Audit Report

Astroport Concentrated Liquidity Pool with Injective Orderbook Integration

v1.0

July 13, 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 with Injective Orderbook Integration.

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	`e72acd507e54fb25e8396bf990890d9e32af46f0`
Scope	Only the following directories were in scope of this audit: - `contracts/pair_concentrated_inj` - `contracts/periphery/fee_granter` - `packages/astroport/src/fee_granter.rs` - `packages/astroport/src/injective_ext.rs` - `packages/astroport/src/pair_concentrated_inj.rs` - `packages/circular_buffer`
Fixes verified at commit	`53a89ae759d8c53c93b3b9dba7c91125e658de8d`

	Note that changes to the codebase beyond fixes after the initial audit have not been in 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

This audit covers the integration of Astroport's Passive Concentrated Liquidity (PCL) pools with the Injective exchange. The integration enables Astroport's PCL pools to provide liquidity not only as an AMM but also as a maker in the Injective orderbook.

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	Medium-High	Most functions are well documented with clear and concise comments.
Test coverage	Medium-High	The following test coverage was reported: `pair_concentrated_inj`: 86% `fee_granter`: 85% `circular_buffer`: 89%

Summary of Findings

No	Description	Severity	Status
1	Incorrect validation of updated administrators may lead to exceeding the `MAX_ADMINS` limit	Minor	Resolved
2	Inconsistent admin duplicate validation	Minor	Resolved
3	Lack of denom validation	Minor	Resolved
4	Variable names and comments differ from the implementation	Minor	Resolved
5	Lack of orderbook state validation upon update	Minor	Resolved
6	Market tick size parameters cannot be updated	Minor	Resolved
7	Inconsistent contract version between `Cargo.toml` and migration	Minor	Resolved
8	No attributes are added to some message handlers' responses	Informational	Resolved
9	Unnecessary duplicate asset validation	Informational	Resolved
10	Confusing error message during params update	Informational	Resolved
11	Overflow checks not enabled for release profile	Informational	Acknowledged
12	Misleading behavior in case of unexpected `match` branch	Informational	Resolved
13	Duplicated code	Informational	Resolved
14	Unhandled zero-amount transfer	Informational	Resolved
15	`TODO` comments in the codebase	Informational	Resolved

Detailed Findings

1. Incorrect validation of updated administrators may lead to exceeding the `MAX_ADMINS` limit

Severity: Minor

The `UpdateAdmins` message is used to update the list of administrators, which are, for example, authorized to revoke grants. The message allows for the addition as well as the removal of addresses. The maximum number of administrators, defined in the `MAX_ADMINS` constant, is 2. In the `update_admins` function, there is a code fragment that verifies whether the added administrators sent as a vector within the `UpdateAdmins` message do not have more elements than `MAX_ADMINS`.

However, this validation is performed only on the `add_admins` parameter, not on the current administrators in `admins` plus the added administrators in `add_admins`. Consequently, if there are any number of existing administrators stored, their number can be extended by a maximum of 2 per `UpdateAdmins` message call.

This leads to a bypass of the `MAX_ADMINS` limit. We classify this issue as minor since the affected functionality is privileged.

Please see the `test_incorrect_validation` test case to reproduce the issue.

Recommendation

We recommend performing validation on the sum of current administrators and those added.

Status: Resolved

2. Inconsistent admin duplicate validation

Severity: Minor

The `free-granter` contract performs validation of the submitted `admin` addresses in `contracts/periphery/fee_granter/src/state.rs:15-23`. However, no checks on address duplication are performed in this function. While the update function performs its own separated checks, this is not the case with the instantiation function.

As there is a maximum of two admins, duplicate addresses would limit operations.

tWe classify this issue as minor since admin addresses can be updated to remove duplicated addresses.

Recommendation

We recommend adding a check for duplicates within the `validate_admin` function.

Status: Resolved

3. Lack of denom validation

Severity: Minor

The `free-granter` contract lacks validation of the `CONFIG.gas_denom` parameter upon instantiation in `contracts/periphery/fee_granter/src/contract.rs:34`. An invalid denom will render the contract useless. Please note that as this parameter is not updatable the incorrect state will not be recoverable, requiring a new deployment.

Recommendation

We recommend checking that the submitted string is a valid denom.

Status: Resolved

4. Variable names and comments differ from the implementation

Severity: Minor

It was noticed that in two cases the comments describing the code as well as the names of the variables are not reflected in the contract code, which decreases maintainability and point to potential bugs in the implementation.

The first case concerns the `update_capacity` function in `packages/circular_buffer/src/lib.rs:125`, used to update buffer capacity. In line `129`, it checks if the value of the `key` is less than `capacity`. If all values after iterating through `self.precommit_buffer` are `True`, the `can_reduce` variable will evaluate to `True` too.

There is a comment in line `122` stating that if the precommit buffer contains keys greater than the new capacity, an error should be raised. However, when they are equal to the new capacity, the error will also be raised, as `can_reduce` becomes `False`.

The second case concerns the `accumulate_swap_sizes` function in `contracts/pair_concentrated_inj/src/utils.rs:394`. Line `443` checks if `ob_state.ready` is `False` and if `buffer.head()` is greater than `ob_state.min_trades_to_avg`. If it is, `ob_state.ready` will be set to `True`.

However, based on the variable name `min_trades_to_avg`, if the minimum is reached, the order book should be enabled, but now, one more trade is required.

Recommendation

We recommend considering the "equal" scenario in both cases and adapting the function code accordingly.

Status: Resolved

5. Lack of orderbook state validation upon update

Severity: Minor

The `pairs-concentrated-inj` contract successfully validates the fields of each `OrderBookState` when a new one is created in `contracts/pair_concentrated_inj/src/state.rs:118-123`, as done during instantiation. However, in `contracts/pair_concentrated_inj/src/contract.rs:897`, the `save` function is used instead, which does not execute `validate_program!` over the new `orders_number` value.

Therefore, any value submitted as `orders_number` is accepted when modifying the `ob_config` storage variable through `update_config`.

Recommendation

We recommend enforcing the same validation upon configuration update as in instantiation.

Status: Resolved

6. Market tick size parameters cannot be updated

Severity: Minor

When creating a new pool, the tick sizes are set in the `set_ticks` function in `contracts/pair_concentrated_inj/src/orderbook/state.rs:158:169` by querying the market found in the Injective exchange model.

Injective's market parameters can be updated via a governance action, but, the variables can't be updated in the pool contract. This could lead to state inconsistencies that prevent the pool from successfully submitting orders to the exchange.

Recommendation

We recommend allowing admins to call `set_ticks` through the addition of another `ConcentratedObPoolUpdateParams` enum variant in the `update_config` function in `contracts/pair_concentrated_inj/src/contract.rs:867-904`.

Status: Resolved

7. Inconsistent contract version between `Cargo.toml` and migration

Severity: Minor

The contract version is set during instantiation and then again during migration using the version defined in `Cargo.toml`.

In the current `migrate` function, the expected migration version is defined as `1.2.0`. However, post-migration the new contract version is set using the value `1.0.0` as defined in `contracts/pair_concentrated_inj/Cargo.toml:3`.

If executed, the semantic version would no longer be correctly applied and may prevent future migrations from executing successfully.

Recommendation

We recommend updating the contract version in `contracts/pair_concentrated_inj/Cargo.toml:3` to the correct next semantic version.

Status: Resolved

8. No attributes are added to some message handlers' responses

Severity: Informational

In some instances, no attributes are added to the response of a contract message handler.

This could impact off-chain services that try to monitor actions performed by the protocol.

Such instances can be found in:

- `contracts/pair_concentrated_inj/src/contract.rs:903` and `928`
- `contracts/pair_concentrated_inj/src/orderbook/sudo.rs:42`, `202`, and `204`
- `contracts/periphery/fee_granter/src/contract.rs:38`

Recommendation

We recommend adding relevant information as attributes to responses so the performed action and outcome can be clearly identified by off-chain services.

Status: Resolved

9. Unnecessary duplicate asset validation

Severity: Informational

The `calc_market_ids` function in `contracts/pair_concentrated_inj/src/orderbook/utils.rs:40` validates `asset_infos`, verifying, among other things, their length and type in lines `41-45`. This function is exclusively called from the `validate` method in `contracts/pair_concentrated_inj/src/orderbook/state.rs:111`, which is called by the `new` method in line `62`.

A new `OrderbookState` structure, which the above method applies to, is created only in two places – in the `instantiate` function of the contract and during its `migration`. In the first case, before calling `new`, the length of `asset_infos` is already validated by a code fragment identical to that in the `calc_market_ids` function. In the second case, `migrate` operates on parameters derived from `CONFIG`, which have been previously validated by the protocol.

Consequently, the validation in `contracts/pair_concentrated_inj/src/orderbook/utils.rs:41-45` is redundant.

Recommendation

We recommend removing the unnecessary `asset_infos` length check fragment in the `calc_market_ids` function.

Status: Resolved

10. Confusing error message during params update

Severity: Informational

The `update_params` function is used to update the parameters characterizing the pool and to perform validation on these parameters. In `contracts/pair_concentrated_inj/src/state.rs:86`, it is checked whether `out_fee` is greater than `self.mid_fee`. If not, an error is returned. However, the message of this error is not descriptive of the situation, making it impossible for users to understand the reason for the error.

Recommendation

We recommend making the error message more descriptive, providing hints to users on how to resolve the problem.

Status: Resolved

11. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/pair_concentrated_inj/Cargo.toml`
- `contracts/periphery/fee_granter/Cargo.toml`

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

12. Misleading behavior in case of unexpected `match` branch

Severity: Informational

The `process_cumulative_trade` function processes filled orders as one cumulative trade. To do that, it compares the values of `ob_state.last_balances[0].amount` and `subacc_balances[0].amount` and, depending on the result, performs certain operations.

However, if the values are equal, the branch `Ordering::Equal` will be reached during pattern-matching, which includes a comment stating that reaching this point is theoretically impossible and should never happen. The function then returns `Ok(messages)`, where `messages` is an empty vector.

Considering the above comment and the fact that this line should never be reached, returning without an error goes against best practices and does not allow the calling context to properly handle the error or return it.

Recommendation

We recommend returning an error for execution paths that should never be reached.

Status: Resolved

13. Duplicated code

Severity: Informational

The `push_many` function of the `circular-buffer` package executes the same code as the `push` function in `packages/circular_buffer/src/lib.rs:147-150` but inside a `for` loop. Instead of calling the `push` function inside the `for` loop, the code is replicated.

Although not a security issue, duplicated code reduces maintainability and can be error-prone.

Recommendation

We recommend calling `push` within `push_many` instead of duplicating code.

Status: Resolved

14. Unhandled zero-amount transfer

Severity: Informational

Using the `TransferCoins` message, an admin or an owner can transfer tokens from the contract. The funds are transferred from the contract to the recipient or sender using the `transfer_coins` function defined in `contracts/periphery/fee-granter/src/contract.rs:197`.

However, this message does not validate whether the amount is greater than zero. If the caller attempts to transfer zero tokens, an error will be returned by the Cosmos SDK, as `BankMsg::Send` does not support zero-amount transfers.

Recommendation

We recommend adding a validation step when transferring funds to ensure the amount is greater than zero.

Status: Resolved

15. TODO comments in the codebase

Severity: Informational

In `packages/circular_buffer/src/lib.rs:137` a TODO comment was found. TODO comments indicate that the codebase is not ready for production deployment.

Recommendation

We recommend resolving and removing TODO comments.

Status: Resolved

Appendix A: Test Cases

1. **Test case for "Incorrect validation of updated administrators leads to exceeding the number of MAX_ADMINS"**

The test case should pass if the `MAX_ADMINS` is exceeded.

```
fn test_update_admins() {
    let owner = Addr::unchecked("owner");
    let admin = Addr::unchecked("admin");
    let mut app = App::new(|router, _, store| {
        router
            .bank
            .init_balance(store, &owner, coins(1000000, GAS_DENOM))
            .unwrap();
    });

    let fee_granter_code_id = app.store_code(fee_granter_contract());
    let fee_granter = app
        .instantiate_contract(
            fee_granter_code_id,
            owner.clone(),
            &InstantiateMsg {
                owner: owner.to_string(),
                admins: vec![admin.to_string()],
                gas_denom: GAS_DENOM.to_string(),
            },
            &[],
            "Test contract",
            None,
        )
        .unwrap();

    app.send_tokens(owner.clone(), admin.clone(), &coins(10, GAS_DENOM))
        .unwrap();
    app.send_tokens(owner.clone(), fee_granter.clone(), &coins(5, GAS_DENOM))
        .unwrap();

    // Admin can only create, revoke grants and transfer coins
    app.execute_contract(
        admin.clone(),
        fee_granter.clone(),
        &ExecuteMsg::TransferCoins {
            amount: 5u128.into(),
            receiver: None,
        },
        &[],
    )
    .unwrap();
```

```
    app.execute_contract(
        owner.clone(),
        fee_granter.clone(),
        &ExecuteMsg::UpdateAdmins {
            add: vec!["admin2".to_string(),"admin3".to_string()],
            remove: vec![],
        },
        &[],
    )
    .unwrap();

    let res: Config = app
        .wrap()
        .query_wasm_smart(&fee_granter, &QueryMsg::Config {})
        .unwrap();

    assert_eq!(res.admins.len() > MAX_ADMINS, true);
}
```