Audit Report

Astroport Governance Updates

v1.0

February 14, 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 Governance 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-governance

Commit hash: `2bddf7d0da6877072b9c64204745636cfda45480`

This audit was performed on changes since our previous audit, which was based on commit `0b1a4282b062cc2f9e8fb3684f203e6fad1f9fd2`.

The following directories were in scope of this audit:

- `contracts/assembly`
- `contracts/generator_controller`
- `contracts/voting_escrow`
- relevant files in `packages/*`

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

The submitted code features a diff audit of the Astroport Governance repository since our last audit.

A summary of changes is provided below:

- `contracts/assembly`
    - Ability to check messages before submitting a proposal.
    - Governing remote chains for proposals via the controller contract.
- `contracts/generator_controller`
    - Implemented a main pool with minimum allocation.
    - Allow kicking blacklisted voters for the generator controller.
- `contracts/voting_escrow`
    - Added a guardian role to blacklist or whitelist `vxASTRO` stakers.

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**	Documentation was limited to a high-level diagram and a short description of each entry point.
Test coverage	**Medium-High**	`cargo tarpaulin` reported a code coverage of 87.31%.

Summary of Findings

No	Description	Severity	Status
1	Attackers can block pool tuning by voting on many dummy pools	Major	Resolved
2	Proposal's message order not enforced on IBC execution	Major	Resolved
3	Modifying `xASTRO` or `vxASTRO` addresses in assembly contract may cause state inconsistency issues	Minor	Resolved
4	Best practices to ensure contract name change does not affect future migrations	Informational	Resolved
5	Empty message proposals are sent via IBC	Informational	Resolved
6	`KickBlacklistedVoters` missing from documentation	Informational	Resolved
7	Unnecessary CW20 sender validation	Informational	Resolved
8	Unnecessary conversion to lowercase in addresses	Informational	Resolved
9	Misleading error messages	Informational	Resolved
10	"Migrate only if newer" pattern is not followed	Informational	Resolved
11	Custom access controls implementation	Informational	Acknowledged

Detailed Findings

1. Attackers can block pool tuning by voting on many dummy pools

Severity: Major

In `contracts/generator_controller/src/contract.rs:398-416`, the `tune_pools` function attempts to process each pool's votes in an unbounded iteration. In the case that the number of pools is too many, the execution may run out of gas and fail.

An attacker can use this loop to block pool tuning:

1. The attacker creates many addresses and stakes `xASTRO` for minimum voting power.
2. To bypass the `pair_info_by_pool` validation in line `311`, the attacker creates a parent contract that implements a `Pair` query message, which returns a dummy `PairInfo` struct when queried.
3. Many child contracts are created that implement the `Minter` query message. The returned `MinterResponse.minter` will point to the parent contract to fully satisfy the LP token address validation.
4. Each address executes the `handle_vote` function with different child addresses, causing them to be registered as valid pools in `contracts/generator_controller/src/utils.rs:171`.
5. Finally, the `tune_pools` function will attempt to process all child contracts added by the attacker in line `399`. The attack will succeed when the function fails due to an out-of-gas error.

As a result, the `TunePools` message will always fail, preventing majority-voted pools from receiving allocation points.

Please refer to the `test_jam_tuning` test case for the attack scenario mentioned above.

Recommendation

We recommend implementing a whitelist of liquidity pool contracts that can be voted on.

Status: Resolved

2. Proposal's message order not enforced on IBC execution

Severity: Major

The `assembly` contract's `execute_proposal` function sorts proposal messages by its `order` field in `contracts/assembly/src/contract.rs:481` when the proposal is executed on the same chain. However, the ordering is not enforced if the proposal is to be executed over IBC, as seen in lines `458-473`.

Executing proposal messages without following the expected order could lead to unwanted results in some circumstances. For example, if the community decides to pass a proposal to grant the rewards accrued in a different protocol to a specific user for their contributions, it will cause very different results if the `claim` message is ordered before or after the `transfer` message.

Recommendation

We recommend sorting the proposal's messages by order when including them in the `IbcExecuteProposal` message.

Status: Resolved

3. Modifying `xASTRO` or `vxASTRO` addresses in assembly contract may cause state inconsistency issues

Severity: Minor

In `contracts/assembly/src/contract.rs:583-589`, the `update_config` function allows the `xASTRO` or `vxASTRO` token addresses to be updated. This is problematic because state inconsistency issues would occur, potentially disrupting normal operations of the protocol.

For example, the actual token balance held by the contract might be different from the recorded storage state amount, causing proposals being unable to be ended due to insufficient refund balances.

We classify this issue as minor because only governance can modify the token addresses.

Recommendation

We recommend removing the ability for governance to update the `xASTRO` and `vxASTRO` token addresses.

Status: Resolved

4. Best practices to ensure contract name change does not affect future migrations

Severity: Informational

In `contracts/generator_controller/Cargo.toml:2`, the generator controller's name is changed from `astroport-generator-controller` to `generator-controller`. The contract name is used when performing a migration, and may lead to issues if changed.

In future releases, a migration could be developed for a new version of a contract named `generator-controller` while there may still be contracts deployed with the old name.

Recommendation

We recommend preemptively adding the latest version contract name to the match function in `contracts/generator_controller/src/contract.rs:631` to ensure that in the future, developers are reminded of the contract name change.

Status: Resolved

5. Empty message proposals are sent via IBC

Severity: Informational

When creating proposals to be executed on other chains via IBC in `contracts/assembly/src/contract.rs:458-473`, an `IbcExecuteProposal` message is created, independ whether it contains any messages to be executed.

In the case of a proposal without any messages, this causes the `satellite` contract to record the proposal forever without the ability for the proposal to be executed or removed.

Recommendation

We recommend verifying whether all IBC proposals contain messages prior to the creation and execution of `IbcExecuteProposal` messages in `contracts/assembly/src/contract.rs:458-473`, as in the case of a non-IBC proposal.

Status: Resolved

6. `KickBlacklistedVoters` missing from documentation

Severity: Informational

In `contracts/generator_controller/src/contract.rs:89-108`, the generator controller's execution messages are documented. However, the `ExecuteMsg::KickBlacklistedVoters` is not defined.

Recommendation

We recommend adding the description of the function `ExecuteMsg::KickBlacklistedVoters` to the documentation.

Status: Resolved

7. Unnecessary CW20 sender validation

Severity: Informational

In `contracts/assembly/src/contract.rs:170`, the `assembly` contract's `receive_cw20` function performs address validation for the sender parameter of the CW20 message. This is unnecessary because the sender parameter comes from `info.sender`, validated by Cosmos SDK already.

Recommendation

We recommend not validating `cw20_msg.sender`.

Status: Resolved

8. 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

9. Misleading error messages

Severity: Informational

In `contracts/assembly/src/contract.rs:680` and `683`, the `assembly` contract's `update_ibc_proposal_status` function returns `Unauthorized` errors even for cases that are not related to authorization, which can be misleading for users.

This is the case if the `proposal.status` does not match one of the expected ones in line `671`.

This also happens if `config.ibc_controller` is configured to `None`, the `if` statement in line `669` will result in an `Unauthorized` error.

Recommendation

We recommend raising a different error if the expected proposal status does not match. Additionally, consider returning a meaningful error when the `ibc_controller` is `None`.

Status: Resolved

10. “Migrate only if newer” pattern is not followed

Severity: Informational

The contracts within scope of this audit are currently migrated without regard of 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

11. Custom access controls implementation

Severity: Informational

The contracts within scope implement custom access controls. Although no instances of broken controls or bypasses have been found, using a single assert function to validate controls reduces potential risks while improving the codebase’s readability and maintainability.

Recommendation

We recommend using modular functions to implement any access control check that has to be used multiple times.

Status: Acknowledged

Appendix: Test Cases

1. Test case for "Attackers can block tuning pools by voting many dummy pools"

```
#[test]
fn test_jam_tuning() {
    // reproduced in contracts/generator_controller/tests/integration.rs

    let mut router = mock_app();
    let owner = "owner";
    let owner_addr = Addr::unchecked(owner);
    let helper = ControllerHelper::init(&mut router, &owner_addr);

    let mut pools_to_vote : Vec<(String, u16)> = vec![];

    // attacker creates 1000 child contracts that implements the Minter{}
    // the MinterResponse.minter query points to the parent contract implements
the Pair{} query
    for _ in 0..1000 {
        let pool = helper
            .create_pool_with_tokens(&mut router, &"FOO", &"BAR")
            .unwrap();
        let pool_to_vote = (pool.to_string(), 1000_u16);
        pools_to_vote.push(pool_to_vote);
    }
    assert_eq!(pools_to_vote.len(), 1000);

    // attacker create 100 accounts and stakes for voting power
    let mut attacker_accounts : Vec<String> = vec![];
    for i in 0..100 {
        let s = String::new() + "user" + &i.to_string();
        attacker_accounts.push(s.clone());
        helper.escrow_helper.mint_xastro(&mut router, &s, 1000);
        helper.escrow_helper.create_lock(&mut router, &s, 2 * WEEK,
100f32).unwrap();
    }

    // 100 accounts each vote for 10 pools (hopefully its realistic enough)
    let per_account = pools_to_vote.len() / attacker_accounts.len();

    let mut pool_iter = pools_to_vote.clone().into_iter();

    for acc in attacker_accounts {
        let to_vote: Vec<_> = pool_iter.by_ref().take(per_account).collect();

        helper.vote(
            &mut router,
            &acc,
            to_vote.clone(),
        )
        .unwrap();
    }

    // ensure all 1000 pools are recorded in contract
    assert_eq!(pool_iter.len(), 0);

    /*
    - Try to tune pool, since all POOLS are fetched without limit, an attacker
can cause it to fail

    // contracts/generator_controller/src/contract.rs:399
    let pool_votes: Vec<_> = POOLS
        .keys(deps.as_ref().storage, None, None, Order::Ascending)
        .collect::<Vec<_>>()
        .into_iter()
        .map()
    */
    router.next_block(WEEK * 2);

    helper.tune(&mut router).unwrap();

}
```