Security Audit Report

Drop Updates 3

v1.0

March 20, 2025

Table of Contents

License

Disclaimer

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 Updates 3.

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-private
Commit	`a01ff8fa535db98bbf95af2d270910527f51765f`
Scope	The scope of this audit is restricted to all the changes since our previous audit, which was performed at commit `97b87a5dd628cfeeb8fa754fe7b6753e2cb90ad7`.
Fixes verified at commit	`686b17cac595414b61809f966390c921eaabc4c1` 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 `43.64%` test coverage.

Summary of Findings

No	Description	Severity	Status
1	Incompatible `ClaimRewards` message used in native puppeteer contract prevents claiming and transferring rewards	Major	Resolved
2	Unrestricted downgrade and version compatibility issues in migration handler	Minor	Acknowledged
3	Potential gas exhaustion due to unbounded delegation execution	Minor	Acknowledged
4	Lack of code ID validation and configuration enforcement in the factory	Minor	Acknowledged
5	Silent failure in `query_delegations` may lead to incorrect exchange rate calculation	Minor	Resolved
6	Missing `CanBeRemoved` implementation in native-sync-bond-provider contract	Informational	Resolved
7	Missing `Redelegate` implementation in puppeteer-native contract	Informational	Acknowledged
8	Missing contract metadata validation for `val_ref_address` in the factory contract	Informational	Resolved
9	Missing contract name validation in migration handler	Informational	Resolved
10	Surplus funds are not returned	Informational	Resolved
11	Missing validation for transfer recipient	Informational	Resolved
12	Unbounded iteration in delegation queries	Informational	Acknowledged

Detailed Findings

1. Incompatible `ClaimRewards` message used in native puppeteer contract prevents claiming and transferring rewards

Severity: Major

In `contracts/puppeteer-native/src/contract.rs:402-410`, the `execute_claim_rewards_and_optionaly_transfer` function sends the `ExecuteMsg::ClaimRewards` message to the distribution module contract that is maintained by Neutron.

However, the `ExecuteMsg::ClaimRewards` message interface does not match the contract's expected interface. Specifically, the optional receiver address of the rewards is supposed to be named `to_address`, instead of `receiver`.

As a result, the message will return an error, preventing claiming rewards.

Recommendation

We recommend adjusting the `ClaimRewards` message by renaming `receiver` to `to_address`.

Status: Resolved

2. Unrestricted downgrade and version compatibility issues in migration handler

Severity: Minor

In `contracts/core/src/contract.rs:1398-1452`, the `migrate` handler currently allows downgrades, even when there are storage format incompatibilities between versions.

For example, the key `"pause"` is not parsable to the previous `Pause` struct definition, rendering the downgraded store unreadable.

Attempting a downgrade in this state leads to failures, resulting in an unusable contract and corrupted state.

Additionally, the migration procedure is strictly dependent on the version being migrated from, but the handler only verifies that the original version is older than the current one.

It does not enforce compatibility with a specific version, meaning migrations from unsupported versions may introduce unintended behavior or data corruption.

Recommendation

We recommend implementing strict version checks to ensure migrations occur only from explicitly supported versions. The `migrate` handler should verify both version ordering and compatibility, rejecting any downgrade attempts and unsupported version transitions.

Status: Acknowledged

3. Potential gas exhaustion due to unbounded delegation execution

Severity: Minor

In `contracts/puppeteer-native/src/contract.rs:284-355`, the `execute_delegate` function processes a vector of tuples containing validators and their corresponding delegation shares, as provided by the strategy contract. It iterates through this vector and executes a `Delegate` message for each validator.

If the strategy contract supports all available validators, assuming a typical CometBFT setup with `100` validators, this could result in executing `100` delegation messages in a single transaction, potentially exceeding gas limits and causing transaction failure.

Recommendation

We recommend implementing a batching mechanism to prevent exceeding gas limits or enforcing a configurable maximum number of validators.

Status: Acknowledged

The client states that they do not plan to have more than 40 validators in the validator set at this time. Consequently, the number of unbonding delegations will not exceed 280.

4. Lack of code ID validation and configuration enforcement in the factory

Severity: Minor

In `contracts/factory/src/contract.rs:514-554`, the `validate_contract_metadata` function ensures that the contract at the specified `contract_addr` has a name containing the provided `valid_names` and is owned by the factory contract.

However, it does not verify the code ID, which is critical to ensure that the contract is the intended one and not just any contract with a matching name. Given that the `instantiate` function already tracks code IDs, incorporating this validation would be straightforward.

Additionally, the previous implementation enforced correctness and consistency in contract configuration by passing it directly from the factory contract.

However, the current implementation does not enforce proper configuration, potentially leading to misconfigured contracts.

Recommendation

We recommend adding code ID validation to `validate_contract_metadata` to ensure that only the intended contract is recognized. Additionally, the contract's configuration should be queried and verified after instantiation to enforce correctness and consistency.

Status: Acknowledged

The client states that during the protocol deployment process, they obtain code IDs from the upload script and incorporate them into the instantiation script configuration. This approach ensures accuracy and prevents errors with code IDs. Nevertheless, they plan to add this verification in the future.

5. Silent failure in `query_delegations` may lead to incorrect exchange rate calculation

Severity: Minor

In `contracts/puppeteer-native/src/contract.rs:111-151`, the `query_delegations` function can silently fail, returning an empty vector or partial data without indicating an error in case the inner Stargate query fails.

However, since the function's output is used to compute the exchange rate, incorrect or incomplete data could lead to incorrect price computations.

Recommendation

We recommend implementing strict error handling to detect and flag incomplete or unexpected query results instead of failing silently. The function should return an error when delegation data cannot be retrieved correctly, ensuring that incorrect calculations are avoided.

Status: Resolved

6. Missing `CanBeRemoved` implementation in native-sync-bond-provider contract

Severity: Informational

In `contracts/native-sync-bond-provider/src/contract.rs:68-71`, the `QueryMsg::CanBeRemoved` message is not implemented, preventing the bond provider from being removed.

While this could be intended, without proper handling, this results in a panic instead of providing a meaningful error response.

Recommendation

We recommend implementing `QueryMsg::CanBeRemoved` or returning an appropriate error message clearly indicating why the bond provider cannot be removed.

Status: Resolved

7. Missing `Redelegate` implementation in puppeteer-native contract

Severity: Informational

In `contracts/puppeteer-native/src/contract.rs`, the `Redelegate` message is not implemented, unlike in both the default and Initia versions of the puppeteer contract.

This omission limits the contract's ability to reallocate delegations between validators without first undelegating and then re-delegating, potentially leading to inefficiencies and decreased rewards.

Recommendation

We recommend implementing the `Redelegate` message in the puppeteer-native contract to maintain consistency with the other versions and enable seamless delegation adjustments without requiring an intermediate undelegation step.

Status: Acknowledged

8. Missing contract metadata validation for `val_ref_address` in the factory contract

Severity: Informational

The `validate_pre_instantiated_contracts` function in `contracts/factory/src/contract.rs:556-615` validates the contract metadata of the contracts specified in the `PreInstantiatedContracts` struct.

However, the validation for the optional `val_ref_address` contract is missing.

Recommendation

We recommend adding validation for the `val_ref_address` contract in the `validate_pre_instantiated_contracts` function.

Status: Resolved

9. Missing contract name validation in migration handler

Severity: Informational

In the following few instances, contracts are currently migrated without regard to their contract name, contrary to how the `migrate` function is implemented in most contracts that are in the scope of the review:

- `contracts/native-sync-bond-provider/src/contract.rs:298-312`
- `contracts/puppeteer-native/src/contract.rs:481-495`
- `contracts/validators-stats/src/contract.rs:452-470`

This can be improved by validating that both the stored contract name and the new `CONTRACT_NAME` constant match.

Recommendation

We recommend validating the old and the new contract names to ensure they are equal.

Status: Resolved

10. Surplus funds are not returned

Severity: Informational

In `contracts/puppeteer-native/src/contract.rs:301`, the contract checks if the attached amount is greater than or equal to the amount to stake but does not return any surplus funds to the user.

Consequently, when users send more native tokens than required for staking, the excess funds become stuck in the contract.

Recommendation

We recommend implementing a mechanism to return any surplus funds to the sender after the staking operation completes.

Status: Resolved

11. Missing validation for transfer recipient

Severity: Informational

In `contracts/puppeteer-native/src/contract.rs:390`, there is a comment indicating that the `to_address` should be sent to the withdrawal manager. However, no validation is implemented to enforce this requirement, allowing transfers to any recipient address.

Recommendation

We recommend adding validation to ensure the transfer recipient is the withdrawal manager or removing the misleading comment if the current implementation is intentional.

Status: Resolved

12. Unbounded iteration in delegation queries

Severity: Informational

In `contracts/puppeteer-native/src/contract.rs:179-212`, the `query_unbonding_delegations` function loops unboundedly, executing the `DelegatorUnbondingDelegations` query in batches of `500` elements at a time. If additional entries exist, the function continues iterating until all unbonding delegations are retrieved.

Given that `DefaultMaxEntries` is `7` per validator, and assuming a typical CometBFT setup with `100` validators, the worst-case scenario results in `700` unbonding delegations, which could lead to gas exhaustion and transaction failure.

Similarly, in `query_delegations`, unbounded iteration occurs as well, but here, the total number of iterations is naturally capped by the number of validators, making it less prone to excessive gas consumption compared to `query_unbonding_delegations`.

Recommendation

We recommend capping the number of validators where the contract can delegate.

Status: Acknowledged

The client states that they do not plan to have more than 40 validators in the validator set at this time. Consequently, the number of unbonding delegations will not exceed 280.