Astroport Concentrated Liquidity Pool Audit Report

Prepared for Astroport, 23rd February 2023

Table of Contents

Introduction

SCV was engaged by Delphi Labs to assist in identifying security threats and vulnerabilities that have the potential to affect their security posture. Additionally, SCV will assist the team in understanding the risks and identifying potential mitigations.

Scope

SCV performed the security assessment on the following codebase:

- Astroport Concentrated Liquidity Pool -
    - https://github.com/astroport-fi/astroport-core/pull/333
- Code hash: *cd203084825fdeb483093139afcb2197ae2960d8*

Remediations were applied by Delphi Labs team and reviewed by SCV on the following pull request:

- https://github.com/astroport-fi/astroport-core/pull/342
- Code hash: *123259b6656557e675ca48a20c203e6ea7615cec*

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to Delphi Labs. Testing includes, but is not limited to, the following:

- Understanding the application and its code base purpose;
- Deploying SCV in-house tooling to automate dependency analysis and static code review;
- Analyze each line of the code base and inspect application security perimeter;
- Review underlying infrastructure technologies and supply chain security posture;

Code Criteria and Test Coverage

This section below represents how *SUFFICIENT* or *NOT SUFFICIENT* each code criteria was during the assessment

Criteria	Status	Notes
Provided Documentation	**SUFFICIENT**	While we were able to reference the Curve whitepaper, there was no specific Astroport documentation on the implementation of the concentrated liquidity pool.
Code Coverage Test	**SUFFICIENT**	Extensive integration testing and simulations provided robust test coverage.
Code Readability	**SUFFICIENT**	The codebase had good readability and utilized many Rust and CosmWasm best practices.
Code Complexity	**SUFFICIENT**	The concentrated liquidity pool was highly complex.

Threat Modeling

The goal of threat modeling is to identify and evaluate potential threats to a system or application and to develop strategies to mitigate or manage those threats. Threat modeling is an important part of the software development life cycle, as it helps developers and security professionals to proactively identify and address security risks before they can be exploited by attackers.

The main objectives of threat modeling includes (not limited to) the following :

- **Identify threats**: The first objective of threat modeling is to identify potential threats that could affect the security posture of the underlying smart contracts or application. This can include threats from external attackers, internal actors, or even accidental events that could happen.
- **Evaluate risks:** Once potential threats have been identified, the next objective is to evaluate the risks associated with each threat. This involves assessing the likelihood of each threat occurring and the potential impact it could have overall.
- **Mitigation strategies:** After identifying potential threats and evaluating the associated risks, the next objective is to develop strategies to mitigate or reduce the impact of threats. This can include implementing technical controls, such as access controls or further security measures around developing policies and procedures to reduce the likelihood or impact of a threat.
- **Communicate findings:** The final objective of threat modeling is to communicate the findings and recommendations to relevant stakeholders, such as developers, security teams, and management. This helps ensure that everyone involved in the development and maintenance understands the potential risks and the best strategies for addressing them.

During the security audit engagement, SCV performed the following technical details following the threat modeling methodologies described above:

Approach

- Manual Analysis
    - Thorough line by line review of the audit scope.
- Test case analysis
    - Devoted a large amount of time to reviewing test case coverage and simulations.
- Business logic analysis
    - Analyzed potential business logic errors common to AMM pools / pairs.
    - Compared implementation with well known concentrated liquidity pool implementations such as Curve and Uniswap v3.

Actors:

- User
    - Ensure any user cannot call privileged functions;
    - Ensure user receives expected amounts from swap and providing liquidity;
        - Validate through simulation test cases.
        - Ensure first liquidity provider gets fair share,
    - Ensure share is correctly calculated and cant be externally diluted.
        - For example if an attacker can manipulate shares / balances by sending funds directly to the contract
    - Basic validations
        - Only swap expected tokens.
        - Fees are correctly calculated and paid.
    - Ensure users can withdraw successfully and this functionality cannot become blocked (Ex. spamming, out of gas errors, config param changes).
    - Ensure users cannot spam or grief other users or the contract.
    - Ensured user actions could not lead to overflow / underflow.
- Owner
    - Ensure the owner cannot introduce any misconfigurations that affect the contract state, liquidity providers, etc.
    - Ensure the owner does not prevent a centralization risk.
- Contract
    - Ensure accounting is handled correctly on the contract side.
    - Ensure structs are not initialized with default parameters that may harm the protocol.
    - Ensure parameter validations are within reasonable ranges and wont have unintended consequences.

Vulnerabilities Summary

#	Summary Title	Risk Impact	Status
1	Pairs allow callers to provide liquidity after they have been deregistered from the factory	**Low**	**Resolved**
2	Concentrated pair does not contain migration functionality	**Low**	**Acknowledged**
3	First user loses `MINIMUM_LIQUIDITY_AMOUNT`	**Low**	**Acknowledged**
4	`Check_imbalanced_provide` test case does not correctly assess imbalance provide test case	**Informational**	**Resolved**

Audit observations

- The client included extensive integration and simulation tests to ensure coverage of edge cases for the `pair_concentrated` contract. These tests even implemented Proptests. Proptest is a property testing framework which tests that certain properties of the code hold for arbitrary inputs. In summary, SCV concludes that the audited code is concise and has a very strong quality overall.

Detailed Vulnerabilities

1 – Pairs allow callers to provide liquidity after they have been deregistered from the factory

---

Risk Impact: Low - **Status**: Resolved

Description

In the pair_concentrated's `execute` function, there is a migration check performed before any execute message occurs. This checks the factory contract's `PAIRS_TO_MIGRATE` vector. `PAIRS_TO_MIGRATE` is updated when the factory contract confirms an ownership transfer. This ownership transfer updates all pairs stored in the factory contract's `PAIRS` item.

There is currently no validation that the pair is registered with the factory contract. Because the `pair_concentrated` contract can still receive liquidity even after it has been deregistered from the factory contract, this means that a deregistered pair will bypass this migration check. This can lead to a situation where a pair contract can be deprecated, and potentially with a different owner than the factory contract.

We classify this as a low severity because there is a very low likelihood of this occurring and the situation can be resolved by directly updating the contract's owner through the ownership proposal process.

Recommendations

We recommend adding a check that validates with the factory that the pair is registered, and if the pair is no longer registered to prevent liquidity being added.

2 – Concentrated pair does not contain migration functionality

Risk Impact: Low - **Status**: Acknowledged

Description

The `pair_concentrated` contract does not contain a `migrate` entrypoint. This will prevent the contract from being migratable. This differs from the implementations of the other pair contracts.

Recommendations

We recommend adding a migrate entrypoint, otherwise we recommend clearly documenting why the `pair_concentrated` diverges from the implementations of the other pair contracts.

3 – First user loses `MINIMUM_LIQUIDITY_AMOUNT`

Risk Impact: Low - **Status**: Acknowledged

Description

In the `provide_liquidity` function in `contracts/pair_concentrated/src/contract.rs:459`, during the first liquidity provision the `MINIMUM_LIQUIDITY_AMOUNT` is deducted from the caller's share and then minted to the contract. This is likely because the first liquidity provider will be associated with the project and will know about this feature. While this is expected common functionality within the pair contracts, an unknowing user may supply liquidity to the contract and lose a portion of their share.

Recommendations

We recommend designating an address as the first liquidity provider or creating a whitelist to support this functionality. This will ensure that only an address associated with the project can call `provide_liquidity` to perform the first liquidity provision.

4 – `Check_imbalanced_provide` test case does not correctly assess imbalance provide test case

Risk Impact: Informational - **Status**: Resolved

Description

The `check_imbalanced_provide` test cases in `contracts/pair_concentrated/tests/pair_concentrated_integration.rs` do not correctly simulate an imbalanced provide. This is because during the simulation, the test does not correctly add required liquidity to the contract after the first liquidity provision. The function queries `total_share` in `contracts/pair_concentrated/src/contract.rs:408`. The value of `total_share` is used in the conditional in `line 457`. When the second deposit occurs, it does not correctly enter the match branch where the logic can be tested. This occurs because the test case uses two different pools, and makes the first provide to each of them.

Recommendations

We suggest adding a second provide liquidity message to the test to provide better test coverage for the `calc_provide_fee` function.

Document control

Version	Date	Approved by	Changes
0.1	14/02/2023	SCV-Security Team	Document Pre-Release
0.15	21/02/2023	Vinicius Marino	Adding Thread Modeling details
0.2	23/02/2023	SCV-Security Team	Remediation Revisions
1.0	23/02/2023	Vinicius Marino	Document Release

Appendices

A. Appendix – Risk assessment methodology

A qualitative risk assessment is performed on each vulnerability to determine the impact and likelihood of each.

Risk rate will be calculated on a scale. As per criteria Likelihood vs Impact table below:

	Rare	Unlikely	Possible	Likely
Critical	Medium	Severe	Critical	Critical
Severe	Low	Medium	Severe	Severe
Moderate	Low	Medium	Medium	Severe
Low	Low	Low	Low	Medium
Informational	Informational	Informational	Informational	Informational

LIKELIHOOD

- Likely: likely a security incident will occur;
- Possible: It is possible a security incident can occur;
- Unlikely: Low probability a security incident will occur;
- Rare: In rare situations, a security incident can occur;

IMPACT

- Critical: May cause a significant and critical impact;
- Severe: May cause a severe impact;
- Moderate: May cause a moderated impact;
- Low: May cause low or none impact;
- Informational: May cause very low impact or none.

B. Appendix – Report Disclaimer

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts SCV-Security to perform a security review. The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The content of this audit report is provided “as is”, without representations and warranties of any kind, and SCV-Security disclaims any liability for damage arising out of, or in connection with, this audit report.