Audit Report

Astroport Core Updates

v1.0

February 10, 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 several updates to the Astroport Core CosmWasm 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-core

Commit hash: `30f7bf348da4600d0b3f56f0e89de9e0c0495299`

The following contracts have been audited fully:

- `contracts/pair`
- `contracts/pair_astro_xastro`
- `contracts/pair_stable`
- `contracts/tokenomics/generator`
- `contracts/whitelist`

For the following contracts, an audit was performed of changes since our previous audit, which was based on commit `4a0a4fc619a7549e1f347727d57e17522719f3fb`:

- `contracts/router`
- `contracts/token`
- `contracts/tokenomics/maker`
- `contracts/tokenomics/staking`
- `contracts/tokenomics/vesting`
- `contracts/tokenomics/xastro_token`

Additionally, the changes in commit `77cad13c6d2c86cbe034137a5ec78d667f9e5ecb` have been reviewed during this audit.

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, decentralized exchange protocol in the Cosmos Ecosystem. This audit is specific to Astroport's core repository updates, split into audits with different changes since the last audit and full re-audits.

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	Most functions are well-documented with clear and concise comments.
Level of documentation	Medium-High	Detailed documentation was provided by the client.
Test coverage	Medium-High	Cargo tarpaulin reported an 87.31% test coverage for the codebase.

Summary of Findings

No	Description	Severity	Status
1	Incorrect virtual amount calculated for generators with reward proxy	Major	Resolved
2	Proxy rewards not included in active pools will be lost upon claiming	Major	Resolved
3	Inadequate conditions for token blacklisting	Major	Resolved
4	Assertion of slippage tolerance in `pair_stable` contract is not performed	Major	Acknowledged
5	Lack of address validation could lead to locked funds	Minor	Resolved
6	Existing pools are removed from `active_pools` during the execution of `execute_setup_pools`	Minor	Acknowledged
7	Max fee value causes swap function to fail	Minor	Resolved
8	Updated contract versions not incremented	Minor	Resolved
9	Insufficient validation of `init_params` of the `pair_astro_xastro` contract	Minor	Resolved
10	Proxy reward holder admin is not updated to new owner	Minor	Acknowledged
11	Max spread can be set above `MAX_ALLOWED_SLIPPAGE`	Minor	Resolved
12	Setting max allowed governance percentage leads to failing CW20 transfers to staking contract	Minor	Resolved
13	Pending TODOs lead to no-effect function	Minor	Acknowledged
14	Staking contract and governance contract can be misconfigured as `None` causing distribution to fail	Minor	Resolved
15	Incorrect event attributes emitted when deactivating pools	Minor	Resolved
16	Tax is calculated but not subtracted from `return_amount`	Minor	Resolved
17	Similar naming may confuse users and developers	Informational	Resolved

18	Token contract version not updated during migration	Informational	Resolved
19	Misleading unauthorized error being raised	Informational	Resolved
20	Use of magic numbers decreases maintainability	Informational	Resolved
21	Unnecessary conversion to lowercase in addresses	Informational	Resolved
22	Additional funds sent to the contract are lost	Informational	Resolved
23	Custom access controls implementation	Informational	Acknowledged
24	Multiple typographical errors	Informational	Resolved
25	Misleading argument naming for query function	Informational	Acknowledged
26	Best practices are not followed during reply	Informational	Resolved
27	Events are not emitted when modifying default bridge and removing governance contract	Informational	Resolved
28	“Migrate only if newer” pattern is not followed	Informational	Resolved

Detailed Findings

1. Incorrect virtual amount calculated for generators with reward proxy

Severity: Major

In `contracts/tokenomics/generator/src/contract.rs:1284-1299`, the `deposit` function updates the user's virtual amount first before sending the liquidity pool tokens to the reward proxy. In this update, new user deposits are not included, causing the user's virtual amount to be calculated incorrectly.

This occurs as the `update_virtual_amount` function in `contracts/tokenomics/generator/src/state.rs:270-273` determines the total token balance by performing a `Deposit` query message. The returned amount does not include the user's deposited liquidity pool tokens when the `deposit` function is called, leading to an amount that is too low.

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

Recommendation

We recommend calculating the total liquidity pool tokens balance when depositing into the generator with a reward proxy.

Status: Resolved

2. Proxy rewards not included in active pools will be lost upon claiming

Severity: Major

In `contracts/tokenomics/generator/src/contract.rs:231`, the `ClaimRewards` message fetches accrued proxy rewards from all active pools instead of the specified liquidity pools (see lines `748-754`). When the `claim_rewards` function in line `982` is executed during the callback, only the specified liquidity pools will have their proxy reward index updated.

This is problematic because if the provided liquidity pools do not contain all the active pools, the proxy reward index for left-out pools will not be updated, causing a loss of rewards for depositors.

This issue was discovered by the client independently during the audit, but it is still present in the commit hash used for the audit.

Recommendation

We recommend modifying the `update_rewards_and_execute` function to support updating multiple pools so the `ClaimRewards` message can update the specified liquidity pools to claim instead of all active pools.

Status: Resolved

3. Inadequate conditions for token blacklisting

Severity: Major

In `contracts/tokenomics/generator/src/contract.rs:488-491`, the `update_blocked_tokens_list` function does not allow native tokens or the `ASTRO` token to be blacklisted. Since IBC tokens are considered native tokens, this means the owner and guardian cannot blacklist IBC tokens, limiting their ability to block undesired tokens.

We classify this issue as major because it affects the correct functionality of the system.

Recommendation

We recommend implementing a whitelist of tokens that cannot be blacklisted.

Status: Resolved

4. Assertion of slippage tolerance in `pair_stable` contract is not performed

Severity: Major

In `contracts/pair_stable/src/contract.rs:412`, the function `assert_slippage_tolerance` is called. However, this function returns `Ok(())` and does not actually assert the slippage tolerance.

While in normal operation, slippage between stable pairs should be minimal, there could be scenarios where slippage is large – for example, if the pool becomes extremely unbalanced. If such a scenario occurred, slippage can become significant. As the provided slippage tolerance is ignored, users will unknowingly provide liquidity at an imbalanced ratio. This will cause them to incur unexpected losses.

Recommendation

We recommend performing slippage tolerance checks in the function `assert_slippage_tolerance` in the `pair_stable` contract.

Status: Acknowledged

5. Lack of address validation could lead to locked funds

Severity: Minor

The generator contract allows users to deposit in benefit of a different address when sending tokens alongside the `Cw20HookMsg::DepositFor` message, which takes a `beneficiary` address as an argument. However, neither the CW20 message handling in `contracts/tokenomics/generator/src/contract.rs:1133` nor the `deposit` function in line `1236` performs validation on the `beneficiary` address. Consequently, this might lead to users depositing to invalid addresses, causing the funds to be locked in the contract forever in case of a typo or mistake.

Recommendation

We recommend validating the `beneficiary` address when handling the `Cw20HookMsg::DepositFor` message.

Status: Resolved

6. Existing pools are removed from `active_pools` during the execution of `execute_setup_pools`

Severity: Minor

When executing the `execute_setup_pools` function in `contracts/tokenomics/generator/src/contract.rs:590-674`, the contract owner or generator controller provides the details of liquidity pool tokens and the allocation points to assign. After instantiation of the pools has been performed, the submitted pools are then written to the `active_pools` vector of the contract config in line `669`.

However, any previous pools that were not re-instantiated will be removed from the `active_pools` vector. This would prevent the deactivation of these pools and automated updates through the execution of `mass_update_pools` at various points in the contract.

We classify this issue as minor because only the contract owner or generator controller can cause it.

Recommendation

We recommend ensuring that all previous pools defined in `prev_pools` in `contracts/tokenomics/generator/src/contract.rs:658`, that are not previously deactivated are appended to the `active_pools` config in line `669`.

Status: Acknowledged

The client states that this is desired behavior designed specifically for the `generator_controller` contract.

7. Max fee value causes swap function to fail

Severity: Minor

In `contracts/pair/src/contract.rs:633` and `contracts/pair_stable/src/contract.rs:714`, the `query_fee_info` function is called to fetch the latest fee amount from the factory contract. If the contract owner sets or updates the `total_fee_bps` value to the max limit of `10_000`, the `return_amount` calculated from the `compute_swap` function would be `0`, causing the transfer message to fail if the ask asset is a CW20 token.

Please see the `test_max_fee_cw20_fail` test case to reproduce the issue.

We classify this issue as minor because only the contract owner can cause it.

Recommendation

We recommend only returning assets if the amount is greater than zero.

Status: Resolved

8. Updated contract versions not incremented

Severity: Minor

For each specific contract, the version is defined in the `Cargo.toml` file. However, for a number of contracts, the version number defined has not been incremented:

- `contracts/router/Cargo.toml.rs:3`
- `contracts/token/Cargo.toml.rs:3`
- `contracts/astroport-pair-stable/Cargo.toml:3`

In each case, the contract has been modified and will require migration, during which time the existing contract version is queried and updated with the new version. The new version is read from the contracts `Cargo.toml` file.

Incorrect increments of the version could lead to incorrect behavior in future migrations and confusion about the current contract version.

Recommendation

We recommend incrementing the contract versions in the relevant `Cargo.toml` files using the relevant semantic version, e.g. major, minor, or patch.

Status: Resolved

9. Insufficient validation of `init_params` of the `pair_astro_xastro` contract

Severity: Minor

During contract instantiation in `contracts/pair_astro_xastro/src/lib.rs:30`, the params are saved by reading the binary field `msg.init_params`. The parameters saved for the `pair_astro_xastro` contract are the addresses of a number of tokens.

However, the addresses are never validated prior to storage and cannot be updated. This could lead to the contract being unable to execute whenever interacting with incorrectly defined addresses.

Recommendation

We recommend performing validation of the `init_params` prior to storage in `contracts/pair_astro_xastro/src/lib.rs:30`.

Status: Resolved

10. Proxy reward holder admin is not updated to new owner

Severity: Minor

In `contracts/tokenomics/generator/src/contract.rs:1619`, the admin of the proxy rewards holder contract is set to the current contract owner. If the contract owner is transferred to another address through the `ProposeNewOwner` and `ClaimOwnership` messages, the proxy rewards holder contract's admin remains with the old owner address.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend updating the proxy reward holder's admin to the new owner's address.

Status: Acknowledged

The client states that they cannot update the admin of the contract as ownership transfer is done via a 2-stage process: `propose_new_owner` and `claim_ownership`. Thus, they acknowledge this issue. In each case of ownership transfer, they will update the admin manually.

11. Max spread can be set above `MAX_ALLOWED_SLIPPAGE`

Severity: Minor

In `contracts/tokenomics/maker/src/contract.rs:64-68`, the max spread provided is validated to not be greater than 100%. This is problematic as the max allowed slippage for a swap is hardcoded in `MAX_ALLOWED_SLIPPAGE` to be 50%, as seen in `contracts/pair/src/contract.rs:1170-1172` and `contracts/pair_stable/src/contract.rs:1349-1351`.

This issue is also present during configuration update in `contracts/tokenomics/maker/src/contract.rs:731-735`.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend ensuring the max spread is lower than 50%.

Status: Resolved

12. Setting max allowed governance percentage leads to failing CW20 transfers to staking contract

Severity: Minor

In `contracts/tokenomics/maker/src/contract.rs:633-636`, the amount to send to the staking contract depends on the remaining amount deducted from the governance amount. The governance amount will be the maximum value if the `governance_percent` is set to the max value of 100%, causing 0 funds to be sent to the staking contract. Consequently, the `distribute` function will fail because CW20 tokens prevent sending 0 amounts of funds.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend only sending funds if the amount is greater than zero.

Status: Resolved

13. Pending TODOs lead to a no-effect function

Severity: Minor

In `contracts/tokenomics/generator/src/contract.rs:1401-1445`, the `build_claim_pools_asset_reward_messages` function is defined for the generator

contract. Part of the logic of the function, lines `1428-1440`, is commented out and marked as `TODO`. As a result, the function does not perform any change to the contract's storage and always returns an empty vector instead of building claim reward messages for a specific generator as expected.

Recommendation

We recommend resolving the pending `TODO` comments and fully implementing the functionality.

Status: Acknowledged

The client states that they prefer to keep these TODOs as reminders. At the moment, Astroport does not have a requirement to accrue underlying assets rewards (like it was on Terra Classic for `bLuna`), but this may come out in the future.

14. Staking contract and governance contract can be misconfigured as `None` causing distribution to fail

Severity: Minor

In `contracts/tokenomics/maker/src/contract.rs:710-712`, the contract owner can remove the governance contract which sets it to `None` value. This is problematic because the validation in lines `95-99` does not allow both the staking contract and governance contract to be set as `None`. Since the staking contract can be set as `None` during contract instantiation, removing the governance contract breaks the invariant mentioned above, causing no funds to be distributed via the `distribute` function.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend ensuring the staking contract is `Some` before allowing the governance contract to be removed.

Status: Resolved

15. Incorrect event attributes emitted when deactivating pools

Severity: Minor

In `contracts/tokenomics/generator/src/contract.rs:909`, the `deactivate_pool` function emits an attribute action with its value as `setup_pool`. This is incorrect and could cause confusion to users and offchain services that consume events.

Recommendation

We recommend replacing the attribute action `setup_pool` with `deactivate_pool`.

Status: Resolved

16. Tax is calculated but not subtracted from `return_amount`

Severity: Minor

In `packages/pair_bonded/src/base.rs:387`, the tax payable is calculated. However, this amount is not deducted from the `return_amount` that is subsequently transferred in line `390`.

As a result, transactions would fail in the scenario that tax is no longer zero.

Recommendation

We recommend calculating and deducting tax from `return_amount` prior to the creation of the `return_asset` in `packages/pair_bonded/src/base.rs:382`.

Status: Resolved

17. Similar naming may confuse users and developers

Severity: Informational

The generator contract defines two messages and functions with very similar naming but very different authorization levels.

In `contracts/tokenomics/generator/src/contract.rs:183-184`, `DeactivatePools` and `DeactivatePool` are defined. The underlying functions that handle the input of these messages are named `deactivate_pool` and `deactivate_pools`.

Small one-character typos like `ExecuteMsg::DeactivatePools {...} => deactivate_pool(...)` are hard to spot. Using a very close naming convention increases the chances of the mentioned typo causing a bug as the functions would end up being "switched", potentially leading to a security issue.

Recommendation

We recommend using a different naming convention for the affected functionalities. For example, `DeactivatePools` and `deactivate_pools` could be renamed to `DeactivateBlacklisted` and `deactivate_blacklisted_pools` instead.

Status: Resolved

18. Token contract version not updated during migration

Severity: Informational

During the migration of the token contract in `contracts/token/src/contract.rs:201-203`, the contract version is not updated as specified in the CW2 standard. This could confuse users over which version of the specific token contract is currently deployed and increase the complexity of future migrations.

Recommendation

We recommend executing the function `set_contract_version`, as performed during instantiation, during the migration in lines `contracts/token/src/contract.rs:201-203`.

Status: Resolved

19. Misleading unauthorized error being raised

Severity: Informational

The `ContractError::Unauthorized` error is returned under different circumstances that do not relate to access control or authorization.

Although not a security issue, it is confusing for users to receive an authorization error when submitting an invalid message or supplying incorrect assets to a `Swap` message.

This issue is present in the following lines:

- `contracts/tokenomics/xastro_token/src/contract.rs:282`
- `contracts/pair/src/contract.rs:202`
- `contracts/pair_stable/src/contract.rs:229`
- `packages/pair_bonded/src/base.rs:237`

Recommendation

We recommend using meaningful errors.

Status: Resolved

20. Use of magic numbers decreases maintainability

Severity: Informational

In `contracts/tokenomics/generator/src/state.rs:268` and `282`, hard-coded number literals without context or a description are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers are

unable to easily understand their use and may make inconsistent changes across the codebase.

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

21. Unnecessary conversion to lowercase in addresses

Severity: Informational

The contracts within scope used the `addr_validate_to_lower` helper function to sanitize addresses. Since CosmWasm `1.0.0`, the `addr_validate` utility also validates address capitalization, hence making it redundant to perform this check manually.

Recommendation

We recommend removing the `addr_validate_to_lower` and performing address validation through `api.addr_validate` instead.

Status: Resolved

22. Additional funds sent to the contract are lost

Severity: Informational

A check for existence of a `Coin` with the expected `denom` field is performed in:

- `contracts/pair/src/contract.rs:333` and
- `contracts/pair/src/contract.rs:643`.

This validation does not ensure that no other native tokens are sent though, and any additional native tokens are not returned to the user, so they will be stuck in the contract forever.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

23. Custom access controls implementation

Severity: Informational

Multiple contracts within scope implement custom access controls. Although no instances of broken controls or bypasses have been found, using a battle-tested implementation reduces potential risks and the complexity of the codebase.

Also, the access control logic is duplicated across the handlers of each function, which negatively impacts the code's readability and maintainability as it is error-prone.

Recommendation

We recommend using a well-known access control implementation such as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html).

Status: Acknowledged

24. Multiple typographical errors

Severity: Informational

The following typographical errors are present in the codebase. Although not a security issue, this decreases documentation quality and readability.

- `contracts/pair/src/contract.rs:736` - “ontract” instead of “contract”
- `contracts/pair_stable/src/math.rs:48` - “offer tokens to swap” instead of “ask tokens to swap”
- `contracts/router/src/contract.rs:446,455` - `offer_amount` instead of `ask_amount` or `return_amount`
- `contracts/tokenomics/generator/src/contract.rs:1792` - “specifi” instead of “specific”
- `contracts/tokenomics/generator/src/migration.rs:74` - `config(V2.0.0)` instead of `config(V2.1.0)`
- `contracts/tokenomics/xastro_token/src/contract.rs:743` - [`aStdError`] instead of [`StdError`]

Recommendation

We recommend correcting these typographical errors.

Status: Resolved

25. Misleading argument naming for query function

Severity: Informational

The xAstro token contract implements the `QueryMsg::AllAllowances` query entry point, which returns the allowances of any address supplied. However, the address parameter has been named `owner` and the underlying function `query_owner_allowances` which is misleading as this feature is not restricted to the owner address.

Recommendation

We recommend using a more accurate naming for this function. For example, `user` instead of `owner` and `query_user_allowances` instead of `query_owner_allowances`.

Status: Acknowledged

The client states that these namings came from basic `cw20` implementation https://github.com/CosmWasm/cw-plus/blob/339c9d5fb0f3de95b8b61e9c56127d2cd646e644/packages/cw20/src/query.rs#L40. In this context, `owner` is an owner of tokens who approved allowances.

26. Best practices are not followed during reply handling

Severity: Informational

When executing sub-message replies the `reply` call code IDs are not validated. See:

- `contracts/pair/src/contract.rs:119`
- `contracts/pair_stable/src/contract.rs:145`
- `contracts/tokenomics/staking/src/contract.rs:127`

In the instances listed above no validation of the reply ID is performed, which is against best practices.

Missing validation of sub-message IDs can lead to unexpected execution, increased code complexity, and maintenance issues as the codebase grows.

Recommendation

We recommend that all sub-message responses are validated for a specific ID prior to continued execution and an error message is returned in the case that an unknown ID is received.

Status: Resolved

27. Events are not emitted when modifying default bridge and removing governance contract

Severity: Informational

In `contracts/tokenomics/maker/src/contract.rs:711` and `727`, no attribute is emitted when the contract owner modifies the default bridge or removes the governance contract. This might cause issues for network participants who are informed about changes to the default bridge or the removal of the governance contract.

Recommendation

We recommend emitting attributes when the contract owner modifies the default bridge or removes the governance contract.

Status: Resolved

28. "Migrate only if newer" pattern is not followed

Severity: Informational

The pair `astroport-pair-astro-xastro` and `astroport-pair-stable` contracts within scope are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommended following the migrate "only if newer" pattern defined in the CosmWasm documentation.

Status: Resolved

Appendix A: Test Cases

1. **Test case for "Incorrect virtual amount calculated for generators with reward proxy"**

The test case should pass if the vulnerability is patched.

```
#[test]
fn test_proxy_generator_incorrect_virtual_amount() {
    // reproduced in contracts/tokenomics/generator/tests/integration.rs

    let mut app = mock_app_helper();
    let owner = Addr::unchecked("owner");
    let helper_controller = ControllerHelper::init(&mut app, &owner);

    let user1 = Addr::unchecked(USER1);

    let token_code_id = store_token_code(&mut app);

    // init cw20 tokens
    let cny_token = instantiate_token(&mut app, token_code_id, "CNY", None);
    let eur_token = instantiate_token(&mut app, token_code_id, "EUR", None);
    let val_token = instantiate_token(&mut app, token_code_id, "VAL", None);

    // create two lp pairs, one with proxy another without proxy
    let (pair_cny_eur, lp_without_proxy) = create_pair(
        &mut app,
        &helper_controller.factory,
        None,
        None,
        [
            AssetInfo::Token {
                contract_addr: cny_token.clone(),
            },
            AssetInfo::Token {
                contract_addr: eur_token.clone(),
            },
        ],
    );

    let (pair_val_eur, lp_with_proxy) = create_pair(
        &mut app,
        &helper_controller.factory,
        None,
        None,
        [
            AssetInfo::Token {
                contract_addr: val_token.clone(),
```

```
            },
            AssetInfo::Token {
                contract_addr: eur_token.clone(),
            },
        ],
    );

    // register lp token to pool
    register_lp_tokens_in_generator(
        &mut app,
        &helper_controller.generator,
        vec![PoolWithProxy {
            pool: (lp_without_proxy.to_string(), Uint128::from(100u32)),
            proxy: None,
        }],
    );

    // verify no proxy set
    let reps: PoolInfoResponse = app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::PoolInfo {
            lp_token: lp_without_proxy.to_string(),
        },
    ).unwrap();

    assert_eq!(None, reps.reward_proxy);

    // mint lp without proxy to user
    mint_tokens(&mut app, pair_cny_eur.clone(), &lp_without_proxy, &user1, 10);

    helper_controller
        .escrow_helper
        .mint_xastro(&mut app, USER1, 100);

    helper_controller
        .escrow_helper
        .create_lock(&mut app, USER1, WEEK * 3, 100f32)
        .unwrap();

    // user deposits lp tokens
    deposit_lp_tokens_to_generator(
        &mut app,
        &helper_controller.generator,
        USER1,
        &[(&lp_without_proxy, 10)],
    );

    // NOTE: user virtual amount should be calculated correctly when deposit
    // first we try query the virtual amount and grab the value
    // secondly we call CheckpointUserBoost to update the user's virtual amount
```

```
to latest value
    // third we query the virtual amount
    // lastly we compare it, should be equal

    // 1: query before checkpoint
    let virtual_amount_before_checkpoint : Uint128 =
app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::UserVirtualAmount {
            lp_token: lp_without_proxy.to_string(),
            user: USER1.to_string(),
        },
    ).unwrap();

    // 2: perform checkpoint, user virtual amount will be updated
    app.execute_contract(
        Addr::unchecked(USER1),
        helper_controller.generator.clone(),
        &ExecuteMsg::CheckpointUserBoost {
            generators: vec![lp_without_proxy.to_string()],
            user: Some(USER1.to_string()),
        },
        &[],
    )
    .unwrap();

    // 3: query after checkpoint
    let virtual_amount_after_checkpoint : Uint128 = app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::UserVirtualAmount {
            lp_token: lp_without_proxy.to_string(),
            user: USER1.to_string(),
        },
    ).unwrap();

    // 4: amounts should be the same, correct!
    assert_eq!(virtual_amount_after_checkpoint,
virtual_amount_before_checkpoint);

    // let's see if its the same for a lp with proxy

    // setup lp to use proxy
    let vkr_staking_instance =
        instantiate_valkyrie_protocol(&mut app, &val_token, &pair_val_eur,
&lp_with_proxy);

    let proxy_code_id = store_proxy_code(&mut app);

    let proxy_instance = instantiate_proxy(
        &mut app,
```

```
        proxy_code_id,
        &helper_controller.generator,
        &pair_val_eur,
        &lp_with_proxy,
        &vkr_staking_instance,
        &val_token,
    );

    let msg = GeneratorExecuteMsg::MoveToProxy {
        lp_token: lp_with_proxy.to_string(),
        proxy: proxy_instance.to_string(),
    };

    app.execute_contract(
        Addr::unchecked(OWNER),
        helper_controller.generator.clone(),
        &msg,
        &[],
    )
    .unwrap();

    // verify proxy has been set
    let reps: PoolInfoResponse = app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::PoolInfo {
            lp_token: lp_with_proxy.to_string(),
        },
    ).unwrap();

    assert_eq!(Some(proxy_instance), reps.reward_proxy);

    // mint lp tokens to user
    mint_tokens(&mut app, pair_val_eur.clone(), &lp_with_proxy, &user1, 10);

    // user deposits lp tokens
    deposit_lp_tokens_to_generator(
        &mut app,
        &helper_controller.generator,
        USER1,
        &[(&lp_with_proxy, 10)],
    );

    // similar with lp without proxy, let's peform the same verification

    // 1: query before checkpoint
    let virtual_amount_before_checkpoint : Uint128 =
app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::UserVirtualAmount {
            lp_token: lp_with_proxy.to_string(),
```

```
            user: USER1.to_string(),
        },
    ).unwrap();

    // 2: perform checkpoint, user virtual amount will be updated
    app.execute_contract(
        Addr::unchecked(USER1),
        helper_controller.generator.clone(),
        &ExecuteMsg::CheckpointUserBoost {
            generators: vec![lp_with_proxy.to_string()],
            user: Some(USER1.to_string()),
        },
        &[],
    )
    .unwrap();

    // 3: query after checkpoint
    let virtual_amount_after_checkpoint : Uint128 = app.wrap().query_wasm_smart(
        &helper_controller.generator,
        &QueryMsg::UserVirtualAmount {
            lp_token: lp_with_proxy.to_string(),
            user: USER1.to_string(),
        },
    ).unwrap();

    /*
    4: compare: error here

    panicked at 'assertion failed: `(left == right)`
        left: `Uint128(4)`,
        right: `Uint128(10)`
    */
    assert_eq!(virtual_amount_before_checkpoint,
virtual_amount_after_checkpoint);

}
```

2. Test case for "Max fee value causes swap function to fail"

The test case should fail if the vulnerability is patched. Please note that slight modification is required in order for this test case to work properly.

```
#[test]
fn test_max_fee_cw20_fail() {
    // reproduced in contracts/pair/src/testing.rs
    // to reproduce this, please modify contracts/pair/src/mock_querier.rs:87 to
10_000 to facilitate full total_fee_bps fee
    let total_share = Uint128::new(30000000000u128);
    let asset_pool_amount = Uint128::new(20000000000u128);
    let collateral_pool_amount = Uint128::new(30000000000u128);
    let offer_amount = Uint128::new(1500000000u128);

    let mut deps = mock_dependencies(&[Coin {
        denom: "uusd".to_string(),
        amount: collateral_pool_amount + offer_amount, /* user deposit must be
pre-applied */
    }]);

    deps.querier.with_token_balances(&[
        (
            &String::from("liquidity0000"),
            &[(&String::from(MOCK_CONTRACT_ADDR), &total_share)],
        ),
        (
            &String::from("asset0000"),
            &[(&String::from(MOCK_CONTRACT_ADDR), &asset_pool_amount)],
        ),
    ]);

    let msg = InstantiateMsg {
        asset_infos: [
            AssetInfo::NativeToken {
                denom: "uusd".to_string(),
            },
            AssetInfo::Token {
                contract_addr: Addr::unchecked("asset0000"),
            },
        ],
        token_code_id: 10u64,
        factory_addr: String::from("factory"),
        init_params: None,
    };

    let env = mock_env();
    let info = mock_info("addr0000", &[]);
    // we can just call .unwrap() to assert this was a success
    let _res = instantiate(deps.as_mut(), env, info, msg).unwrap();
```

```
    // Store liquidity token
    store_liquidity_token(deps.as_mut(), 1, "liquidity0000".to_string());

    // Normal swap
    let msg = ExecuteMsg::Swap {
        offer_asset: Asset {
            info: AssetInfo::NativeToken {
                denom: "uusd".to_string(),
            },
            amount: offer_amount,
        },
        belief_price: None,
        max_spread: Some(Decimal::percent(50)),
        to: None,
    };
    let env = mock_env_with_block_time(1000);
    let info = mock_info(
        "addr0000",
        &[Coin {
            denom: "uusd".to_string(),
            amount: offer_amount,
        }],
    );

    let res = execute(deps.as_mut(), env, info, msg).unwrap();

    // Current price is 1.5, so expected return without spread is 1000
    // 952380952 = 20000000000 - (30000000000 * 20000000000) / (30000000000 +
1500000000)
    let expected_ret_amount = Uint128::new(952_380_952u128);

    // 47619047 = 1500000000 * (20000000000 / 30000000000) - 952380952
    let _expected_spread_amount = Uint128::new(47619047u128);

    // commision modified to full amount
    let expected_commission_amount =
expected_ret_amount.multiply_ratio(1000u128, 1000u128);

    let expected_return_amount = expected_ret_amount
        .checked_sub(expected_commission_amount)
        .unwrap();

    println!("{:?}", res.attributes);

    for i in res.attributes {
        if i.key == "return_amount" {
            assert_eq!(i.value, expected_return_amount.to_string());
        } else if i.key == "commission_amount" {
            assert_eq!(i.value, expected_commission_amount.to_string(),);
```

```
        }
    }

    assert_eq!(
        res.messages[0],
        SubMsg {
            msg: WasmMsg::Execute {
                contract_addr: String::from("asset0000"),
                msg: to_binary(&Cw20ExecuteMsg::Transfer {
                    recipient: String::from("addr0000"),
                    amount: Uint128::from(0u128), // tries to transfer 0 amount
which will fail
                })
                .unwrap(),
                funds: vec![],
            }
            .into(),
            id: 0,
            gas_limit: None,
            reply_on: ReplyOn::Never,
        }
    )
}
```