Audit Report

Nebula Protocol

August 8, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Philip Stanislaus and **Stefan Beyer**

Oak Security

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

Introduction

Purpose of this Report

Oak Security has been engaged by Terraform Labs to perform a security audit of the CosmWasm smart contracts of the Nebula Protocol (https://neb.finance/).

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/nebula-protocol/nebula-contracts

Commit hash: `cf040c0085e10fede56bdf77aa22ec5dd78689e1`

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 smart contracts implement a rebalancing investment fund akin to ETFs.

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**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Missing assets in `asset_amounts` argument of cluster burn function will lead to wrong asset redemption and penalty	Critical	Resolved
2	Removal of target assets from a cluster renders remaining tokens unaccessible	Minor	Resolved
3	Users will lose not supported native assets sent to a cluster when minting	Minor	Resolved
4	Duplicate asset targets in a pool will allow attackers to extract value	Minor	Resolved
5	No validation of initial cluster target weight sum to equal 100	Minor	Resolved
6	Usage of raw queries might break contracts in the future	Minor	Acknowledged
7	Staking reward distribution might fail	Minor	Acknowledged
8	Overwriting an airdrop merkle root will cause users being unable to claim updated amounts	Minor	Resolved
9	Storing an airdrop merkle root for a future stage may cause users being unable to claim updated amounts	Minor	Resolved
10	Protocol fee distribution can be blocked by opening many polls	Minor	Resolved
11	Withdrawal and staking of voting rewards fails if too many locked balance entries exist	Minor	Acknowledged
12	Withdrawal of voting tokens fails if too many locked balance entries exist	Minor	Acknowledged
13	Penalty contract reset relies on manual migration	Minor	Acknowledged
14	Voter weight could be set to a value greater than 1	Minor	Resolved
15	Unused staking and gov contract functionality	Informational	Resolved
16	Minting in an empty cluster will not charge fee	Informational	Acknowledged

17	Last updated values in terraswap-oracle and nebula-dummy-oracle are set to max u64 value	Informational	Acknowledged
18	Vector math functions do not consistently handle different vector lengths	Informational	Resolved
19	Invalid merkle roots can cause panics during airdrop claims	Informational	Resolved
20	Updated quorum value not validated	Informational	Resolved
21	Updated threshold value not validated	Informational	Resolved
22	Overflow checks not set for profile release in `libraries/basket-math/Cargo.toml`	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of Documentation	Medium	-
Test Coverage	Medium-High	-

Detailed Findings

1. Missing assets in `asset_amounts` argument of cluster burn function will lead to wrong asset redemption and penalty

Severity: Critical

In the `try_receive_burn` function in `contracts/nebula-cluster/src/contract/handle.rs:96`, a vector is filled with the amounts to be redeemed from the `asset_amounts` argument. If that argument vector does not contain all the assets that are in the cluster's target asset data, the resulting vector will be shorter. The `query_redeem_amount` call will then contain `asset_amounts` that do not match the other asset vectors, e. g. `inv`, `prices` and `target`. That will cause the `compute_redeem` function of the penalty contract in `contracts/nebula-penalty/src/contract.rs:153` to calculate the wrong redemption amounts, since it assumes that all asset vectors supplied have the same length and order. Such a wrong calculation will result in the wrong redemption and also an incorrect penalty.

Recommendation

We recommend changing the vector assignment logic in `contracts/nebula-cluster/src/contract/handle.rs:96` from a push to the end of a vector to a direct assignment by index into a vector of the expected length, as is done in the `try_mint` function in line `461`.

Status: Resolved

2. Removal of target assets from a cluster renders remaining tokens unaccessible

Severity: Minor

The `try_reset_target` function in `contracts/nebula-cluster/src/contract/handle.rs:237` allows the contract owner and the composition oracle to remove assets from the target list of the cluster. In the `try_receive_burn` function in `contracts/nebula-cluster/src/contract/handle.rs:99`, the amounts to be burned are filtered by the current list of token targets. That means that any removed assets with non-zero balances can no longer be redeemed from the cluster.

Recommendation

We recommend disallowing removal of target assets that still have a non-zero balance.

Status: Resolved

3. Users will lose not supported native assets sent to a cluster when minting

Severity: Minor

Currently, the mint function in `contracts/nebula-cluster/src/contract/handle.rs:423` does not return an error or refund any native tokens if native tokens are sent that are not included in the cluster's target asset list. That can lead to users losing funds. We classify this issue as minor since it is caused by user error.

Recommendation

We recommend returning an error if not supported native assets are sent to the mint function.

Status: Resolved

4. Duplicate asset targets in a pool will allow attackers to extract value

Severity: Minor

There is currently no check in the cluster's `init` function in `contracts/nebula-cluster/src/contract/init.rs:9` and in the `try_reset_target` function in `contracts/nebula-cluster/src/contract/handle.rs:237` for duplicate entries in the `assets` argument. This could lead to several issues:

1. No matter whether the user supplies one or more entries with the duplicated denom in the `asset_amounts` array of either the `try_mint` or the `try_receive_burn` function, only the first entry will be used, all others will be ignored.
2. In the case of a duplicate native token, a user could send just one entry in the `asset_amounts` argument to the `try_mint` function, but there will be a duplicate subtraction of the token amount from the inventory in `contracts/nebula-cluster/src/contract/handle.rs:478`. When that happens, the reward from the penalty contract calculated in `contracts/nebula-penalty/src/contract.rs:134` will stay the same (since the imbalance just shifts to the right). The denominator in line `137` will get smaller though, which will lead to a higher minted token amount for the attacker. If the attacker immediately burns the cluster tokens, they can drain assets from the cluster.

The `assert_sent_native_token_balance` in `contracts/nebula-cluster/src/contract/handle.rs:464` does not prevent this issue since it checks the amount in every iteration, and not the sum of the amounts across all tokens with the same denom.

We classify this issue as minor since it is caused by a wrong parameterization which can only be changed during initialization, by governance or the composition oracle.

Recommendation

We recommend checking the `assets` argument in the `init` and `try_reset_target` functions for duplicates and returning an error if any duplicates are found.

Status: Resolved

5. No validation of initial cluster target weight sum to equal 100

Severity: Minor

There is currently no validation that checks that the sum of all target weights equals 100 in the cluster `init` function in `contracts/nebula-cluster/src/contract/init.rs:44`. Such a validation is performed during the `try_reset_target` call in `contracts/nebula-cluster/src/contract/handle.rs:256` though.

Recommendation

We recommend running validation of the target weight sum during the `init` function as well.

Status: Resolved

All checks for weight sums have been removed from the codebase to allow more flexibility in target weights.

6. Usage of raw queries might break contracts in the future

Severity: Minor

In `contracts/nebula-cluster-factory/src/querier.rs:14` and `53` as well as in `contracts/nebula-gov/src/querier.rs:16`, raw queries are used. Using raw queries is problematic, since they tie the caller to a specific implementation of the target contract's storage layout, which might change in a future migration.

Recommendation

We recommend using well defined query interfaces instead of raw queries.

Status: Acknowledged

7. Staking reward distribution might fail

Severity: Minor

The `_compute_rewards` function of the factory contract uses an unbounded call for asset weights through the `read_all_weight` function in `contracts/nebula-cluster-factory/src/contract.rs:560`. After that call, the `DepositReward` message is sent to the staking contract, which internally also loops over all the assets in `contracts/nebula-lp-staking/src/contract.rs:87` and `contracts/nebula-lp-staking/src/rewards.rs:22`. As more and more assets are added to Nebula over time, this might lead to reward distribution becoming impossible due to out of gas errors.

Recommendation

We recommend profiling the amount of gas used for different numbers of assets. If a change is necessary, we recommend inverting the logic such that the factory contract only stores the amounts to be distributed and maintains an index of distributed rewards per asset, while the staking contract can be called with a specific asset as a parameter to collect undistributed rewards.

Status: Acknowledged

The Nebula team will monitor gas usage.

8. Overwriting an airdrop merkle root will cause users being unable to claim updated amounts

Severity: Minor

The `store_merkle_root` function in `contracts/nebula-airdrop/src/contract.rs:72` does not prevent overwriting merkle roots for existing stages. If a user has already claimed tokens and a new merkle root from a tree with updated amounts is stored for the same stage, the user will not be able to claim the updated amounts since a claim already happened at that stage.

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

Recommendation

We recommend either changing the claim behaviour to store the already claimed amount per user and stage and only transferring the unclaimed amount or disallowing overwrites of merkle roots for existing stages.

Status: Resolved

9. Storing an airdrop merkle root for a future stage may cause users being unable to claim updated amounts

Severity: Minor

The `store_merkle_root` function in `contracts/nebula-airdrop/src/contract.rs:72` does not prevent storing a merkle root for a future stage and does also not update the `latest_stage`. That allows the owner to set a merkle root for a stage in the future, which could be used in a claim by a user. That merkle root may be overwritten at some point by the `register_merkle_root` function. A user that has already claimed tokens for the stage will not be able to claim the amount from the updated merkle tree.

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

Recommendation

We recommend rejecting updates of merkle roots that are for a stage later than `latest_stage` or updating `latest_stage` to the newly stored stage.

Status: Resolved

10. Protocol fee distribution can be blocked by opening many polls

Severity: Minor

The `Distribute` message of the collector contract sends the `DepositReward` message to the gov contract. Within that contract, all polls are read in `contracts/nebula-gov/src/staking.rs:230`. That storage read is unbounded and could run out of gas, reverting the distribution. Theoretically, an attacker can block protocol fee distribution by opening many polls. Since opening polls has an economic cost and distribution can be re-triggered at any time by any user, this is a minor security concern.

Recommendation

We recommend modelling the cost of such an attack. If necessary, we recommend adjusting the economic parameters of polls to mitigate this attack, e. g. by increasing the cost per poll quadratically in the number of polls currently in progress.

Status: Resolved

Resolved by limiting the maximum number of polls in progress to 50.

11. Withdrawal and staking of voting rewards fails if too many locked balance entries exist

Severity: Minor

In the `get_withdrawable_polls` function in `contracts/nebula-gov/src/staking.rs:396`, an unbounded iteration over the entries in `locked_balance` is performed. In each iteration, two storage entries are read. If the transaction runs out of gas, the `WithdrawVotingRewards` and `StakeVotingRewards` messages would fail, and it would be impossible for a user to recover and withdraw their rewards.

This issue only affects individual users and the likelihood of the `locked_balance` list growing to the point of the described issue is very low. Hence we classify this issue as minor.

Recommendation

We recommend profiling the application to check how severe this problem is. If needed, the issue could be solved by adding another call to only withdraw/stake voting rewards for a specific poll. That would allow a user to recover.

Status: Acknowledged

The Nebula team will monitor gas usage.

12. Withdrawal of voting tokens fails if too many locked balance entries exist

Severity: Minor

The `WithdrawVotingTokens` message calls `compute_locked_balance`, which contains an unbounded iteration over the entries in `locked_balance` in `contracts/nebula-gov/src/staking.rs:214`. If too many entries exist in `locked_balance`, that iteration could run out of gas. Since the `WithdrawVotingTokens` message is the only place where the `locked_balance` is cleaned, a user could never recover from that, and could never get their staked tokens back.

This issue only affects individual users and the likelihood of the `locked_balance` list growing to the point of the described issue is very low. Hence we classify this issue as minor.

Recommendation

We recommend profiling the application to check how severe this problem is. There are different approaches to fix it, e. g. a separate message to remove locked balances, a change of the storage such that the largest locked balance is stored rather than computed, or more places where `locked_balance` is cleaned.

Status: Acknowledged

The Nebula team will monitor gas usage.

13. Penalty contract reset relies on manual migration

Severity: Minor

The `try_reset_penalty` function in `contracts/nebula-cluster/src/contract/handle.rs:328` does not contain any logic to migrate the EMA from the old penalty contract to the new one. If the EMA is wrongly set, reward/penalty calculations will be wrong. Manual migration of the EMA is necessary to prevent this.

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

Recommendation

We recommend reading the EMA from the old penalty contract and writing it to the new penalty contract as part of the `try_reset_penalty` function.

Status: Acknowledged

Automatic migration of the EMA would prevent migration to a penalty contract that might not use EMA characteristics.

14. Voter weight could be set to a value greater than 1

Severity: Minor

In the current implementation, the `voter_weight` could be set to a value greater than 1 in `contracts/nebula-gov/src/contract.rs:214`, which would imply that more protocol fee rewards are distributed to voters than have been deposited by the collector contract.

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

Recommendation

We recommend adding an assertion to validate that the `voter_weight` is set to a value smaller than or equal to 1 to prevent this issue.

Status: Resolved

15. Unused staking and gov contract functionality

Severity: Informational

Various functions and stored values within the staking and gov contracts are not used by the Nebula protocol. Examples:

- `premium_updated_time` in `contracts/nebula-lp-staking/src/state.rs:40`
- `premium_min_update_interval` in `contracts/nebula-lp-staking/src/contract.rs:30`
- Shorting is not used anywhere in Nebula, but implemented across the staking contract
- The `VotingPower` struct in `contracts/nebula-gov/src/state.rs:75`

Recommendation

We recommend removing unused logic and storage entries to reduce complexibility and the attack surface as well as decrease costs of deploying the contract.

Status: Resolved

16. Minting in an empty cluster will not charge fee

Severity: Informational

When an empty cluster is initialized by the first user minting in it, no fee is applied in `contracts/nebula-cluster/src/contract/handle.rs:564`. This is different from minting from an non-empty cluster, where a fee is charged in line `510`.

Recommendation

We recommend charging a fee for initializing an empty cluster as well.

Status: Acknowledged

Charging no fee on an empty cluster is intentional. It incentivize the cluster initializer to initialize with high amounts of assets, which is beneficial for the protocol.

17. Last updated values in terraswap-oracle and nebula-dummy-oracle are set to max u64 value

Severity: Informational

The `last_updated_base` and `last_updated_quote` values in the `terraswap-oracle` contract in `contracts/terraswap-oracle/src/contract.rs:85` as well as in the `nebula-dummy-oracle` contract in

`contracts/nebula-dummy-oracle/src/contract.rs:54` are both set to the max u64 value, which undermines the `stale_threshold` check in the `query_basket_state` function in `contracts/nebula-cluster/src/contract/query.rs:53`.

Recommendation

We recommend setting the `last_updated_base` and `last_updated_quote` field values to recent timestamps and adding corresponding test cases.

Status: Acknowledged

Fixed for nebula-dummy-oracle, but not yet for terraswap-oracle. terraswap-oracle is used in testnet currently, but the team committed to fix the issue before deploying the contracts in production.

18. Vector math functions do not consistently handle different vector lengths

Severity: Informational

The vector math functions in `libraries/basket-math/src/vector.rs` do not currently consistently handle multiple vector lengths. On the one hand, the `mul`, `add` and `sub` functions in line `15`, `27` and `31`, respectively, only process the elements until the shorter vector is consumed and then return the resulting vector of that shorter length. On the other hand, the `dot` function in line `7` does not return a shorter amount, but rather panic with an index out of bounds error.

Recommendation

We recommend to consistently always process the shorter amount of elements or panic or return an error.

Status: Resolved

19. Invalid merkle roots can cause panics during airdrop claims

Severity: Informational

Since the merkle root is not validated in the `register_merkle_root` and `update_merkle_root` functions in `contracts/nebula-airdrop/src/contract.rs`, decoding in the `claim` function in `contracts/nebula-airdrop/src/contract.rs:174` could panic.

Recommendation

We recommend validating the decodability of the merkle root before storing it in `register_merkle_root` and `update_merkle_root`.

Status: Resolved

20. Updated quorum value not validated

Severity: Informational

In `contracts/nebula-gov/src/contract.rs:190`, an update of the `quorum` value is not validated.

Recommendation

We recommend calling `validate_quorum(msg.quorum)?;` before updating the value.

Status: Resolved

21. Updated threshold value not validated

Severity: Informational

In `contracts/nebula-gov/src/contract.rs.rs:194`, an update of the `threshold` value is not validated.

Recommendation

We recommend calling `validate_threshold(msg.threshold)?;` before updating the value.

Status: Resolved

22. Overflow checks not set for profile release in `libraries/basket-math/Cargo.toml`

Severity: Informational

While set in all other packages, `libraries/basket-math/Cargo.toml` does not enable `overflow-checks` for the release profile.

Recommendation

We recommend enabling overflow checks in every package to prevent overflows leading to security concerns. We recommend enabling them even if no calculations are currently performed in the package to prevent any issues when the code is extended or refactored in the future.

Status: Resolved