Audit Report

Astroport Fee Sharing

v1.0

October 17, 2024

Table of Contents

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 WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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 Fee Sharing.

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	`6c753c41e72ef5ea60bfa3363657bbc6e775f72d`
Scope	The scope was restricted to the changes in the commit above.

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 changes reviewed during this audit cover the Astroport Fee Sharing implementation based on the ARC-75 proposal, allowing a portion of the swap fees to be shared with the configured recipient address.

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	Low-Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	`cargo tarpaulin` reports a test coverage of `85.73%`.

Summary of Findings

No	Description	Severity	Status
1	Migrate handler does not support `fee_share` migration for all contract versions	Minor	Acknowledged
2	Fee share implementation is not reflected in the `convert_config` function	Minor	Acknowledged
3	Errors during the fee sharing transfer revert swaps	Informational	Acknowledged
4	Edge case scenarios and functional requirements are not covered	Informational	Acknowledged
5	General code improvements	Informational	Acknowledged

Detailed Findings

1. Migrate handler does not support `fee_share` migration for all contract versions

Severity: Minor

The `fee_share` storage field is introduced in `contracts/pair/src/state.rs:25`, `contracts/pair_concentrated/src/state.rs:43`, and `contracts/pair_stable/src/state.rs:32`. This change requires a state migration for existing contracts deployed with an earlier version. However, this migration is not implemented for all contract versions.

In `contracts/pair/src/contract.rs:1336`, the `migrate` function does not migrate the fee share state for the `1.3.0` and `1.3.1` contract versions. This is problematic because migrating from both versions will cause the `Config` storage state to be unable to serialize properly due to the required `fee_share` field being unimplemented.

Likewise, the pair concentrated contract does not implement the required migration for versions `1.2.4`, `2.0.3`, and `2.0.4`, and the pair stable contract does not implement for versions `3.0.0` and `3.1.0`, as seen in `contracts/pair_concentrated/src/contract.rs:954-957` and `contracts/pair_stable/src/contract.rs:1071`.

We classify this issue as minor because the contract migration admin can recover from this issue by implementing the state migration and initiating another contract upgrade.

Recommendation

We recommend implementing the `fee_share` field migration for the contract versions mentioned above.

Status: Acknowledged

2. Fee share implementation is not reflected in the `convert_config` function

Severity: Minor

In `contracts/periphery/liquidity_manager/src/utils.rs:273`, the `convert_config` function sets the `fee_share` to `None`. This is incorrect because if the pair stable contract implements a fee-sharing mechanism, the fee-sharing status will be incorrectly represented.

We classify this issue as minor because future versions may be affected by the incorrect values in the `convert_config` function.

Recommendation

We recommend using `compat_config.fee_share` instead.

Status: Acknowledged

3. Errors during the fee sharing transfer revert swaps

Severity: Informational

The fee share logic implemented for the `pair`, `pair_concentrated`, and `pair_stable` contracts dispatch a transfer to the `fee_share.recipient` with the appropriate fee share amount. This is executed with a Cosmos message as seen in `contracts/pair/src/contract.rs:715`. If the token transfers to the recipient errors, the entire swap transaction will revert.

This error state can be introduced either unintentionally or intentionally. For example, if the recipient is in the token factory denom blacklist, it would cause an error to be thrown by the `BlockBeforeSend` hook. Also, a CW20 contract could be configured to purposefully error to revert all swap transactions.

We classify this issue as informational due to its extremely low likelihood of occurrence. The Astroport community is expected to heavily scrutinize fee share recipients and have little incentive to cause such a disruption to the protocol due to potential reputation damage.

Additionally, Astroport governance can take immediate action to remediate the situation by removing the fee share from the affected contract.

Recommendation

We recommend dispatching the fee transfer to the fee share recipient in a `ReplyOn::Error` sub-message and handling the error case such that the swap operation is not reverted.

Status: Acknowledged

4. Edge case scenarios and functional requirements are not covered

Severity: Informational

The test cases for the codebase are inconsistent across the contracts despite targeting the same mechanism. Also, they do not cover certain edge cases, for instance:

- Edge cases where `0` or `MAX_FEE_SHARE_BPS` is set as fee-sharing value.

- Updating fee-sharing value.

Recommendation

We recommend applying the following recommendations:

- Implement a test verifying that if the sharing value is already set, it can be overwritten by the next correct fee-sharing value.
- Implement the following tests in `contracts/pair_concentrated/tests/pair_concentrated_integration.rs`:
  - A test case for a `0` fee share.
  - A test checking that `fee_share` of `0` and `MAX_FEE_SHARE_BPS` can be set.
  - A test case verifying that fee share can be updated.
  - Ensure that all test cases contain a comment or description (e.g., `contracts/pair/tests/integration.rs:1632` does not have one, while the rest do).
- Implement a test setting the `fee_share` to `MAX_FEE_SHARE_BPS` in `contracts/pair/tests/integration.rs`.

Status: Acknowledged

5. General code improvements

Severity: Informational

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

- The comments in `contracts/pair_concentrated/tests/pair_concentrated_integration.rs:1757` and `contracts/pair_concentrated/tests/pair_concentrated_simulation.rs:101` mention the fee value as 5%, while the implementation sets it to 10%.
- `contracts/pair/src/contract.rs:703` uses a hardcoded value of 10000, which can be converted into a constant.
- The comment in `contracts/pair_concentrated/tests/pair_concentrated_integration.rs:1744` mentions setting the `fee_share` to max+1, but it is set to 0 in the implementation.

Recommendation

We recommend following the recommendations mentioned above.

Status: Acknowledged