Audit Report

Drop Contracts Extension

v1.0

August 12, 2024

Table of Contents

License

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Neutron Audit Sponsorship Program to perform a security audit of an extension to the Drop 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 target:

Repository	https://github.com/hadronlabs-org/drop-contracts
Commit	`debf253e3e7783beff815f570361fb17a01873ee`

Scope	The following contracts are in scope:
	
	`.`
	`└── contracts`
	`    └── staker`

Fixes verified at commit	`9ac134cdcd7b1d46fb759b58f90f835623ca5e17`

	Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Drop is a liquid staking protocol designed for deployment on the Neutron chain within the Cosmos ecosystem.

The protocol leverages Inter-Blockchain Communication (IBC), Interchain Accounts (ICA) and Interchain Queries (ICQ) to facilitate seamless staking and unstaking across the Cosmos ecosystem.

Drop integrates an autocompounding feature, automatically restaking rewards to optimize yield.

This is an extension audit covering the additional staker contract and its integration with the other contracts.

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	**High**	The protocol encompasses several contracts that communicate with each other through message exchanges and callbacks, in addition to utilizing IBC messages, ICA accounts and ICQ queries.
Code readability and clarity	**Medium-High**	-
Level of documentation	**High**	The client provided very detailed documentation comprehensive of diagrams and architectural design.
Test coverage	**Low**	`cargo tarpaulin` reports a `33.06%` test coverage.

Summary of Findings

No	Description	Severity	Status
1	The protocol can get permanently stuck in the `StakingRewards` state due to unhandled errors	Critical	Resolved
2	Accumulation of non-staked funds in the ICA account poses depegging risks	Major	Acknowledged
3	Missing validation in staker contract instantiation and update	Minor	Partially Resolved
4	Neutron governance updating IBC fees could temporarily hinder interchain transactions	Minor	Resolved
5	Additional funds sent for ICA registration are stuck in the contract	Minor	Acknowledged
6	Unbondings could fail due to asynchronous behavior of the `staker` contract	Minor	Externally Reported
7	“Migrate only if newer” pattern is not followed	Informational	Partially Resolved
8	Generic variable names	Informational	Resolved

Detailed Findings

1. The protocol can get permanently stuck in the `StakingRewards` state due to unhandled errors

Severity: Critical

In `contracts/core/src/contract.rs:838-861`, the `execute_tick_staking_rewards` function within the core contract acts as the designated handler when the `ContractState` transitions to `StakingRewards` during routine procedures.

However, if during this process the `puppeteer` contract callback to the `PuppeteerHook` stores a `ResponseHookMsg::Error` because of a timeout or an error in the ICA transaction, the `execute_tick_staking_rewards` function consistently returns a `PreviousStakingWasFailed` error in line `846`.

Consequently, lacking an alternative handler to progress the `ContractState`, the routine procedure becomes unable to proceed, resulting in the protocol becoming permanently stuck.

Recommendation

We recommend implementing a mechanism similar to the one defined in the `staker` contract to prevent the state machine from becoming stuck due to IBC errors or timeouts.

Status: Resolved

2. Accumulation of non-staked funds in the ICA account poses depegging risks

Severity: Major

If the staking operations of the staker contract fail, user funds accumulate in the interchain account. This accumulation continues indefinitely, regardless of the total balance or the frequency of failed staking operations.

However, extended periods of failed operations could lead to the depegging of the liquid staking token, as non-bonded tokens would become unrecoverable or redeemable much later than the conventional bonding period suggests. This delay is due to the extended non-bonding time.

Significant accumulation of non-staked tokens or consecutively failing staking operations might be interpreted as a selling signal by users of liquid staking derivatives, potentially causing a depeg.

Recommendation

We recommend implementing emergency procedures, such as sending balances back to users, triggered by a high absolute or relative value of non-staked funds or continuously failing staking operations.

Status: Acknowledged

3. Missing validation in staker contract instantiation and update

Severity: Minor

In `contracts/staker/src/contract.rs`, during the execution of the `instantiate` and `execute_update_config` functions of the staker contract, data provided by the owner is stored in the contract without any validation.

However, it should be validated to not allow for incorrect configurations.

Recommendation

We recommend validating input data before storing it in the contract, specifically:

- `denom`, `port_id`, `transfer_channel_id`, and `connection_id` should be queried from the core contract
- `timeout` should be validated to be within reasonable bounds
- `allowed_senders` should be deduplicated, validated to only contain valid addresses, and size limited to prevent out-of-gas errors
- `min_ibc_transfer` and `min_staking_amount` should be validated to be within reasonable bounds

Status: Partially Resolved

4. Neutron governance updating IBC fees could temporarily hinder interchain transactions

Severity: Minor

The staker contract constructs the `fee` object required by the Neutron’s `feeRefunder` and `interchaintxs` Cosmos SDK modules with fees defined by the owner.

However, since Neutron’s governance can update the required IBC fees, static fees could render contracts unable to execute transfers or interchain transactions if fees change.

Consequently, the aforementioned contracts may become stuck and unable to execute interchain transactions.

Recommendation

We recommend querying the current IBC fees directly from the `feeRefunder` and `interchaintxs` Cosmos SDK modules instead of hardcoding them.

Status: Resolved

5. Additional funds sent for ICA registration are stuck in the contract

Severity: Minor

In `contracts/staker/src/contract.rs:179-198`, the function `execute_register_ica` is utilized to register a new ICA account on a remote chain. This process mandates that the sender pays the `registration_fee` by transferring a defined amount of coins in the `LOCAL_DENOM` denomination.

However, additional funds sent by the user are not returned or processed further, effectively becoming stuck within the contract.

Recommendation

We recommend modifying the `execute_register_ica` function to strictly enforce the payment of only the required `registration_fee` by using the `must_pay` method or similar logic.

Status: Acknowledged

6. Unbondings could fail due to asynchronous behavior of the `staker` contract

Severity: Minor

The asynchronous nature of the `staker` contract can lead to situations where assets are requested for undelegation before they are fully staked.

In these scenarios, the `get_unbonding_msg` function defined in `contracts/core/src/contract.rs:11291183` should ignore the undelegation request and continue processing, such that once the assets are staked, the undelegation request can be successfully processed at a later time.

This issue has been independently reported by a third party.

Status: Externally Reported

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

Severity: Informational

The contract is 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 recommend following the “migrate only if newer” pattern defined in the CosmWasm documentation.

Status: Partially Resolved

8. Generic variable names

Severity: Informational

In `contracts/staker/src/contract.rs:203-310`, the `execute_stake` function utilizes variables named `sum` and `amount`.

These variable names are overly generic and do not provide clear information about their purpose or content. This lack of descriptive naming reduces the readability and maintainability of the codebase.

Recommendation

We recommend using meaningful variable names, such as `amount_to_stake` and `all_delegations`.

Status: Resolved