## Audit Report

# Drop Contracts Updates

v1.0

August 12, 2024

# Table of Contents

# License

# Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.



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 update 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 |
| Scope and commits | The scope is restricted to the changes applied in the following pull requests:<br>• https://github.com/hadronlabs-org/drop-contracts/pull/107 reviewed at commit `9d7a406f55aadbc40a8f055db676964aa0860daa`, base branch at `f8496c421b3630543369ff716427f574c27c5741`<br>• https://github.com/hadronlabs-org/drop-contracts/pull/87 reviewed at commit `b8cb35dccea1dfe48d9984e625a947aa0dfa0656`, |

base branch at `9f99cbe60ab898db291c78bd7aa995199e4df0e4`

- https://github.com/hadronlabs-org/drop-contracts/pull/90 reviewed at commit `22336456757511cfc87787305b0d56ebcde76679`, base branch at `642daa280ac232603e2c0d50897f3eb4caf9bb75`
- https://github.com/hadronlabs-org/drop-contracts/pull/106 reviewed at commit `56eea374a79ca4ef17083b3dbc62fe53924fd8df`, base branch at `bad1cf40ec4a85b5895c82abcfe3f97d2b620585`
- https://github.com/hadronlabs-org/drop-contracts/pull/92 reviewed at commit `97cd2c3c84ed79dbdd3b4e959b260ec97a5b7885`, base branch at `6a0ac2806235a9f30d71e42ffda50eed408ba6aa`
- https://github.com/hadronlabs-org/drop-contracts/pull/99 reviewed at commit `8a7e30a62e4fcc95f9d4986d9eb87fceaa4cee69`, base branch at `cf45f2e70a7968c73424c261ba9c28e02987e29c`
- https://github.com/hadronlabs-org/drop-contracts/pull/122 reviewed at commit `2ffdd441c186dba9f89bf08eb1c007116a746fdb`, base branch at `5a37b8f879e0b56b63153e7e9043d8f04e1ad395`
- https://github.com/hadronlabs-org/drop-contracts/pull/83 reviewed at commit `dde1052b08a974f8efdc41a07d4f465801df6dc6`, base branch at `dde1052b08a974f8efdc41a07d4f465801df6dc6`
- https://github.com/hadronlabs-org/drop-contracts/pull/86 reviewed at commit `d3942984d80cb89042ea0b59631a4918e8daa634`, base branch at `d708f82c7d0c4d393093da2de7b35c2b30423def`
- https://github.com/hadronlabs-org/drop-contracts/pull/89 reviewed at commit `2b8e47242cd5eb98801dfc449e780faa9af0b659`, base branch at `2b8e47242cd5eb98801dfc449e780faa9af0b659`
- https://github.com/hadronlabs-org/drop-contracts/pull/114 reviewed at commit `daf9b5378fde999eae922877985c43f68edd3e1e`, base branch at `f8300a94d3d618f128478a0a7accadac2650d997`
- https://github.com/hadronlabs-org/drop-contracts/pull/120 reviewed at commit `f89f3eed05de8ce63b8184fe6c1bf1f2d662fc08`, base branch at `c8169f992c9adc0f117344c50cb3be1ee64c8a75`

| | |
|---|---|
| | • https://github.com/hadronlabs-org/drop-contracts/pull/124 reviewed at commit `632739e732800e6ec3a78168e128578ddef8dfae`, base branch at `9b9e779d63e3f908ed07f8c5c5d71f64ade0ab62`<br>• https://github.com/hadronlabs-org/drop-contracts/pull/126 reviewed at commit `c33cd9c00076801a9d9dd7422f6f129666119dde`, base branch at `2042a047407e3c6d8787e210eae0ab1fa969ea11`<br>• https://github.com/hadronlabs-org/drop-contracts/pull/142 reviewed at commit `15c8b47b7ac72d5c846eb04f51e5f18e89f0a52c`, base branch at `df3fa3c977732df4e264b1bb049a0459ea3725ad` |
| Fixes verified at commit | `b1f986c9b7a45ab9dc21ee7c86e48bbf31fdc928`<br><br>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 update audit covering the changes in the aforementioned pull requests.

# 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 `28.13%` test coverage. |

# Summary of Findings

| No | Description | Severity | Status |
|---|---|---|---|
| 1 | Excessive grant duration, lack of revocation mechanism, and multiple delegate grantees | Minor | Acknowledged |
| 2 | Iterations over the `receivers` might run out of gas, failing all distributions | Minor | Acknowledged |
| 3 | Missing validation in the `splitter` contract | Minor | Partially Resolved |
| 4 | Centralization and misconfiguration risks in `splitter` contract | Minor | Acknowledged |
| 5 | Missing validation for the `timeout` parameter in the puppeteer contract | Minor | Resolved |
| 6 | Vulnerability in `curve25519-dalek` package | Minor | Acknowledged |
| 7 | Partial support to CW2 contract versioning standard | Informational | Resolved |
| 8 | Use of magic numbers decreases maintainability | Informational | Resolved |

# Detailed Findings

## 1. Excessive grant duration, lack of revocation mechanism, and multiple delegate grantees

**Severity: Minor**

In `contracts/puppeteer/src/contract.rs:567`, during the execution of the `execute_setup_protocol` function, a `GenericAuthorization` grant is created and executed on the ICA to allow the `delegate_grantee` to send `MsgDelegate` messages.

However, there are several issues with the current implementation:

- The grant is constructed with an expiration of 120 years and one month, an excessively long duration.
- There is no mechanism to revoke the grant, which could cause issues if the `delegate_grantee` account needs to be changed.
- It allows multiple `delegate_grantee` accounts to hold the grant simultaneously, which could lead to mismanagement and unauthorized actions.

### Recommendation

We recommend implementing the following changes:

- Reducing the expiration period of the `GenericAuthorization` grant to a more reasonable duration, adhering to best practices for security and authorization management.
- Introducing a mechanism to revoke the grant to allow changes to the `delegate_grantee` account when necessary.
- Ensuring that only one `delegate_grantee` can hold the grant at any given time to prevent multiple grants and potential misconfigurations.

**Status: Acknowledged**

### 2. Iterations over the `receivers` might run out of gas, failing all distributions

**Severity: Minor**

In `contracts/splitter/src/contract.rs:80-89`, the `execute_distribute` function iterates over all configured receivers.

If the number of receivers is too large, the operation may consume excessive gas, potentially making the `execute_distribute` functionality unusable due to gas exhaustion.

We are reporting this issue with minor severity since the cardinality of `receivers` is expected to be limited and only the contract owner has the ability to set them.

#### Recommendation

We recommend limiting the number of receivers or using batch processing to break down the distribution process into multiple transactions to ensure that the gas limit is not exceeded.

**Status: Acknowledged**

### 3. Missing validation in the `splitter` contract

**Severity: Minor**

In `contracts/splitter/src/contract.rs:21`, during the execution of the `instantiate` function, the `splitter` contract stores the `Config` without performing any validation.

Similarly, in `contracts/splitter/src/contract.rs:53`, the `ExecuteMsg::UpdateConfig` allows the `owner` to update the `splitter` contract `Config` without performing any validation.

Specifically, the `receivers` vector should be checked to ensure it does not contain duplicates and only includes valid addresses. Additionally, the amount defined for the fees should be verified to be less than `10000u128`.

#### Recommendation

We recommend implementing validation checks for the `Config` before storing it in the `splitter` contract.

**Status: Partially Resolved**

### 4. Centralization and misconfiguration risks in `splitter` contract

**Severity: Minor**

In `contracts/splitter/src/contract.rs`, the `splitter` configuration is highly versatile, allowing arbitrary configuration to route funds and set fee shares as they are collected.

This flexibility introduces a centralization concern, for example, if administrative errors occur.

Consequently, funds could be redistributed incorrectly since distributions are permissionless.

**Recommendation**

We recommend implementing stricter controls and validation mechanisms for configuring routes and setting fee shares to mitigate the risk of administrative errors and potential misconfigurations.

**Status: Acknowledged**

### 5. Missing validation for the `timeout` parameter in the puppeteer contract

**Severity: Minor**

In `contracts/puppeteer/src/contract.rs:90`, the `instantiate` function stores the `timeout` value without performing any validation.

This lack of validation can lead to issues if the timeout is set to zero or a relatively small value.

Specifically, if the timeout is too short, it can cause Inter-Blockchain Communication (IBC) transactions to fail and Interchain Account (ICA) channels to close prematurely.

**Recommendation**

We recommend adding validation to ensure that the timeout value is greater than zero. Setting a minimum acceptable timeout value would be advisable to prevent transaction failures and maintain the stability of ICA channels.

**Status: Resolved**

### 6. Vulnerability in `curve25519-dalek` package

**Severity: Minor**

The `curve25519-dalek` package version in use is vulnerable to timing variability, as reported in RUSTSEC-2024-0344.

*“Timing variability of any kind is problematic when working with potentially secret values such as elliptic curve scalars, and such issues can potentially leak private keys and other secrets. Such a problem was recently discovered in curve25519-dalek.*

*The Scalar29::sub (32-bit) and Scalar52::sub (64-bit) functions contained usage of a mask value inside a loop where LLVM saw an opportunity to insert a branch instruction (jns on x86) to conditionally bypass this code section when the mask value is set to zero”*

**Recommendation**

We recommend updating to the latest version of all the packages dependent on `curve25519-dalek`.

**Status: Acknowledged**

## 7. Partial support to CW2 contract versioning standard

**Severity: Informational**

In `contracts/splitter/src/contract.rs`, the `instantiate` function is responsible for initializing the contract.

However, it currently lacks a call to the `cw2::set_contract_version` function, which stores the actual `CONTRACT_VERSION`.

This omission can result in difficulties with contract version tracking and management.

**Recommendation**

We recommend modifying the `instantiate` function to call `cw2::set_contract_version` to store the current `CONTRACT_VERSION`.

**Status: Resolved**

## 8. Use of magic numbers decreases maintainability

**Severity: Informational**

In `contracts/factory/src/contract.rs:634`, a hard-coded number literal is used without context or a description. 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**