Audit Report

Astroport

v0.5

January 19, 2021

Table of Contents

Table of Contents 2

License 4

Disclaimer 4

Introduction 6

- Purpose of this Report 6
- Codebase Submitted for the Audit 6
- Methodology 7
- Functionality Overview 7

How to read this Report 8

Summary of Findings 9

- Code Quality Criteria 11

Detailed Findings 12

- Pools allow attackers to extract free value with minimal cost 12
- Attacker can cause the generator's send orphan proxy rewards function to run out of gas, locking orphan rewards in the contract 13
- Lack of access control on maker contract's set config function allows anyone to set themselves as the fund receiver 13
- Lack of access control on generator contract's set tokens per block function allows anyone to set number of tokens per block and extract value 14
- Lack of mass updating pools before changing alloc points leads to incorrect allocation of pending rewards and causes orphan rewards 14
- Vested but not yet claimed tokens will be lost when replacing a vesting schedule for an account 15
- Missing spread calculation of stable pair contract is misleading to users 15
- Lack of access control on the generator contract's set allowed reward proxies function allows anyone to set proxies 16
- Duplicate storage in two contracts could lead to inconsistencies 16
- Pair and token contract migration is disabled 17
- Sub-message replies are more secure and efficient than used hook pattern 17
- Treating Luna as a special case for tax calculation may lead to problems with Terra protocol updates 18
- Lack of address validation might cause errors when using invalid stored addresses 18
- Lack of fee validation of pair config in factory contract may lead to panics 19
- Unsorted asset infos in a pair in the factory may break backwards-compatibility with TerraSwap 19
- Unbounded iteration over pools could lead to out of gas issues 20
- Generator contract sends update rewards message to reward proxy contract if there are no pending tokens 20

Lack of tax deduction may lead to failure of maker contract's swap to ASTRO function 21
Collection of ASTRO tokens in maker contract is not possible if there are no tokens to be swapped 21
Once set, there is no way to remove an optional governance contract from the maker contract 22
Governance contract has limited permissions and can be replaced by the owner 22
Sub-messages are used where regular messages would be sufficient 22
Unnecessary balance query in pair and pair stable contracts' receive CW20 handler 23
Generator proxy template assumes that unbonded amount is fully available for transfer 23
Duplicated code impacts maintainability 24
Canonical address transformations are inefficient 25
Overflow checks not set for release profile in packages 25
Inefficient query of the proxy reward 26
Approve and transfer pattern impacts usability 26
Unbounded vesting schedules 27

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 Global Partners LLP to perform a security audit of the Astroport 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 GitHub repository:

https://github.com/astroport-fi/astroport

Commit hash: `d76370e439772afebec4f55115f04fcd7fef2f73`

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

Astroport implements an automated, decentralised exchange protocol on the Terra blockchain. The smart contracts audited implement two different types of automated market maker (AMM) pools, constant product pools and stableswap invariant pools, as well as a factory, a router, an oracle that supplies time-weighted average prices (TWAPs), a fee model, staking, vesting and generators, which allow dual liquidity mining/farming on Astroport and other third party protocols.

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**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Pools allow attackers to extract free value with minimal cost	Critical	Resolved
2	Attacker can cause the generator's send orphan proxy rewards function to run out of gas, locking orphan rewards in the contract	Critical	Resolved
3	Lack of access control on maker contract's set config function allows anyone to set themselves as the fund receiver	Critical	Resolved
4	Lack of access control on generator contract's set tokens per block function allows anyone to set extremely high no. tokens per block and extract value	Critical	Resolved
5	Lack of mass updating pools before changing alloc points leads to incorrect allocation of pending rewards and causes orphan rewards	Major	Resolved
6	Vested but not yet claimed tokens will be lost when replacing a vesting schedule for an account	Major	Resolved
7	Missing spread calculation of stable pair contract is misleading to users	Minor	Resolved
8	Lack of access control on the generator contract's set allowed reward proxies function allows anyone to set proxies.	Minor	Resolved
9	Duplicate storage in two contracts could lead to inconsistencies	Minor	Resolved
10	Pair and token contract migration is disabled	Minor	Resolved
11	Sub-message replies are more secure and efficient than used hook pattern	Minor	Resolved
12	Treating Luna as a special case for tax calculation may lead to problems with Terra protocol updates	Minor	Acknowledged
13	Lack of address validation might cause errors when using invalid stored addresses	Minor	Acknowledged
14	Lack of fee validation of pair config in factory contract may lead to panics	Minor	Resolved

15	Unsorted asset infos in a pair in the factory may break backwards-compatibility with TerraSwap	Minor	Acknowledged
16	Unbounded list of the pools get update with rewards that leads to gas exhaustion as there is no limit of pools in a list	Minor	Acknowledged
17	Generator contract sends update rewards message to reward proxy contract if there are no pending tokens	Minor	Resolved
18	Lack of tax deduction may lead to failure of maker contract's swap to ASTRO function	Minor	Resolved
19	Collection of ASTRO tokens in maker contract is not possible if there are no tokens to be swapped	Minor	Resolved
20	Once set, there is no way to remove an optional governance contract from the maker contract	Minor	Resolved
21	Governance contract has limited permissions and can be replaced by the owner	Informational	Resolved
22	Sub-messages are used where regular messages would be sufficient	Informational	Resolved
23	Unnecessary balance query in pair and pair stable contracts' receive CW20 handler	Informational	Resolved
24	Generator proxy template assumes that unbonded amount is fully available for transfer	Informational	Resolved
25	Duplicated code impacts maintainability	Informational	Acknowledged
26	Canonical address transformations are inefficient	Informational	Resolved
27	Overflow checks not set for release profile in packages	Informational	Acknowledged
28	Inefficient query of the proxy reward	Informational	Resolved
29	Approve and transfer pattern impacts usability	Informational	Resolved
30	Unbounded vesting schedules	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of Documentation	Low-Medium	Documentation within the codebase is mostly outdated or non-existent. A few blog posts including a litepaper exist on Medium though.
Test Coverage	Medium-High	-

Detailed Findings

1. Pools allow attackers to extract free value with minimal cost

Severity: Critical

The `compute_swap` function in `contracts/pair/src/contract.rs:733` uses unsigned integers for its calculations without rounding, which opens up a way for an attacker to extract value from a pool with a very small cost.

As an example, imagine a pool with `5_000_000_000_000` A tokens, and a pool with `1_000_000_000` B tokens. If a user now sends `1` B token, we expect the user to get `5_000` A tokens back. When a user sends `1` A token, we expect the user to get `0` B tokens back (actually `0.0002`, but since we are dealing with integers here, the remainder will be dropped). The current implementation incorrectly returns `1` B token though. Imagine further that the value of `1` B token is `5_000` USD, and the value of `1` A token is `1` USD, an attacker can now get a risk free return of around `4_999` USD (minus transaction fees) per transaction. If the attacker repeats this attack, they will be able to drain the pool. Even worse, whenever the attacker shifts the balance enough, other arbitrageurs will be able to extract value by bringing the pool back to the `5_000` to `1` ratio, allowing the attacker to repeat the attack from where they started.

Here is a failing test case demonstrating the example:

```
#[test]
fn compute_swap_rounding() {
    let offer_pool = Uint128::from(5_000_000_000_000_u128);
    let ask_pool = Uint128::from(1_000_000_000_u128);
    let offer_amount = Uint128::from(1_u128);
    let commission_rate = Decimal::from_ratio(0_u128, 1_u128);

    let return_amount = Uint128::from(0_u128);
    let spread_amount = Uint128::from(0_u128);
    let commission_amount = Uint128::from(0_u128);

    assert_eq!(
        compute_swap(offer_pool, ask_pool, offer_amount, commission_rate),
        Ok((return_amount, spread_amount, commission_amount))
    );
}
```

Recommendation

We recommend adjusting the calculation in the `compute_swap` function to remove the rounding issue described above. We also recommend adding test cases to the `compute_swap` and `compute_offer_amount` functions with a wide coverage of edge cases to ensure no other rounding issues exist. A simple way to achieve this is by using a fuzzing library.

Status: Resolved

2. Attacker can cause the generator's send orphan proxy rewards function to run out of gas, locking orphan rewards in the contract

Severity: Critical

In the `send_orphan_proxy_rewards` function in `contracts/tokenomics/generator/src/contract.rs:721`, the `USER_INFO` storage map is iterated over with the LP token prefix. That iteration is unbounded. An attacker can deposit many small amounts to make the iteration long enough for the `send_orphan_proxy_rewards` function to run out of gas. There is currently no way to recover from such an attack, any orphan rewards would be locked forever in the contract.

The `query_orphan_proxy_rewards` query handler exhibits the same issue in line `880`, which could potentially cause calling contracts to run out of gas.

Recommendation

We recommend removing the iteration over the `USER_INFO` storage map and instead tracking the amount of orphan proxy rewards when they emerge, i. e. store them during emergency withdrawals.

Status: Resolved

3. Lack of access control on maker contract's set config function allows anyone to set themselves as the fund receiver

Severity: Critical

There is no access restriction on the `set_config` function in `contracts/tokenomics/maker/src/contract.rs:225`, implying that anyone can change the maker contract's config. An attacker can for example set themselves as the fund receiver, or change the percentage of funds that goes to governance.

Recommendation

We recommend restricting access to the `set_config` function to the owner.

Status: Resolved

4. Lack of access control on generator contract's set tokens per block function allows anyone to set number of tokens per block and extract value

Severity: Critical

There is no access restriction on the `set_tokens_per_block` function in `contracts/tokenomics/generator/src/contract.rs:756`, implying that anyone can change the `tokens_per_block`. An attacker can for example set a high number of tokens per block and extract value.

Recommendation

We recommend restricting access to the `set_tokens_per_block` function to the owner.

Status: Resolved

5. Lack of mass updating pools before changing alloc points leads to incorrect allocation of pending rewards and causes orphan rewards

Severity: Major

In the `add` function in `contracts/tokenomics/generator/src/contract.rs:154`, `total_alloc_point` gets updated, which implicitly changes the allocation for all other pools. Currently, no mass update is performed before that change. That implies that any pending rewards will be distributed according to the updated allocation, causing some rewards to stay unclaimed in the contracts.

For example, imagine there is only one pool A, which has 100 `alloc_point` out of a `total_alloc_point` of also 100. Also suppose that a total of 10 tokens have been accrued so far in rewards, but they have not yet been assigned to pool A. According to these numbers, the LP holders of pool A are entitled to a total of 10 reward tokens. Imagine now that pool B gets added with an `alloc_point` of 300, and immediately afterwards, the rewards for pool A are calculated and allocated. At that point, pool A will only have 100 `alloc_point` out of 400 `total_alloc_point`, and LP holders will now only receive 2.5 of the reward tokens. The remaining 7.5 rewards will stay unallocated in the contract. Those tokens may be withdrawn using the `SendOrphanProxyReward` message.

The same issue exists in the `set` function in `197`.

Recommendation

We recommend enforcing a mass update of all pools before alloc points are updated to correctly allocate pending rewards. Alternatively, for a cleaner architecture without iterating over all pools, global and pool-based reward indexes could be used. With that approach, only one global reward index needs to be updated.

Status: Resolved

6. Vested but not yet claimed tokens will be lost when replacing a vesting schedule for an account

Severity: Major

When updating a vesting account in the `register_vesting_accounts` function, previous vesting schedules may be replaced with updated ones. During that logic, any unclaimed amounts from the vesting schedule are added to `to_receive` in `contracts/tokenomics/vesting/src/contract.rs:136`. `to_receive` is then used to reduce the amount that will be transferred from the owner in line `155` (or is refunded to the owner in line `168`). Any vested, but not yet claimed amounts are also part of `to_receive`, and will hence be lost from a users point of view.

Recommendation

We recommend computing the vested amount and sending any unclaimed vested tokens to the user before replacing an old vesting schedule.

Status: Resolved

7. Missing spread calculation of stable pair contract is misleading to users

Severity: Minor

In the pair stable contract's `compute_swap` and `compute_offer_amount` functions, the spread is not yet computed, but rather hardcoded to 0 in `contracts/pair_stable/src/contract.rs:832` and `872`. Without the actual spread, the max spread assertion, as well as the returned spread amount, are useless, which is misleading to users.

A `TODO` exists in the code that mentions the need for spread calculation.

Recommendation

We recommend adding spread calculation for stable pairs.

Status: Resolved

The spread for stable pairs is now calculated as the deviation from a 1 to 1 exchange rate.

8. Lack of access control on the generator contract's set allowed reward proxies function allows anyone to set proxies

Severity: Minor

There is no access restriction on the `set_allowed_reward_proxies` function in `contracts/tokenomics/generator/src/contract.rs:655`, implying that anyone can set proxies that can be used when adding an LP token to the pool.

We consider this only a minor issue since only an owner can add LP tokens to a pool. Still, an attacker can grief the owner by removing reward proxies from the contract.

Recommendation

We recommend restricting access to the `set_allowed_reward_proxies` function to the owner.

Status: Resolved

9. Duplicate storage in two contracts could lead to inconsistencies

Severity: Minor

Both factory and pair contracts store the `PairInfo` struct which contains information about pairs, i. e. `asset_infos`, `contract_addr`, `liquidity_token`, and `pair_type`, in `contracts/factory/src/state.rs:19` and in `contracts/pair/src/state.rs:16` and `contracts/pair_stable/src/state.rs:17`. This duplicate storage might lead to inconsistencies between the two contract states.

Recommendation

Consider using queries from one central place instead of state duplication in two contracts for increased consistency and better maintainability.

Status: Resolved

10. Pair and token contract migration is disabled

Severity: Minor

In the `Instantiate` message for the pair and token contracts, the `admin` field is set to `None` in `contracts/factory/src/contract.rs:224`, `contracts/pair/src/contract.rs:75`, `contracts/pair_stable/src/contract.rs:80`, and `contracts/tokenomics/staking/src/contract.rs:42`. This implies that pair and token contracts cannot be migrated.

Recommendation

We recommend setting the admin field to the contract owner or governance contract to allow migrations.

Status: Resolved

The pair contract has been made upgradable. The CW20 token contracts still have migrations disabled since their code is not expected to change in the future.

11. Sub-message replies are more secure and efficient than used hook pattern

Severity: Minor

In certain places in the codebase, a hook pattern is used to receive a reply from another contract, for example, the address of an instantiated contract in a post instantiation hook. That hook pattern relies on the called contract calling back the hook. It also requires an exposed message entrypoint for the callback message. That entrypoint needs to have some access control to prevent users from manipulating contract state in a non-intended way.

The hook pattern (calls and callback) is used in:

- `contracts/factory/src/contract.rs:58`
- `contracts/factory/src/contract.rs:229`
- `contracts/factory/src/contract.rs:245`
- `contracts/pair/src/contract.rs:69`
- `contracts/pair/src/contract.rs:84`
- `contracts/pair_stable/src/contract.rs:74`
- `contracts/pair_stable/src/contract.rs:89`
- `contracts/token/src/contract.rs:58`
- `contracts/tokenomics/staking/src/contract.rs:53`

Since Terra's Columbus-5 upgrade, sub-message replies allow for a more idiomatic alternative to that hook pattern. With sub-message replies, the called contract does not need to support

callbacks and no exposed callback entrypoint is needed, which reduces the attack surface of the contract. Additionally, sub-message replies are more efficient than callbacks.

Even though our audit did not reveal any vulnerability with the current usage of the hook pattern, we classify this issue as minor since it could lead to vulnerabilities in the future.

An example is the user supplied message as the `init_hook` to the factory contract's `execute_create_pair` function. That message is sent as a message from the factory contract in `contracts/factory/src/contract.rs:247`. If the factory contract ever holds funds or if it will be extended in the future to have elevated permissions on itself or another contract, this mechanism can be exploited.

Recommendation

We recommend using sub-message replies instead of the hook pattern for a more secure and efficient architecture.

Status: Resolved

12. Treating Luna as a special case for tax calculation may lead to problems with Terra protocol updates

Severity: Minor

In `contracts/router/src/querier.rs:7` and in `packages/astroport/src/asset.rs:36`, Luna is treated as a special case for tax calculations, with a hard-coded zero value. However, this might lead to inconsistencies if Terra changes Luna tax policy in a future protocol update. In such a case, the contract would pay the tax, leading to liquidity being used in the case of the pair contracts or operations failing in the router contract.

Recommendation

We recommend treating Luna the same as other native tokens and querying the tax rate from Terra.

Status: Acknowledged

13. Lack of address validation might cause errors when using invalid stored addresses

Severity: Minor

In several places in the codebase, the `Addr` type is used for user input in the form of unvalidated addresses. An example can be found in

`packages/astroport/src/factory.rs:45`. That leaves those addresses unvalidated, which potentially leads to errors later when using an invalid stored address.

Recommendation

We recommend following the best practice in CosmWasm to accept addresses as `String` types, and then use `let user_addr: Addr = deps.api.addr_validate(input)?` to validate the address and convert the `String` into an `Addr` type.

Status: Acknowledged

14. Lack of fee validation of pair config in factory contract may lead to panics

Severity: Minor

The `PairConfig`'s `total_fee_bps` and `maker_fee_bps` are currently not validated, neither in the factory's `instantiate` function (`contracts/factory/src/contract.rs:25`), nor in the `execute_update_pair_config` function (line `144`). If those values are bigger than 10,000, the fee calculation will panic during swaps.

Recommendation

We recommend adding validation to ensure `total_fee_bps` and `maker_fee_bps` are always smaller than or equal to 10,000.

Status: Resolved

15. Unsorted asset infos in a pair in the factory may break backwards-compatibility with TerraSwap

Severity: Minor

Astroport's Litepaper states that backwards compatibility with TerraSwap should be maintained. One difference between TerraSwap and Astroport is that the assets in a pair are no longer sorted when stored. This

In TerraSwap, the `asset_infos` stored in the factory contract were sorted, while they are stored in the order provided by the user in Astroport in `contracts/factory/src/contract.rs:215` were sorted.

That change is not a problem for the audited contracts, since they use a `pair_key` helper function that generates a key based on the sorted `asset_infos`. This change, however,

might break other contracts that depend on the previous design of stored sorted `asset_infos`.

Recommendation

We recommend storing the `asset_infos` sorted to minimize the probability of breaking other contracts.

Status: Acknowledged

16. Unbounded iteration over pools could lead to out of gas issues

Severity: Minor

In `contracts/tokenomics/generator/src/contract.rs:256` and `349`, unbounded iteration over `POOL_INFO` entries are performed. That implies that the gas consumption increases with the number of pools, which could eventually lead to transactions that hit gas limits.

We classify this issue as minor since only the owner can add pools to `POOL_INFO`.

Recommendation

We recommend adding limits to the number of possible pools to prevent any potential gas issues.

Status: Acknowledged

The Astroport team acknowledges this issue, stating that partially updating pools may lead to wrong reward calculation. Since only governance will be able to add new pools, it is unlikely that this issue will have an impact. If it does, a contract migration can be used to resolve it.

17. Generator contract sends update rewards message to reward proxy contract if there are no pending tokens

Severity: Minor

In `contracts/tokenomics/generator/src/contract.rs:300`, an `UpdateRewards` message is sent to the reward proxy contract if either the `PendingToken` query returned `None` or when it returned a positive amount. In the first case of returning `None`, there is no point in updating rewards.

Recommendation

We recommend changing the condition from

```
res.is_none() || !res.unwrap().is_zero()
```

to

`!res.unwrap_or(Uint128::zero()).is_zero()`.

Status: Resolved

18. Lack of tax deduction may lead to failure of maker contract's swap to ASTRO function

Severity: Minor

In `contracts/tokenomics/maker/src/contract.rs:203`, no taxes are deducted before sending native assets to the pair contract in order to swap them for ASTRO tokens. This implies that the maker's native token balance will be used to pay taxes, until eventually the balance will not suffice and the message will revert.

Recommendation

We recommend deducting taxes from the amount sent.

Status: Resolved

19. Collection of ASTRO tokens in maker contract is not possible if there are no tokens to be swapped

Severity: Minor

The condition in `contracts/tokenomics/maker/src/contract.rs:129` implies that if there are no assets to be swapped, the reply will be skipped. This implies that a positive ASTRO balance in the maker cannot be collected.

Recommendation

We recommend adding an else block to the condition in line `129` that calls `distribute_astro` directly.

Status: Resolved

20. Once set, there is no way to remove an optional governance contract from the maker contract

Severity: Minor

The optional governance contract stored in the config of the maker contract can be updated in `contracts/tokenomics/maker/src/contract.rs:243`, but there is no way to set it back to `None`.

Recommendation

We recommend changing the function signature to accept a `governance_contract` of type `Option<Option<String>>` to allow removing the governance contract.

Status: Resolved

21. Governance contract has limited permissions and can be replaced by the owner

Severity: Informational

The factory contract has both a governance contract and an owner address set in its config. Those roles have different permissions in the current set of contracts – the governance contract can set the amplification parameter for stableswap invariant pools, while the owner can execute many other updates, including replacing the governance contract. This allows owner control and opens up an attack vector in the case owner keys are compromised.

Recommendation

We recommend communicating the governance and owner setup of Astroport clearly in the documentation.

Status: Resolved

22. Sub-messages are used where regular messages would be sufficient

Severity: Informational

Throughout the codebase, sub-messages are used without processing the results. In those cases, regular messages should be used, since they have simpler semantics.

Recommendation

We recommend using regular messages whenever the result of a sub-message is not processed.

Status: Resolved

23. Unnecessary balance query in pair and pair stable contracts' receive CW20 handler

Severity: Informational

In the `receive_cw20` function of both the pair and pair stable contracts, the balance of both tokens in the pool is queried through the `config.pair_info.query_pools` call in `contracts/pair/src/contract.rs:162` and `contracts/pair_stable/src/contract.rs:167`. These balance queries are unnecessary since the returned balance is never used. Just the `contract_address` of the pool is used, which is already stored in the config and does not need to be queried.

Recommendation

We recommend removing the balance queries.

Status: Resolved

24. Generator proxy template assumes that unbonded amount is fully available for transfer

Severity: Informational

In the generator proxy template, the reference implementation suggests that the unbonded amount is fully available for transfer in line `templates/generator_proxy_template/src/contract.rs:130` and `157`. In practice, some protocols apply withdrawal fees, taxes, or might subject bonded funds to slashing. In either case, the received balance might be less than what was unbonded, and the withdrawal would fail.

Recommendation

We recommend changing the reference implementation to query the balance before and after unbonding the funds and only transferring the difference.

Status: Resolved

25. Duplicated code impacts maintainability

Severity: Informational

The codebase contains duplicated code in multiple places, most duplications exist in the pair and pair stable contracts. Duplicated code impacts maintainability and can lead to the introduction of bugs through inconsistencies between duplicated code in the future.

The following are examples, but this list is not exhaustive:

- The `assert_slippage_tolerance` function is identical in `contracts/pair/src/contract.rs:816` and in `contracts/pair_stable/src/contract.rs:925`
- The `assert_max_spread` function is identical in `contracts/pair/src/contract.rs:788` and in `contracts/pair_stable/src/contract.rs:897`
- The `pool_info` function is identical in `contracts/pair/src/contract.rs:849` and in `contracts/pair_stable/src/contract.rs:958`
- The `amount_of` function is unused and identical in `contracts/pair/src/contract.rs:726` and in `contracts/pair_stable/src/contract.rs:799`
- The `get_share_in_assets` function is identical in `contracts/pair/src/contract.rs:402` and in `contracts/pair_stable/src/contract.rs:421`
- The `calculate_maker_fee` function is identical in `contracts/pair/src/contract.rs:558` and in `contracts/pair_stable/src/contract.rs:625`
- The `get_fee_info` function is identical in `contracts/pair/src/contract.rs:580` and in `contracts/pair_stable/src/contract.rs:647`
- The `FeeInfo` struct is identical in `contracts/pair/src/contract.rs:574` and in `contracts/pair_stable/src/contract.rs:641`
- The `compute_tax` function is almost identical in `contracts/router/src/querier.rs:6` and in `packages/astroport/src/asset.rs:35`

Recommendation

We recommend deduplicating the code for higher maintainability.

Status: Acknowledged

The Astroport team decided to keep the duplications to allow easier extensions by other teams.

26. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend removing any transformation from human to canonical addresses and vice versa.

Status: Resolved

27. Overflow checks not set for release profile in packages

Severity: Informational

While set implicitly through the workspace `Cargo.toml`, packages do not explicitly enable overflow checks for the release profile. A future refactor may break implicitly enabled overflow checks, which could lead to security issues through undetected under- or overflows.

The following manifest files are affected:

- `contracts/factory/Cargo.toml`
- `contracts/pair/Cargo.toml`
- `contracts/pair_stable/Cargo.toml`
- `contracts/pair_stable/sim/Cargo.toml`
- `contracts/periphery/oracle/Cargo.toml`
- `contracts/router/Cargo.toml`
- `contracts/token/Cargo.toml`
- `contracts/tokenomics/generator/Cargo.toml`
- `contracts/tokenomics/generator_proxy_to_mirror/Cargo.toml`
- `contracts/tokenomics/maker/Cargo.toml`
- `contracts/tokenomics/staking/Cargo.toml`
- `contracts/tokenomics/vesting/Cargo.toml`
- `packages/astroport/Cargo.toml`
- `templates/generator_proxy_template/Cargo.toml`

Recommendation

We recommend enabling overflow checks in every package, even if no calculations are currently performed in the package, to prevent any issues when the code is extended or refactored in the future.

Status: Acknowledged

28. Inefficient query of the proxy reward

Severity: Informational

In `contracts/tokenomics/generator/src/contract.rs:396`, the reward is queried from the proxy, but it may not be needed if the condition in line `400` evaluates to false.

Recommendation

We recommend moving the query into the block after line `400`.

Status: Resolved

29. Approve and transfer pattern impacts usability

Severity: Informational

In different places in the codebase, an approve and transfer pattern is used to transfer CW20 tokens. Usage of approvals is generally considered a bad practice since they require an additional transaction from the account to grant the allowance and require care in setting appropriate limits or revoke the allowance after interaction with the contract is done. That leads to a degradation in usability. Instances of the approve and transfer pattern are found in:

- `contracts/tokenomics/generator/src/contract.rs:485`
- `contracts/tokenomics/generator/src/contract.rs:497`
- `contracts/tokenomics/staking/src/contract.rs:131`
- `contracts/tokenomics/staking/src/contract.rs:162`

Recommendation

We recommend using the CW20 receive pattern instead.

Status: Resolved

30. Unbounded vesting schedules

Severity: Informational

In `contracts/tokenomics/vesting/src/contract.rs:98`, the owner can register vesting accounts by passing a vector of `VestingAccount`, which contains a vector of `VestingSchedule` for each account. That schedule vector has no length limit – lots of entries make claims more gas-intensive and will eventually lead to a claim hitting gas limits.

This issue can only be caused by the owner and can be recovered from by overwriting vesting schedules.

Recommendation

We recommend adding a limit to the number of vesting schedules that can be added for any account.

Status: Acknowledged