Audit Report

Astroport IBC

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 the Astroport IBC 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_ibc

Commit hash: `c6ca81b347fc71fe0f520cf3ed704fb96b1c2f20`

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 the Astroport IBC repository, which enables inter-blockchain communications for Astroport Governance.

The audit was conducted on the following contracts:

- `controller` contract, an IBC controller contract intended to be hosted on the main chain.
- `cw20-ics20` contract, an IBC-enabled contract that receives CW20 tokens and sends them over the IBC channel to a remote chain.
- `satellite` contract, an IBC-enabled Astroport satellite contract intended to be hosted on a remote chain.

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	Functions are structured and easy to read. However, no clear documentation or concise comments were provided.
Level of documentation	Low-Medium	The only piece of documentation provided was a high-level diagram. In addition, contract's functionality was not commented at all with the exception of a few lines of comments on the storage.
Test coverage	Medium-High	`cargo tarpaulin` reported a code coverage of 78.30%.

Summary of Findings

No	Description	Severity	Status
1	Incorrect permissioning of `IbcExecuteProposal` execution leads to failure of proposal execution and elevated owner privileges	Major	Resolved
2	No validation of IBC message timeout	Minor	Resolved
3	`assembly` contract owner can overwrite proposals	Minor	Resolved
4	Inconsistent expected format for main controller port	Minor	Resolved
5	`RESULTS` storage does not save empty-message proposals	Minor	Resolved
6	`LAST_ERROR` storage can only be queried raw	Informational	Resolved
7	Insufficient description for unimplemented endpoints	Informational	Acknowledged
8	Custom access controls implementation	Informational	Acknowledged
9	Non-compatible parameters accepted upon configuration update	Informational	Resolved
10	Unnecessary conversion to lowercase in addresses	Informational	Resolved
11	“Migrate only if newer” pattern is not followed	Informational	Resolved

Detailed Findings

1. Incorrect permissioning of `IbcExecuteProposal` execution leads to failure of proposal execution and elevated owner privileges

Severity: Major

When governance proposals are to be executed on another chain using IBC, the `assembly` contract creates an `IbcExecuteProposal` message, which the `controller` contract executes, see `contracts/controller/src/contract.rs:52-75`.

This is problematic because proposals submitted by the `assembly` contract would fail to execute as the validation in lines `57-59` attempts to verify the caller is the contract owner rather than the `assembly` contract. Additionally, the contract owner can execute arbitrary IBC messages without governance approval.

Recommendation

We recommend modifying `contracts/controller/src/contract.rs:57-59` to `config.assembly != info.sender` to ensure that only the `assembly` contract can execute IBC governance proposals.

Status: Resolved

2. No validation of IBC message timeout

Severity: Minor

The length of IBC message timeouts is defined during the instantiation and update of `controller` and `satellite` contracts in:

- `contracts/controller/src/contract.rs:36`
- `contracts/satellite/src/contract.rs:44`
- `contracts/satellite/src/state.rs:55`

However, there is no validation of this timeout value, which could lead to timeouts being defined that are of zero value or very high, up to `u64::MAX`. This may prevent the execution of cross-chain interactions as messages may expire before they can be handled or, in the case of very high timeout values messages may remain active indefinitely.

We classify this issue as minor since it can only be caused by the contract owner.

Recommendation

We recommend validating the timeout value to ensure it is greater than zero and less than an appropriate maximum value.

Status: Resolved

3. `assembly` contract owner can overwrite proposals

Severity: Minor

When creating a new IBC message packet in `contracts/controller/src/contract.rs:61-64`, the proposal ID is appended to the message and then stored.

However, no validation is performed to ensure that a proposal with this `proposal_id` has not previously been executed. This can lead to existing proposals being overwritten multiple times with different messages.

We classify this issue as minor since it can only be caused by the contract owner.

Recommendation

We recommend ensuring that the `proposal_id` being used to create an IBC message has not previously been used prior to the creation and storage of the proposal in `contracts/controller/src/contract.rs:69`.

Status: Resolved

4. Inconsistent expected format for main controller port

Severity: Minor

The `satellite` contract includes a `main_controller_port` string variable as part of its configuration. During the `instantiate` function in `contracts/satellite/src/contract.rs:40`, the value of it is formed by concatenating `wasm.` and the value of `main_controller`.

On the other hand, the configuration's `update` function takes the full `main_controller_port` string as input without automatically prefixing it with `wasm.`, as done in the `instantiate` function. Consequently, updating the configuration without including the `wasm` field would cause an invalid port id.

We classify this issue as minor since it can only be caused by the contract owner.

Recommendation

We recommend using the same expected format in both the instantiate and update functions. If the full `main_controller_port` string is used, its format should be validated to follow the specifications.

Status: Resolved

5. `RESULTS` storage does not save empty message proposals

Severity: Minor

In `contracts/satellite/src/ibc.rs:127-141`, the reply handler would only get called if the messages to execute are not empty. This means if an empty message proposal is sent, the `RESULTS` storage state will not save the `proposal_id` as seen in `contracts/satellite/src/contract.rs:56`.

Since the comment in `contracts/satellite/src/state.rs:63` hints the storage state can be considered as a flag to check that proposal was executed, this might confuse callers as the `ProposalState` query message will return an error for the empty-message proposal. However, the `assembly` contract in the governance repository will store the proposal status as `ProposalStatus::Executed`, which is inconsistent with the `satellite` contract.

Even though this issue currently only affects queries, we classify this issue as minor since state inconsistencies may have severe consequences in the future if the contracts are extended.

Recommendation

We recommend saving the empty message proposal to the `RESULTS` storage state.

Status: Resolved

6. `LAST_ERROR` storage can only be queried raw

Severity: Informational

In `contracts/controller/src/ibc.rs:153`, the `LAST_ERROR` storage state is stored in case the IBC acknowledgment returns an error. Since there is currently no custom query message that returns the stored error value, callers are forced to use a raw query by directly accessing the value stored for the `last_error` storage key as binary data, increasing the complexity of external queries for both users and smart contracts.

Recommendation

We recommended exposing a custom query message that returns the `LAST_ERROR` value.

Status: Resolved

7. Insufficient description for unimplemented endpoints

Severity: Informational

Both the `controller` and `satellite` contracts call the `unimplemented!` macro in some of their IBC entry points in `contracts/controller/src/ibc.rs:69`, `contracts/satellite/src/ibc.rs:149`, and `158`. The error returned by this macro does not give much information, resulting in a potentially confusing experience for the user.

On a side note, the provided diagram of the Astroport IBC architecture includes a two-sided arrow between the `controller` and `satellite` contracts. However, the `ibc_packet_receive` function on the `controller` is set as `unimplemented!`, which looks counterintuitive given the diagram.

Recommendation

We recommend returning descriptive error messages on the entry points that are required to be empty.

Status: Acknowledged

8. 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

9. Non-compatible parameters accepted upon configuration update

Severity: Informational

In `contracts/satellite/src/state.rs:32-40`, the `satellite` contract's configuration `update` function accepts both the `gov_channel` and `accept_new_connections` parameters as options. If the second is submitted, it will set `gov_channel` to `None`. However, if both are submitted, `accepts_new_connections` will always take precedence, effectively overwriting the submitted `gov_channel` without warning.

Although not a security issue, it can be confusing for users and error-prone, as they may submit both options by mistake and not realize that one of the values is ignored.

Recommendation

We recommend raising an error if the `gov_channel` option is `Some()` and the `accepts_new_connections` is `Some(True)` at the same time instead of just forcing precedence without an error.

Status: Resolved

10. 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

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

Severity: Informational

The contracts within scope are currently migrated without regard to their version. The pattern can be improved by adding validation to ensure that the migration is only occurring if the version supplied is newer.

Recommendation

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

Status: Resolved