Zellic

**Prepared for**

Astroport Protocol Foundation

**Prepared by**

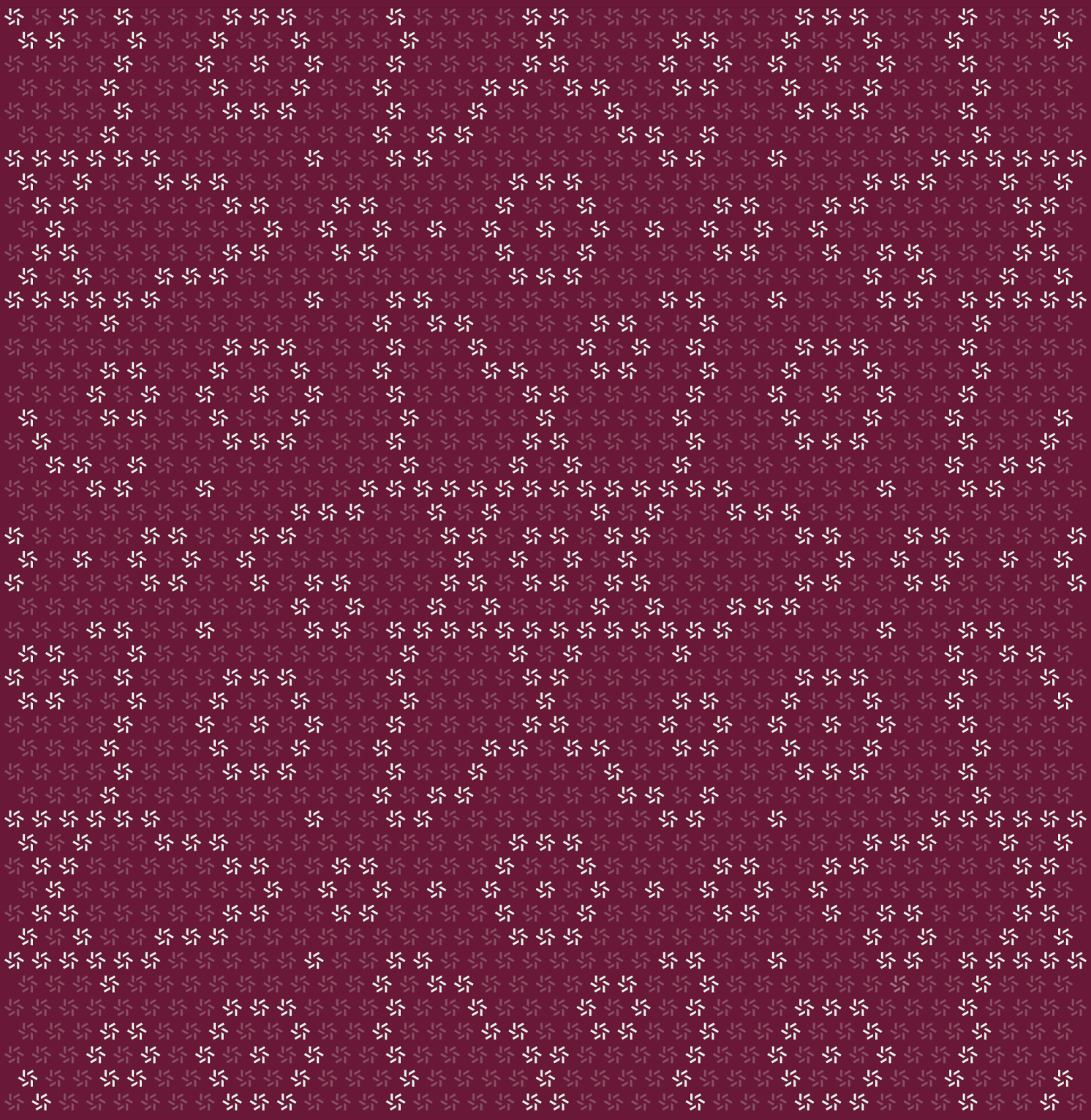
**Filippo Cremonese**

Zellic

**March 15, 2024**

# Astroport Pair XYK Sale Tax

## Smart Contract Security Assessment



# Contents

## About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

# 1. Overview

## 1.1. Executive Summary

Zellic conducted a security assessment for Astroport Protocol Foundation from March 6th to March 7th, 2024. During this engagement, Zellic reviewed Astroport Pair XYK Sale Tax's code for security vulnerabilities, design issues, and general weaknesses in security posture.

## 1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Can the AMM be exploited through a series of trades to drain the liquidity pool?
- Can a user avoid paying sales tax while making a swap?
- Does the slippage protection work as intended?
- Can an unauthorized user update the configuration of the AMM?

## 1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

During this assessment, the two-day time frame prevented us from doing a complete review of the test suite for the contracts.

## 1.4. Results

During our assessment on the scoped Astroport Pair XYK Sale Tax contracts, there were no security vulnerabilities discovered.

## Breakdown of Finding Impacts

| Impact Level | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 0 |
| Informational | 0 |

# 2. Introduction

## 2.1. About Astroport Pair XYK Sale Tax

Astroport Protocol Foundation contributed the following description of Astroport Pair XYK Sale Tax:

> Where traders and LPs meet
> The marketplace never closes on Astroport. Anyone can set up a merchant stall by supplying liquidity in one of three supported pool types. Then, anyone else can trade against those tokens at any time - no centralized exchange needed.

## 2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

**Basic coding mistakes.** Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

**Business logic errors.** Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

**Integration risks.** Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

**Code maturity.** We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect

its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

## 2.3. Scope

The engagement involved a review of the following targets:

### Astroport Pair XYK Sale Tax Contracts

| | |
|---|---|
| **Repository** | https://github.com/astroport-fi/astroport-core ↗ |
| **Version** | astroport-core: `514d83331da3232111c5c590fd8086ef62025ca9` |
| **Program** | contracts/pair_xyk_sale_tax/src/contract.rs |
| **Type** | Rust |
| **Platform** | Cosmos |

## 2.4. Project Overview

Zellic was contracted to perform a security assessment with one consultant for a total of three person-days. The assessment was conducted over the course of two calendar days.

### Contact Information

The following project manager was associated with the engagement:

**Chad McDonald**
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

**Filippo Cremonese**
Engineer
fcremo@zellic.io ↗

## 2.5. Project Timeline

The key dates of the engagement are detailed below.

| | |
|---|---|
| **March 6, 2024** | Start of primary review period |
| **March 7, 2024** | End of primary review period |

# 3. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

## 3.1. Messages

### Message: `ExecuteMsg::ProvideLiquidity`

This message can be sent by anyone and is used to add liquidity to the AMM in return for LP tokens.

The parameters that a user can control are:

- `assets` — This is the list of the two input assets (denom and amount). These must be the same as the assets in the pool. The amounts as checked in `assert_coins_properly_sent`, must be the same as the amounts sent in the message.
- `receiver` — This is the address of the receiver of the LP tokens. If not set, the LP tokens are sent to the sender of the message.
- `auto_stake` — This determines whether the LP tokens are automatically staked. If set, the minted liquidity tokens are instead staked in the Generator contract on behalf of the receiver.
- `slippage_tolerance` — This is an optional parameter that determines the maximum slippage that the user is willing to accept. The transaction will fail if the pool price moves more than the slippage tolerance.

The handler for this message computes the amount of LP tokens to mint and sends them to the receiver. It choses the minimum of the two amounts of LP tokens that would be minted from the two assets. The input assets are checked for normalization to prevent duplicates.

Appropriate messages are generated to mint the LP tokens and to transfer the input assets to the contract. If configured, the asset balances are updated and tracked. The cumulative prices are then updated in the config and the response including the messages and attributes is returned.

### Message: `ExecuteMsg::Swap`

This message can be sent by anyone and allows to perform a swap.

The parameters that a user can control are:

- `offer_asset` — This is the input asset (denom and amount) the user offers. It must be a valid native asset (token asset swaps need to use the `ExecuteMsg::Receive` flow). The amount must match the actual amount bundled with the incoming message, and the asset must be one of the two assets in the pool.

- `belief_price` — This is an optional user-controlled price used to calculate the swap spread, in turn used to enforce anti-slippage checks.
- `max_spread` — This is an optional user-controlled number representing the maximum acceptable spread, for anti-slippage checks.
- `to` — This is the optional address that will receive the output assets. If not provided, the output will be transferred to the message sender.
- `ask_asset_info` — This field of `ExecuteMsg::Swap` is ignored.

The handler for the message computes the output amount that satisfies the k-invariant, accounting for commission fees and maker fees (taken from the output asset) and sales tax (taken from the input asset). All the fees are configurable for each pool. The contract checks that the effective exchange rate satisfies the anti-slippage threshold provided by the user. The message handler generates the appropriate messages to transfer the computed amounts to the required destinations. If configured to do so, the contract will also keep track of asset balances. Lastly, the contract will update the cumulative prices and return a response that includes metadata about the completed swap, including the addresses of the sender and (potentially different) receiver, input and output amounts, commission fees, maker fees, and sales tax amounts.

### Message: `ExecuteMsg::UpdateConfig`

This message allows to update the configuration of the contract, and it can only be sent by the contract owner, or by the tax config admin.

The user provides a serialized `SalesTaxConfigUpdate` struct, and is able to supply the following optional fields:

- `tax_configs` — Can only be provided if the caller is the tax config admin. Allows to update the tax configuration. The new configuration is validated, to ensure it refers to assets in the pool, that the tax rate is not over 50%, and that the tax recipient address is valid.
- `tax_config_admin` — Can only be provided if the caller is the tax config admin. Allows to set the new tax config admin.
- `track_asset_balances` — If true, configures the pool to explicitly keep track of asset balances. Note that asset balance tracking cannot be disabled. Can only be used if the sender is the contract owner.

### Message: `Cw20HookMsg::WithdrawLiquidity`

This message can only be sent by the LP token contract associated with the pool.

The parameters that a user can control are:

- `assets` — This is a list of pool assets. However, this needs to be empty as imbalanced withdraw is not implemented.
- `amount` — This is the amount of LP tokens to burn, in order to receive the share of the pool assets.

- `sender` — This is address that receives the pool assets. It's fixed to initiator of the LP token contract call.

The handler for this message computes the share ratio based on the amount of LP tokens to burn, and sends the pool assets share to the sender. It updates the pool balances and the cumulative prices, and returns the response including the messages and attributes.

### Reply handler

The reply entrypoint only processes successful `INSTANTIATE_TOKEN_REPLY_ID` messages, which can only be the result of a token contract instantiation done in `initialize`. Additionally, the handler also checks that the contract was not initialized previously, by ensuring that the stored LP token address is empty.

If the check passes, the function updates the stored LP token contract address and returns an empty response with a `liquidity_token_addr` attribute that helps to discover the LP token address.

# 4. Assessment Results

At the time of our assessment, the reviewed code was not deployed to Mainnet.

During our assessment on the scoped Astroport Pair XYK Sale Tax contracts, there were no security vulnerabilities discovered.

## 4.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.