Audit Report

Lido Finance stATOM on Cosmos Hub

v1.0

April 29, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Lido Finance to perform a security audit of stATOM on Cosmos Hub.

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/lidofinance/lido-cosmos-hub-contracts

Commit hash: `c5db038e7bf2d0d4169fb7b31c1a4f3b3a2467c8`

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

Lido Finance offers a liquid staking protocol that automatically reinvests ATOM staking rewards. Shares in the pool of staked ATOM tokens are represented by fungible stATOM tokens. The audited smart contracts are intended to be deployed on Cosmos Hub after CosmWasm smart contracts have been enabled.

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 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	Slashing distribution may create a tragedy of the commons where slashing is never applied and new users are disincentivized to enter the protocol	Critical	Resolved
2	Unbounded unbond history entries may cause all deposited funds stuck in the hub contract	Major	Acknowledged
3	Unbounded unbond wait entities may cause user funds to be stuck in hub contract	Major	Acknowledged
4	Missing validation of Lido fee rate may cause reward dispatch to fail	Minor	Resolved
5	Missing validation for stored validator and guardian addresses	Minor	Resolved
6	Unbounded guardians query may run out of gas	Minor	Resolved
7	Slashing is not accounted for in queries which may give wrong results and cause unexpected behavior	Minor	Resolved
8	Negative unbonded amounts are not being handled when calculating the withdraw rate	Minor	Resolved
9	Missing denomination check may become problematic in the future	Minor	Acknowledged
10	Users might lose unbonded funds to users that unbond in the next batch if the unbonding period of the hub contract is shorter than the one of the underlying blockchain	Minor	Acknowledged
11	Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented	Informational	Acknowledged
12	Ability to pause hub contract increases risks associated with compromised owner key	Informational	Resolved
13	Exchange rate is not updated after bonding rewards and may exhibit small variations due to rounding/truncation	Informational	Resolved
14	Storing exchange rate and total stATOM issued in hub's state is inefficient and complicates the codebase	Informational	Acknowledged

15	Outdated references to Terra implementation	**Informational**	**Resolved**
16	Duplicate check slashing message is inefficient	**Informational**	**Resolved**
17	Inaccurate logic conditions in delegation calculation can undermine the efficiency of the contracts	**Informational**	**Resolved**
18	Name and version of stATOM contract are incorrectly set up	**Informational**	**Resolved**
19	Remove unused code to improve contract size and readability	**Informational**	**Resolved**
20	Marking paused parameter in the Hub as an Option adds unnecessary complexity	**Informational**	**Resolved**
21	Incorrect query function name may negatively affect user and developer experience	**Informational**	**Resolved**
22	Ignoring negative case of signed integer subtraction is bad practice	**Informational**	**Resolved**

Code Quality Criteria

Criteria	**Status**	**Comment**
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	-
Test coverage	**Medium-High**	-

Detailed Findings

1. Slashing distribution may create a tragedy of the commons where slashing is never applied and new users are disincentivized to enter the protocol

Severity: Critical

The `calculate_new_withdraw_rate` function in `lido_cosmos_hub/src/unbond.rs:129` updates the historical `withdraw_rate` of a given batch to account for slashing events that happened between the unbond epoch and the current time (minus an `unbonding_period` delta time). In line `140` of `calculate_new_withdraw_rate`, each batch gets assigned a `batch_slashing_weight`, which is the proportion of the unbonded ATOM amount of a given batch to the total unbonded ATOM of all batches since the last `execute_withdraw_unbonded` execution. That implies, that no matter when the slashing occurred, slashing will affect all users that have unbonded but have not yet withdrawn.

This mechanic can create a "tragedy of the commons" situation, as waiting unstakers are incentivized to not withdraw, since their slashing amount will be reduced the more other users unbond in subsequent batches. Worst case, this can result in a deadlock where no one is incentivized to withdraw. Moreover, new stakers coming in when there are pending slashes will participate in those not yet applied slashes, disincentivizing new users to stake funds.

Due to the condition in `contracts/lido_cosmos_hub/src/unbond.rs:59`, external users cannot intervene to resolve the problem by triggering a new `withdraw_rate` calculation, since withdrawing is restricted to users that have pending withdrawals.

Recommendation

We recommend allowing anyone to process pending unbonding batches, for example by returning instead of reverting in `contracts/lido_cosmos_hub/src/unbond.rs:60`.

Status: Resolved

2. Unbounded unbond history entries may cause all deposited funds stuck in the hub contract

Severity: Major

The `execute_withdraw_unbonded` function contains unbounded loops which have the risk of running out of gas causing stuck irrecoverable funds in the hub contract. In `lido_cosmos_hub/src/contract.rs:102`, inside the `calculate_newly_added_unbonded_amount` function, there is an unbounded loop that iterates over the unprocessed `UnbondHistory` entries. In a similar manner, in line `203`, within the `process_withdraw_rate` function, another unbounded loop iterates over the same `UnbondHistory` entries that haven been just processed.

If there are no withdrawals for a long enough timeframe, or if the `epoch_period` is very short, there may be too many entries so this cannot be processed as it would run out of gas, leaving all funds stuck forever in the contract.

We do not classify this issue as critical since any user withdrawing unbonded funds will move the `last_processed_batch` forward, reducing the number of iterations the next withdrawing user needs to process.

Recommendation

We recommend enforcing a minimum `epoch_period` length and a maximum limit of unbond history entries. Alternatively, pagination could be added to this message.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that it is unlikely to cause problems until several years in the future. The team has added the issue to their backlog and plans to upgrade the contracts before it becomes problematic.

3. Unbounded unbond wait entities may cause user funds to be stuck in hub contract

Severity: Major

The `get_finished_amount` function contains an unbounded loop starting in `lido_cosmos_hub/src/state.rs:113` that iterates over each `UnbondWaitEntity` of a user. Removal of deprecated batches in the `remove_unbond_wait_list` function in line `67` is also unbounded.

If there are no withdrawals for a long enough timeframe, or if the `epoch_period` is short, there may be too many entries so this iteration would run out of gas, leaving the funds of the given user stuck in the contract.

The same issue could cause the `query_unbond_requests` function in `lido_cosmos_hub/src/contract.rs:483` and the `query_get_finished_amount` function in `contracts/lido_cosmos_hub/src/state.rs:137` to run out of gas, making the `WithdrawableUnbonded` and `UnbondRequests` query unusable.

We do not consider this issue to be critical, since the probability of a user experiencing this issue is fairly small. However, it’s theoretically possible and may happen when using bots and/or automated bonding/unbonding strategies.

Recommendation

We recommend enforcing a minimum `epoch_period` length and a maximum limit of `UnbondWaitEntity` per user. Alternatively, pagination could be added to the withdraw and query messages.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that it is unlikely to cause problems until several years in the future. The team has added the issue to their backlog and plans to upgrade the contracts before it becomes problematic.

4. Missing validation of Lido fee rate may cause reward dispatch to fail

Severity: Minor

The `lido_fee_rate` parameter in the Cosmos Reward Dispatcher contract determines the fee Lido applies to bonded Atom rewards. However, there is no validation that this value is smaller than `Decimal::one()` at contract instantiation (`lido_cosmos_rewards_dispatcher/src/contract.rs:39`) or in the `execute_update_config` function (line `113`). If the fee parameter is bigger than one, dispatch of rewards will fail due to an underflow in line `147`.

Recommendation

We recommend validating that the `lido_fee_rate` is equal to or smaller than `Decimal::one()`.

Status: Resolved

5. Missing validation for stored validator and guardian addresses

Severity: Minor

The list of addresses of validators stored in `REGISTRY` in `lido_cosmos_validators_registry/src/contract.rs:46` is not validated as `Addr` type. Likewise, addresses are not validated in the `add_validator` function in line `116`.

A similar issue is present in `lido_cosmos_hub/src/contract.rs:142` in the `execute_add_guardians` function, where the new `GUARDIANS` addresses stored are not validated as `Addr`.

Recommendation

We recommend validating that the addresses of both `REGISTRY` validators and `GUARDIANS` pass `Addr` validation to prevent human errors.

Status: Resolved

6. Unbounded guardians query may run out of gas

Severity: Minor

In `contracts/lido_cosmos_hub/src/contract.rs:409`, `query_guardians` iterates over all guardians in storage. Since there can be an unlimited amount of guardians in storage, this iteration can run out of gas. We mark this issue as minor since guardians can only be added by the owner.

Recommendation

We recommend setting a limit on the number of guardians that can be created, or adding pagination to the query.

Status: Resolved

7. Slashing is not accounted for in queries which may give wrong results and cause unexpected behavior

Severity: Minor

In `lido_cosmos_hub/src/contract.rs:451` the function `query_withdrawable_unbonded` returns the withdrawable amount for a given address. However, the latest `withdraw_rate` is not taken into account which is discounting any slashing events that may have occurred. Hence, this query may return incorrect results which may cause unexpected behavior on 3rd party applications integrated with Lido.

Recommendation

We recommend accounting for slashing and updating to the last `withdraw_rate` before calculating the withdrawal unbonded amount for a given address.

Status: Resolved

8. Negative unbonded amounts are not being handled when calculating the withdraw rate

Severity: Minor

In `contracts/lido_cosmos_hub/src/unbond.rs:163`, when calculating the new `withdraw_rate`, there is a signed subtraction that might return a negative value. If that were to happen, tokens would be sent to the users that the user should not receive. This could occur if the full amount were slashed, or if nearly the full amount is slashed due to rounding/truncation.

Even though this scenario is unlikely, it is theoretically possible as the Cosmos SDK allows a slashing fraction of 100%.

Recommendation

We recommend validating if the signed integer is negative, and defaulting to 0 in case it is.

Status: Resolved

9. Missing denomination check may become problematic in the future

Severity: Minor

Currently, the Lido Cosmos Hub contracts only support staking in the `underlying_coin_denom`, which cannot be updated. In `contracts/lido_cosmos_hub/src/contract.rs:341` there is a check to add the delegation amount of the `underlying_coin_denom`. However, other denominations that are delegated at the same time will be ignored. This could cause issues if the protocol were to add new staking tokens in the future. A similar missing denomination check is also present in `contracts/lido_cosmos_validators_registry/src/contract.rs:160` and `223`.

Recommendation

We recommend following best practices and checking the expected denominations and filtering out other staked tokens for future-proof contract logic.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that it is unlikely that the Cosmos Hub will add a new staking denomination in the near future. If this were to happen, they would prepare a specific update to resolve this issue.

10. Users might lose unbonded funds to users that unbond in the next batch if the unbonding period of the hub contract is shorter than the one of the underlying blockchain

Severity: Minor

Within the `execute_withdraw_unbonded` function, slashing is applied to an undelegation batch by adjusting the `statom_withdraw_rate` of that badge. This is done by comparing the expected undonded amount with the actual unbonded amount. The `actual_unbonded_amount` is calculated by subtracting the stored `prev_hub_balance` from the hub's current ATOM balance in `contracts/lido_cosmos_hub/src/unbond.rs:193`. The expected undbonded amount is then computed in the `calculate_newly_added_unbonded_amount` function by iterating over all unbonding history entries from the last processed/released one to the one that's older than `unbonding_period` param.

This works well if the `unbonding_period` param is equal to or greater than the underlying blockchain's unbonding period. If the `unbonding_period` param is too short though, the expected unbonded amount may be bigger than the actual unbonded amount, even if no slashing happens. In that case, the `statom_withdraw_rate` will be too low, and unbonding users lose funds to users in the next unbonding batch.

Despite the severe implications of a too short `unbonding_period` param, we classify this issue as minor, since governance can ensure equality of the periods by carefully making required changes.

Recommendation

If possible, we recommend querying the actual unbonding period from the underlying blockchain. Alternatively, we recommend monitoring any changes of the underlying blockchain's unbonding period and ensuring that the `unbonding_period` param is updated whenever changes are needed.

Status: Acknowledged

Lido Finance acknowledges this issue and states that the unbonding functionality will be replaced by a new voucher mechanism, to be audited in the future.

11. Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented

Severity: Informational

The current implementation of Lido finance does undelegations in batches for efficiency reasons. Undelegations happen at most every `epoch_period` through the logic in the `execute_unbond_statom` function at `lido_cosmos_hub/src/unbond.rs:289`.

That implies that users that have sent unbond messages to the hub contract will still be subject to slashing until the delegation batch is executed. This behavior is different from Cosmos SDK's slashing module, which only slashes delegators that were active when the slashing event occurred.

This difference is currently not documented.

Recommendation

We recommend documenting that undelegated funds in Lido finance will still be subject to slashing until the undelegation is executed, which can only happen after the current epoch.

Status: Acknowledged

Lido Finance acknowledges this issue and has updated the correspondent documentation.

12. Ability to pause hub contract increases risks associated with compromised owner key

Severity: Informational

The hub contains a `paused` param, which can be set/unset at any time by the owner to pause/unpause the contracts in `contracts/lido_cosmos_hub/src/contract.rs:177` and `196`. If the owner key is ever compromised or lost, funds may be left inaccessible forever in the contract.

We classify this issue as informational since a compromised owner key has other severe implications and proper key management is an underlying assumption of the protocol in any case.

Recommendation

We recommend using a time-lock that automatically expires instead of a boolean `paused` param. A time-lock has the benefit that the contracts will return to normal operation automatically after the lock has expired. That mitigates the issue of lost access over the owner key.

Status: Resolved

13. Exchange rate is not updated after bonding rewards and may exhibit small variations due to rounding/truncation

Severity: Informational

In `contracts/lido_cosmos_hub/src/bond.rs:94`, when the `BondType` is `StAtom`, the exchange rate is not updated. Since the new stATOM is minted in proportion to the exchange rate, it should remain identical. However, due to rounding/truncation, the exchange rate may be subject to small variations and these are not accounted for.

Recommendation

We recommend updating the exchange rate in the `BondType::StAtom` case as well to cover any small variations due to rounding errors.

Status: Resolved

14. Storing exchange rate and total stATOM issued in hub's state is inefficient and complicates the codebase

Severity: Informational

The hub contract's `State` contains a field `statom_exchange_rate` in `packages/basset/src/hub.rs:28`, which is updated in several places throughout the codebase, for example whenever rewards are accumulated. Since rewards are accruing on on an ongoing basis, the stored exchange rate will be almost instantly outdated. To use the correct value, the current implementation updates the stored exchange rate before every usage, including before a query of the `State`. Consequently, there is little point in storing the `statom_exchange_rate` in the `State` in the first place. Storing unnecessary data is inefficient and makes the codebase more complicated.

Similarly, the `State` contains a field `total_statom_issued` in line `26`. However, this is inefficient as it should be equivalent to the supply of stATOM, which is queried within the function call in `contracts/lido_cosmos_hub/src/bond.rs:71`.

Recommendation

We recommend removing both the `statom_exchange_rate` and the `total_statom_issued` fields from the `State` to make contracts more gas-efficient.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that they prefer to keep the `statom_exchange_rate` in storage for indexing purposes.

15. Outdated references to Terra implementation

Severity: Informational

The documentation states that any logic related to the Terra implementation should be marked as an issue. In this informational issue, we list comments and code that references the Terra implementation.

- All contracts and packages use version 0.16.0 of `cosmwasm-std`, which is a legacy version only used on Terra.
- Comment in `lido_cosmos_validators_registry/src/contrac.rs:156` referencing Terra Core.
- Outdated documentation title in `contracts/lido_cosmos_hub/README.md:1`.

Recommendation

We recommend using the latest stable version of CosmWasm related packages, or the one which will be used in the Cosmos Hub, and updating outdated Terra references in comments and documentation.

Status: Resolved

16. Duplicate check slashing message is inefficient

Severity: Informational

In the `Burn` message in `contracts/lido_cosmos_token_statom/src/handler.rs:51-62`, there is a duplicated slashing message in case the burn sender is not the `hub_contract`. The second `CheckSlashing` message does not have any new effect on the state, so it seems unnecessary.

Recommendation

We recommend removing the second check slashing message staring in line `56-62` to improve contract efficiency.

Status: Resolved

17. Inaccurate logic conditions in delegation calculation can undermine the efficiency of the contracts

Severity: Informational

In the `calculate_delegations` function in `lido_cosmos_validators_registry/src/common.rs:38`, if `coins_per_validator + extra_coin` is equal to `validator.total_delegated`, an unnecessary iteration over the loop will be made when bonding or removing a validator.

Recommendation

We recommend updating the condition to <= which won't skip any validator, making the code slightly more performant.

Status: Resolved

18. Name and version of stATOM contract are incorrectly set up

Severity: Informational

The stATOM cw20-compliant contract inherits its instantiate function from `cw20-base`, calling the `cw20_init` function in `lido_cosmos_token_statom/src/contract.rs:41`. This function will incorrectly store the `CONTRACT_NAME` to `crates.io:cw20-base` and `CONTRACT_VERSION` to `0.8.0`, instead of `crates.io:lido_cosmos_token_statom` and `1.0.0` respectively.

Recommendation

We recommend implementing a custom `cw20_init` function or overriding the contract name and versions to reflect the correct stATOM token contract values.

Status: Resolved

19. Remove unused code to improve contract size and readability

Severity: Informational

Unused code such as the `MAX_DEFAULT_RANGE_LIMIT` `const` in `lido_cosmos_hub/src/state.rs:35` increases contract bloat without providing any functionality. Another example is `enum` `ContractError` in `package/basset/src/contract_error.rs:5`.

Recommendation

We recommend removing any unused code snippets to improve the contract's size and readability.

Status: Resolved

20. Marking paused parameter in the Hub as an Option adds unnecessary complexity

Severity: Informational

The hub contract has a `paused` parameter switch, defined in `packages/basset/src/hub.rs:141`, that guardians and the contract owner can use in critical situations to disable most interactions with the contracts. However, the type of `paused` is `Option<bool>` instead of just `bool`, which adds extra unnecessary complexity.

Recommendation

We recommend implementing `paused` as a `bool` type, and setting it to `False` as default to replicate the current logic when paused is set to `None`, reducing unnecessary code complexity.

Status: Resolved

21. Incorrect query function name may negatively affect user and developer experience

Severity: Informational

The function `query_unbond_requests_limitation` in `contracts/lido_cosmos_hub/src/contract.rs:489` is referencing a different storage map `UNBOND_HISTORY_MAP` which may confuse users and may cause further issues if 3rd party developers query this function.

Recommendation

We recommend renaming the function to `query_history_limitation` to improve user and 3rd-party developer experience.

Status: Resolved

22. Ignoring negative case of signed integer subtraction is bad practice

Severity: Informational

The case of a negative result of the subtraction in `contracts/lido_cosmos_hub/src/unbond.rs:193` is silently ignored. While the value should never be negative, it is considered best practice to panic if it is.

Recommendation

We recommend either using unsigned types or adding an assertion for a positive value.

Status: Resolved