Audit Report

Astroport on Osmosis

v1.0

January 29, 2024

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **3**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 6
Functionality Overview 6
How to Read This Report **7**
Code Quality Criteria 8
Summary of Findings **9**
Detailed Findings **10**
1. Double refunds on excess offer assets allow draining pools 10
2. Attackers can swap any token and drain funds from pools 10
3. Internal price is updated regardless of trade size 10
4. msg.whitelist_code_id field is ignored during instantiation 11
5. Updated configuration values are not emitted 11
6. General code improvements 12
7. before_swap_check validation is not implemented in SwapExactAmountOut and reverse simulation queries 12
Appendix **14**
1. Command to compute test coverage 14

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

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 Astroport Protocol Foundation to perform a security audit of Astroport on Osmosis smart contracts.

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-on-osmosis
Commit	`b4c1ce1b8639d46e58c726c1cbc152ade19b3101`
Scope	All contracts and the `/packages` folder were in scope.
Fixes verified at commit	`1e759d295170de45b93e296b84364d964ebc8dc8` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

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

The submitted codebase contains modifications to Astroport's modules, enabling pools to be created through the `cosmwasmpool` module on the Osmosis chain.

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-High	The complexity increases due to integration with Osmosis's `cosmwasmpool` module.
Code readability and clarity	Medium-High	-
Level of documentation	Medium	Documentation is available at `contracts/pair_concentrated/src/README.md`.
Test coverage	High	`cargo tarpaulin` reports a test coverage of `95.38%`. For more information on how we compute the coverage, please see this section in the appendix.

Summary of Findings

No	Description	Severity	Status
1	Double refunds on excess offer assets allow draining pools	Critical	Resolved
2	Attackers can swap any token and drain funds from pools	Critical	Resolved
3	Internal price is updated regardless of trade size	Minor	Resolved
4	`msg.whitelist_code_id` field is ignored during instantiation	Informational	Resolved
5	Updated configuration values are not emitted	Informational	Resolved
6	General code improvements	Informational	Resolved
7	`before_swap_check` validation is not implemented in `SwapExactAmountOut` and reverse simulation queries	Informational	Resolved

Detailed Findings

1. Double refunds on excess offer assets allow draining pools

Severity: Critical

In `contracts/pair_concentrated/src/sudo.rs:238-249`, the `swap_exact_amount_out` function computes the excess offer asset amount and refunds it to the sender. This is problematic because the `cosmwasmpool` module also refunds the excess amount to the sender, resulting in double refunds.

Consequently, an attacker can repeatedly call `SwapExactAmountOut` on the `poolmanager` module with large `token_in_max_amount` and low `token_out` values to drain funds from the pool, causing a loss of funds for liquidity providers.

Recommendation

We recommend removing lines `238-249` to prevent duplicate refunds.

Status: Resolved

2. Attackers can swap any token and drain funds from pools

Severity: Critical

In `contracts/pair_concentrated/src/sudo.rs:78`, the `swap_exact_amount_out` function does not validate that `token_in_denom` is one of the pool assets. This is problematic because users can provide fake or low-value native tokens to swap for other assets in the pool.

Consequently, an attacker can create and mint a large amount of a token factory denom and call `SwapExactAmountOut` on the `poolmanager` module to trade any token for pool assets, causing a loss of funds for liquidity providers.

Recommendation

We recommend validating `token_in_denom` to be one of the pool assets.

Status: Resolved

3. Internal price is updated regardless of trade size

Severity: Minor

In `contracts/pair_concentrated/src/sudo.rs:165-171`, the `swap_exact_amount_out` function updates the internal oracle price with the last price.

This is problematic because the price update should only occur if the trade sizes are small, as seen in `contracts/pair_concentrated/src/contract.rs:733-745`.

Consequently, the internal oracle price will be updated incorrectly during a `SwapExactAmountOut` sudo call, causing an incorrect price to be used.

Recommendation

We recommend implementing the condition to update the price only if the trade size is small, similar to `contracts/pair_concentrated/src/contract.rs:735-736`.

Status: Resolved

4. `msg.whitelist_code_id` field is ignored during instantiation

Severity: Informational

In `contracts/factory/src/contract.rs:59`, the `whitelist_code_id` is set to zero. This is problematic because `msg.whitelist_code_id` is provided in the `InstantiateMsg` struct, but not used.

Consequently, specifying `msg.whitelist_code_id` during contract instantiation will not be reflected in the `whitelist_code_id` configuration.

We classify this issue as informational because the `whitelist_code_id` configuration is not used across the protocol.

Recommendation

We recommend replacing line `59` with `msg.whitelist_code_id`.

Status: Resolved

5. Updated configuration values are not emitted

Severity: Informational

The `execute_update_config` function in `contracts/factory/src/contract.rs:215` emits only the action attribute. For state-changing functions, It is best practice to emit events and attributes to support off-chain event listeners and blockchain indexers.

Recommendation

We recommend emitting all of the updated attribute values in the `execute_update_config` function.

Status: Resolved

6. General code improvements

Severity: Informational

In several instances of the codebase, the code quality and readability can be improved:

- `osmosis-test-tube` is outdated and contains dependencies with known vulnerabilities (RUSTSEC-2022-0093, RUSTSEC-2023-0052). Consider updating the package to the latest version.
- The `auto_stake` argument's comment was not removed in `contracts/pair_concentrated/src/contract.rs:297`, and the `receiver` argument's comment was not removed in `contracts/pair_concentrated/src/contract.rs:510`. Consider removing them.
- `contracts/pair_concentrated/src/contract.rs:601-604` and `contracts/factory/src/contract.rs:179-184` lack documentation for the arguments. This is inconsistent with the rest of the functions in the codebase. Consider adding them.
- The error in `contracts/factory/src/error.rs:44` is not used. Consider removing it.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Resolved

7. `before_swap_check` validation is not implemented in `SwapExactAmountOut` and reverse simulation queries

Severity: Informational

In `contracts/pair_concentrated/src/contract.rs:693` and `contracts/pair_concentrated/src/queries.rs:228`, the `before_swap_check` function is implemented in the `internal_swap` (when `SwapExactAmountIn` is called) and `query_simulation` functions. The `before_swap_check` function is used to ensure the offer amount is not zero, and the pool assets are not empty.

However, the validation is not implemented on the reverse swap mechanism in the `swap_exact_amount_out` (when `SwapExactAmountOut` is called) and `query_reverse_simulation` functions.

Implementing the validation helps to save gas fees as the error will be triggered first during a reverse simulation query, preventing users from initiating a failed transaction.

Recommendation

We recommend implementing the `before_swap_check` validation on `swap_exact_amount_out` and `query_reverse_simulation` functions.

Status: Resolved

Appendix

1. Command to compute test coverage

The client shared the following command to compute the test coverage, which is executed in the workspace root directory:

```
cargo tarpaulin --target-dir target/tarpaulin_build --skip-clean --exclude-files
\*tests\*.rs target\*.rs \  -e astroport-osmo-e2e-tests --out Html --workspace
```