Security Audit Report

Drop Updates 2

v1.0

December 21, 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 GmbH has been engaged by Droplet Labs Ltd. to perform a security audit of Drop Contracts Updates 2.

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	`a1c1e7066a4253b5e3bf6d8e03bd8f008623e66d`
Scope	The scope of this audit is restricted to all the changes since our previous audit, which was performed at commit `b1f986c9b7a45ab9dc21ee7c86e48bbf31fdc928`.
Fixes verified at commit	`97b87a5dd628cfeeb8fa754fe7b6753e2cb90ad7` 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 auto-compounding feature, automatically restaking rewards to optimize yield.

This is an update audit covering the aforementioned changes.

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**, **Partially Resolved**, 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 detailed documentation including diagrams of the architecture.
Test coverage	**Low**	`cargo tarpaulin` reports a `44.35%` test coverage.

Summary of Findings

No	Description	Severity	Status
1	Attackers can manipulate `onTop` amount	Major	Acknowledged
2	IBC fees are not forwarded through contracts	Minor	Acknowledged
3	Lack of denomination segregation may cause miscalculation of bonded amounts	Minor	Resolved
4	Insufficient validations in bond provider management create configuration risks	Minor	Resolved
5	Missing validations in configuration can lead to incorrect IBC setup	Minor	Resolved
6	State machine can get stuck during remote balance synchronization	Minor	Acknowledged
7	Missing input deduplication	Minor	Acknowledged
8	Missing address validation in the Puppeteer contract	Minor	Resolved
9	Missing validations in Mirror contract initialization and configuration updates	Minor	Resolved
10	Missing validations in LSM and Native Bond Provider contracts initialization and configuration updates	Minor	Partially Resolved
11	Iterations over delegations might run out of gas and fail to bond	Minor	Acknowledged
12	Inconsistent validator set synchronization across contracts	Minor	Resolved
13	Gas exhaustion in case of large payload processing in Sudo messages reply handler	Minor	Resolved
14	Indefinite pausing allows for potential permanent contract lock	Informational	Acknowledged
15	Inconsistent naming of function and return value	Informational	Resolved
16	Decentralized address configuration and access controls increase operational complexity	Informational	Partially Resolved
17	Redundant `can_process_on_idle` query	Informational	Acknowledged

18	Inconsistent backup address requirement across bonding functions	Informational	Acknowledged
19	Unnecessary reply handlers	Informational	Resolved
20	Unfinished development	Informational	Acknowledged
21	Redundant code	Informational	Resolved
22	Usage of `panic` for handling errors is discouraged	Informational	Acknowledged

Detailed Findings

1. Attackers can manipulate `onTop` amount

Severity: Major

In `contracts/val-ref/src/contract.rs:71-109`, the `execute_bond_hook` function within the `val-ref` contract is triggered by the Core contract to increase the `onTop` amount for a specified validator based on referral.

A message is then created and forwarded to the `validators-set` contract to increase the validator's `onTop` allocation.

However, the current implementation lacks a decrement mechanism for the `onTop` value during the unstaking process.

This omission allows users to repeatedly bond and unbond, artificially inflating a validator's `onTop` value, which leads to a distorted view of the validator's actual stake.

Recommendation

We recommend implementing a decrement mechanism to appropriately reduce the `onTop` value during unstaking events.

Status: Acknowledged

The client acknowledges the issue and states that, while there is a possibility to abuse the stake distribution mechanics, several factors mitigate this concern:

- It is challenging to implement this process purely on-chain, necessitating off-chain curation.
- Decreasing the on-top stake during unstaking does not align with the intended purpose of the feature.
- Any significant abuse would be highly visible on-chain, and validators are unlikely to risk harming their reputation..
- A similar situation in Lido on Terra, where stakes were much higher, was never exploited to the best of the client's knowledge.

The client concludes that, although the problem exists, they prefer to observe how it develops while maintaining manual control to address any issues that arise.

2. IBC fees are not forwarded through contracts

Severity: Minor

In `contracts/core/src/contract.rs:267`, during the execution chain from `execute_tick` through to `execute_process_on_idle` in the bond provider, IBC fees are not properly propagated. When the bond provider sends an IBC message in `contracts/native-bond-provider/src/contract.rs:384`, the fees in `LOCAL_DENOM` are not forwarded in `info.funds` causing them to be paid by the protocol's Puppeteer contract instead of being covered by the user's initial transaction.

This is problematic because it requires the admin to deposit funds to prepaid the operations and may cause denial of service if not properly maintained leading also to unnecessary protocol expenses.

Recommendation

We recommend implementing a mechanism to pass IBC fees from the core contract through the entire execution chain to properly distribute network costs.

Status: Acknowledged

3. Lack of denomination segregation may cause miscalculation of bonded amounts

Severity: Minor

In `contracts/core/src/contract.rs:945-1040`, the `execute_bond` function allows users to bond their coins. During this process, the function verifies that the total bonded coin amount, stored in the `BONDED_AMOUNT` storage item, plus the current bonding amount does not exceed the defined `bond_limit`, and then updates the `BONDED_AMOUNT` with the new total.

However, since `BOND_PROVIDERS` supports multiple denominations, the coins processed by this function are not guaranteed to be of the same denomination.

As a result, saving the `BONDED_AMOUNT` without denomination segregation could lead to inaccurate calculations of bonded amounts.

Recommendation

We recommend modifying the `BONDED_AMOUNT` storage item to segregate values by denomination.

Status: Resolved

4. Insufficient validations in bond provider management create configuration risks

Severity: Minor

In `contracts/core/src/contract.rs:323-357`, the `execute_add_bond_provider` and `execute_remove_bond_provider` functions lack validations for bond provider management.

Specifically, multiple providers can be added for the same denom, providers can be removed even with non-zero balances, and there are no limits on the total number of providers. Additionally, duplicate bond providers can be added to the system.

This is a minor since only the contract owner can manage bond providers.

Recommendation

We recommend implementing the following validations:

- Ensure only one provider can be registered per denom and no duplicates can be registered.
- Validate that a provider's async tokens balance is zero before removal
- Implement a maximum limit on the total number of bond providers to prevent out-of-gas errors.

Status: Resolved

5. Missing validations in configuration can lead to incorrect IBC setup

Severity: Minor

In `contracts/mirror/src/contract.rs:126-148`, the `update_config` function performs minimal validation of new configuration values. The `source_channel`, `source_port`, and `ibc_timeout` parameters are stored directly without any validation. An incorrect `ibc_timeout` value could cause messages to timeout too quickly or too slowly, while invalid source_channel or source_port values could prevent successful IBC communication.

This is a minor issue since only the contract owner can update these configuration values.

Recommendation

We recommend implementing the following validations:

- Query the IBC module to verify the configuration of the `source_channel` and `source_port`.
- Validate that `ibc_timeout` is greater than zero and less than a reasonable maximum value.

Status: Resolved

6. State machine can get stuck during remote balance synchronization

Severity: Minor

In `contracts/core/src/contract.rs:791`, the condition to validate if the `msg.remote_height` is after the `balances_response.remote_height` can cause the state machine to get stuck in the peripheral state.

This occurs when the remote height fails to increase in subsequent ticks, which could happen if the remote chain is experiencing issues. While the protocol is designed to handle temporary remote chain unavailability, a persistent failure could leave the system in a stuck state requiring manual intervention.

We classify this as minor because it only occurs during remote chain downtime or upgrades, situations where the protocol cannot take meaningful action.

Recommendation

We recommend implementing an emergency pause of the protocol if the failure happens repeatedly.

Status: Acknowledged

7. Missing input deduplication

Severity: Minor

In `contracts/core/src/contract.rs:298-321`, the `execute_set_bond_hooks` function, which handles the `SetBondHooks` message, processes a list of addresses by performing validation and then stores the entries in the `BOND_HOOKS` storage item.

Similarly, in `contracts/validators-set/src/contract.rs:144`, the `execute_update_validators` allows storing validator addresses.

However, the current implementation lacks deduplication logic and does not enforce a maximum size limit for this list.

As this vector is accessed multiple times within the codebase, the lack of deduplication and size constraints may result in inefficiencies and potential performance bottlenecks.

Recommendation

We recommend implementing deduplication logic within the `execute_set_bond_hooks` and `execute_update_validators` functions to ensure that unique addresses are stored. Additionally, enforce a maximum size limit for the list to optimize performance.

Status: Acknowledged

8. Missing address validation in the Puppeteer contract

Severity: Minor

In `contracts/puppeteer/src/contract.rs:331`, the `native_bond_provider` address is stored without any validation.

Storing an unverified address could introduce risks, as invalid or unintended addresses may impact the contract's functionality.

Recommendation

We recommend implementing the following validations:

- Validate the `native_bond_provider` address.
- If the *Decentralized address configuration* issue is fixed, remove direct address storage and instead query addresses from a central registry contract

Status: Resolved

9. Missing validations in Mirror contract initialization and configuration updates

Severity: Minor

In `contracts/mirror/src/contract.rs:41-44` and `112-149`, both the `instantiate` and `execute_update_config` functions directly store configuration

parameters without validation. The `source_port`, `source_channel`, and `ibc_timeout` are stored without verification in both functions.

Invalid or non-existent IBC channels could prevent cross-chain communication from working correctly, while an improper timeout value could cause messages to fail prematurely or wait unnecessarily long.

Additionally, storing contract addresses directly rather than querying them from a central registry creates configuration management overhead. This is a minor issue since only the contract owner can set these values.

Recommendation

We recommend implementing the following validations:

- Query the IBC module to verify that `source_port` and `source_channel` exist and are active
- Validate that `ibc_timeout` is within reasonable bounds
- If the *Decentralized address configuration* issue is fixed, remove direct address storage and instead query addresses from a central registry contract

Status: Resolved

10. Missing validations in LSM and Native Bond Provider contracts initialization and configuration updates

Severity: Minor

In `contracts/lsm-share-bond-provider/src/contract.rs:40-63` and `257-321`, the `instantiate` and `execute_update_config` functions lack parameter validations.

Configuration parameters like `lsm_min_bond_amount`, `lsm_redeem_threshold`, `lsm_redeem_maximum_interval`, `timeout`, `port_id`, and `transfer_channel_id` are stored without validation. Invalid values for these parameters could lead to unexpected behavior.

For instance, a too-low `min_bond_amount` could cause unnecessary on-chain overhead, while an excessive `redeem_maximum_interval` could lock user funds longer than intended.

Additionally, as noted in the *Decentralized address configuration* issue, contract addresses should be queried from a central registry rather than stored independently. This is a minor issue since only the contract owner can set these values.

The equivalent issue is found in the native bond provider.

Recommendation

We recommend implementing the following validations:

- Validate `lsm_min_bond_amount` is above a reasonable minimum threshold
- Ensure `lsm_redeem_threshold` is greater than `lsm_min_bond_amount`
- Set upper and lower bounds for `lsm_redeem_maximum_interval`
- Validate `timeout` is within reasonable bounds
- Query the IBC module to verify `port_id` and `transfer_channel_id`
- If the *Decentralized address configuration* issue is fixed, remove direct address storage and instead query addresses from a central registry contract

Status: Partially Resolved

11. Iterations over delegations might run out of gas and fail to bond

Severity: Minor

In `contracts/distribution/src/contract.rs:190,245` the functions `calc_deposit_normal` and `calc_deposit_on_top` iterate over the `delegations` twice because of the newly introduced on-top values. Similarly, in `contracts/lsm-share-bond-provider/src/contract.rs:563` iterations are performed over delegations.

These iterations are unbounded and could run out of gas making the bonding process fail.

Recommendation

We recommend limiting the iterations and well-diversified validator sets. Therefore limiting the amount of validators and delegations is only a short-term remedy. In the long run, we recommend a pagination logic, that allows for rebalancing in multiple transactions.

Status: Acknowledged

12. Inconsistent validator set synchronization across contracts

Severity: Minor

The lack of synchronization between the `val-ref`, `validators-set`, and `validators-stat` contracts creates inconsistencies that may lead to transaction failures.

Specifically, a validator may be registered in the val-ref contract without being present in the `validators-set` contract, or validators could be reset in `validators-set` without corresponding updates in `validators-stat`.

These mismatches introduce the risk of operational disruptions, as pending transactions may fail until alignment is restored across these contracts.

Recommendation

We recommend implementing a strict synchronization mechanism to ensure consistent validator information across the `val-ref`, `validators-set`, and `validators-stat` contracts.

Status: Resolved

13. Gas exhaustion in case of large payload processing in Sudo messages reply handler

Severity: Minor

In `contracts/puppeteer-initia/src/contract.rs:790` and `contracts/puppeteer/src/contract.rs:960`, the `sudo_response` function manages the acknowledgement of sudo messages by processing provided binary `data`.

However, the gas limit of `1,000,000` imposed on Neutron for handling acknowledgements can be exceeded when processing messages with large payloads.

This issue is particularly evident when initial calls involve dynamic data structures, such as vectors, as seen in the `ClaimRewardsAndOptionallyTransfer` message.

The gas exhaustion reverts the transaction, necessitating re-execution with a smaller payload.

Recommendation

We recommend removing unneeded data processing in the `sudo_response` function to reduce gas consumption.

Status: Resolved

14. Indefinite pausing allows for potential permanent contract lock

Severity: Informational

The Core contract includes a pausing function intended as an emergency measure.

However, the current implementation lacks a maximum duration for this pause, allowing it to be set indefinitely.

Without a time restriction, the contract could remain paused indefinitely, resulting in a permanent lock that could prevent further contract interactions and operations.

Recommendation

We recommend modifying the pausing mechanism to include a time limit.

Status: Acknowledged

15. Inconsistent naming of function and return value

Severity: Informational

The function `query_total_async_tokens` in `contracts/core/src/contract.rs:123-135` returns the variable `total_async_shares`, which is also used for computations.

This is inconsistent and might be confusing to maintainers.

Recommendation

We recommend consistent naming of variables and functions i.e. renaming the `total_async_shares` to `total_async_tokens`.

Status: Resolved

16. Decentralized address configuration and access controls increase operational complexity

Severity: Informational

In the current architecture, allowed senders and contract addresses are managed independently across different contracts, each with slightly different validation implementations. This decentralized approach to access control and contract addressing creates operational overhead and increases the risk of configuration inconsistencies.

For instance, different core addresses could be configured across bond providers or a contract update might require changes to multiple locations.

Additionally, the varying implementations of sender validation across contracts reduce code maintainability and increase the likelihood of errors.

Recommendation

We recommend implementing a centralized "phonebook" contract that

- Acts as a single source of truth for all contract addresses in the system

- Provides a unified interface for querying and updating contract addresses
- Optionally: Defines and enforce the allowed message paths between contracts

Status: Partially Resolved

17. Redundant `can_process_on_idle` query

Severity: Informational

In `contracts/core/src/contract.rs:554-566`, the `can_process_on_idle` query is executed twice: once directly and again when `ProcessOnIdle` is called.

The second query is redundant since the state cannot have changed between the two calls within the same atomic transaction.

Recommendation

We recommend keeping the initial `can_process_on_idle` query in line `554` and removing the check performed within `ProcessOnIdle`.

However, if the architecture might change significantly in the future such that other contracts can trigger the `ProcessOnIdle`, a redundant query might provide additional security for maintainers.

Status: Acknowledged

18. Inconsistent backup address requirement across bonding functions

Severity: Informational

In `contracts/mirror/src/contract.rs:166`, the backup address is required and returns an error if not provided, while in `contracts/mirror/src/contract.rs:207` the `execute_bond` function does not enforce this requirement.

This inconsistency may cause user confusion and require additional transactions to set the backup address separately.

Recommendation

We recommend, either requiring the backup address in both functions or automatically setting `info.sender` as the backup address for a more consistent user experience.

Status: Acknowledged

19. Unnecessary reply handlers

Severity: Informational

In `contracts/core/src/contract.rs:1423-1437` and across the codebase, multiple CosmWasm reply handlers are defined without implementing any functional logic; they merely return an error if an error occurs.

This leads to inefficiencies and unnecessary complexity in the code.

Recommendation

We recommend removing these redundant reply handlers, this will improve code efficiency and reduce unnecessary overhead.

Status: Resolved

20. Unfinished development

Severity: Informational

In `contracts/validators-set/src/contract.rs:141`, there is a `TODO` comment, and in `contracts/puppeteer-initia/src/contract.rs:117`, code execution leads to an `unimplemented()` call. The presence of such development artifacts may cause confusion about intended functionality, and reduce code clarity and maintainability.

Recommendation

We recommend implementing the missing functionality or removing these development markers before deployment to production.

Status: Acknowledged

21. Redundant code

Severity: Informational

In `contracts/distribution/src/contract.rs:25`, an empty vector is initialized and immediately returned, which adds no functional value to the code and unnecessarily increases code complexity.

Recommendation

We recommend removing this redundant operation.

Status: Resolved

22. Usage of `panic` for handling errors is discouraged

Severity: Informational

In `contracts/distribution/src/contract.rs:138` and multiple other locations in the file, `assert!` macros are used for validation, which causes panic instead of gracefully handling errors.

This approach deviates from best practices and results in suboptimal error handling, as transactions fail without meaningful error messages.

Recommendation

We recommend replacing `assert!` statements with proper error handling by returning appropriate error types with descriptive messages.

Status: Acknowledged