);` where `2` is a scalar value of the same type as `deposits`.

Status: Resolved

11. Duplicated fee address checks during `Swap`

Severity: Informational

During the handling of `Swap` messages, there are multiple checks to ensure that the `fee_address` is provided.

In `contracts/pair_concentrated/src/contract.rs:747`, an explicit check is made but it is not required since the check in line `748` depends on the logic in line `702` that already validated the existence of the fee address.

Recommendation

We recommend removing the duplicated fee address check.

Status: Acknowledged

Appendix: Test Cases

1. Test case for [“Unsafe math usage could lead to division by zero error”]

```
#[test]
fn query_d_test() {
let owner = Addr::unchecked("owner");

let test_coins = vec![TestCoin::native("uusd"), TestCoin::cw20("USDX")];

let params = ConcentratedPoolParams {
    amp: f64_to_dec(40f64),
    gamma: f64_to_dec(0.000145),
    mid_fee: f64_to_dec(0.0026),
    out_fee: f64_to_dec(0.0045),
    fee_gamma: f64_to_dec(0.00023),
    repeg_profit_threshold: f64_to_dec(0.000002),
    min_price_scale_delta: f64_to_dec(0.000146),
    price_scale: Decimal::one(),
    ma_half_time: 600,
};

// create pair with test_coins
let mut helper = Helper::new(&owner, test_coins.clone(),
params).unwrap();

let assets = vec![
    helper.assets[&test_coins[0]].with_balance(100_000_000_000000u128),
    helper.assets[&test_coins[1]].with_balance(100_000_000_000000u128),
];

// query current pool D value before providing any liquidity
let d = helper.query_d().unwrap();
assert_eq!(d, Decimal256::from_str("200000000").unwrap());
}
```