Security Audit Report

SatLayer LST

v0.5

April 10, 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 Mithril Labs Pte. Ltd. to perform a security audit of SatLayer LST.

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/satlayer/satlayer-lst
Commit	`02e6b8a12bdd84db50b243d20d988531cb727913`
Scope	All contracts were in scope.
Fixes verified at commit	`bf764588f3fd95b5edc1abf19773005f836ba4ec` 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

SatLayer LST implements a CosmWasm-based liquid staking protocol that interfaces with Cosmos validators, enabling users to deposit native tokens and receive liquid staking tokens (LST) with an algorithmically maintained exchange rate.

The system distributes delegations across registered validators according to a balanced distribution algorithm, processes unstaking requests in batches with an enforced unbonding period, and handles reward collection through a separate dispatcher contract that takes configurable protocol fees.

The protocol maintains internal accounting of delegations, processing potential slashing events through exchange rate adjustments while managing the lifecycle of staking, unstaking, and withdrawal operations through a series of state transitions and cross-contract message passing.

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	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Low	No documentation other than comments describing the logic was provided.
Test coverage	Low	`cargo tarpaulin` reports `1.43%` test coverage.

Summary of Findings

No	Description	Severity	Status
1	Unstaked tokens cannot be withdrawn from the `lst_staking_hub` contract	Critical	Resolved
2	Potential DoS in withdrawal processing due to unbounded loop	Critical	Resolved
3	The `lst_validators_registry` contract instantiation will always return an error	Major	Resolved
4	Improper `ExecuteMsg::Unstake` handling logic risks draining any `lst_staking_hub` contract LST balance	Major	Resolved
5	Uncapped fee rate allows potential funds diversion and blocks reward distribution	Minor	Resolved
6	Missing parameter bounds validation	Minor	Resolved
7	Query vs execution logic mismatch for withdrawable unstaked funds	Minor	Acknowledged
8	Inconsistent exchange rate update	Minor	Resolved
9	Arbitrary increment of slashed amount reduces user withdrawals	Minor	Resolved
10	Pause mechanism is a centralization risk	Minor	Resolved
11	Validator removal lacks tracking of failed redelegations	Minor	Resolved
12	Owner is a single point of failure	Minor	Acknowledged
13	Lack of sanity checks when reconciling the actual `lst_staking_hub` contract state	Minor	Acknowledged
14	Dangerous LST token address updates possibility	Minor	Resolved
15	Missing event attributes for validator management operations	Informational	Resolved
16	“Migrate only if newer” pattern is not followed	Informational	Resolved
17	Simplification of decimal arithmetic using `cosmwasm_std` math improves code maintainability	Informational	Acknowledged
18	Lack of role-based access controls for the pausing mechanism	Informational	Acknowledged

19	Outdated version convention	Informational	Resolved
20	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
21	Unused files and dead code occurrences	Informational	Resolved
22	Optimize functions by returning early	Informational	Resolved
23	Unnecessary access granted to a protected function	Informational	Resolved
24	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Unstaked tokens cannot be withdrawn from the `lst_staking_hub` contract

Severity: Critical

LST holders should be able to redeem their balances for the underlying staked token by issuing the `Cw20ExecuteMsg::Send` message to the `lst_token` contract and specifying the `lst_staking_hub` contract address along with an encoded `Cw20HookMsg::UnStake` message.

After receiving a `Cw20HookMsg::UnStake` message from the configured `lst_token` contract address, the `execute_unstake` function is called on line `contracts/lst_staking_hub/src/contract.rs:272` with the `sender` address parameter set to `MessageInfo::sender`. However, this address is not the original sender of the `Cw20ExecuteMsg::Send` message, but instead the address of the `lst_token` contract.

As a result, all undelegated funds will only be withdrawable by the `lst_token` contract, which has no facility for taking this action and distributing the tokens to the rightful owners.

Recommendation

The `Cw20ReceiveMsg::sender` field should be used as the `sender` parameter to `execute_unstake`. This ensures that the undelegated tokens are withdrawable by the correct user.

Status: Resolved

2. Potential DoS in withdrawal processing due to unbounded loop

Severity: Critical

In `contracts/lst_staking_hub/src/unstake.rs:198-225`, the `process_withdraw_rate` function contains an unbounded loop that processes all unreleased batch histories without pagination or gas limits.

The loop iterates through batches starting from `last_processed_batch + 1` until it reaches either a batch with timestamp greater than `historical_time`, an already released batch, or no more batches exist. If many epochs pass without withdrawals (especially with short epoch lengths), this could accumulate enough unprocessed batches to exceed block gas limits, permanently preventing withdrawals and locking user funds.

Recommendation

We recommend implementing a maximum processing limit by modifying the function to only process a capped number of batches per transaction (e.g., 50), while maintaining a cursor to allow subsequent transactions to continue from where processing stopped.

Status: Resolved

3. The `lst_validators_registry` contract instantiation will always return an error

Severity: Major

In `contracts/lst_common/src/lib.rs:20-28`, the `to_checked_address` function uses `Api::addr_validate`, which rejects any address whose HRP does not match the "regular" account prefix (e.g., `bbn` on Babylon). Since validator operator addresses use a different prefix (e.g., `bbnvaloper{…}`), the `instantiate` handler will always fail when `to_checked_address` is used on each validator address in `InstantiateMsg::validators` on `contracts/lst_validators_registry/src/contract.rs:46`.

Additionally, the `add_validator` handler would also always fail, as `to_checked_address` is called on the `Validator::address` field on line `85`.

Recommendation

We recommend using `QuerierWrapper::query_validator` to check that validator operator addresses are correct. If the query does not fail and returns a validator, the address is valid.

Status: Resolved

4. Improper `ExecuteMsg::Unstake` handling logic risks draining any `lst_staking_hub` contract LST balance

Severity: Major

The `lst_staking_hub` contract permits any account to invoke the `ExecuteMsg::Unstake` message. This is intended to allow unstaking via the CW20 Approve API, where the user requests the `lst_staking_hub` contract to burn their LST tokens from their balance using the `Cw20ExecuteMsg::BurnFrom` message.

However, when handling the message, the `execute_unstake` function is called on `contracts/lst_staking_hub/src/contract.rs:99`, which in turn issues the

`Cw20ExecuteMsg::Burn` message on `contracts/lst_staking_hub/src/unstake.rs:65-70`.

As a result, the contract attempts to burn the specified `amount` from its own LST token balance rather than the sender's. If the `lst_staking_hub` contract holds no LSTs, the burn fails and the transaction reverts. If the contract has a non-zero LST balance, any sender can burn that amount and receive withdrawal rights to the corresponding undelegated tokens.

Recommendation

When the `ExecuteMsg::Unstake` message is received, ensure the handler issues the `Cw20ExecuteMsg::BurnFrom` message to the `lst_token` contract. Consider checking that the sender's balance and the `lst_staking_hub` contract's allowance are sufficient before issuing the message in order to fail early with an informative error message with lower gas consumption.

Also consider adding another `ExecuteMsg` variant that allows the current batch to be processed (i.e., undelegated if the `epoch_period` has expired) without invoking the `lst_token` contract at all.

Status: Resolved

5. Uncapped fee rate allows potential funds diversion and blocks reward distribution

Severity: Minor

The `lst_reward_dispatcher` contract lacks validation for the `satlayer_fee_rate` parameter. This parameter determines what percentage of staking rewards is taken as protocol fees, but the contract accepts any value without upper bounds during both contract initialization and subsequent parameter updates.

When rewards are processed, the contract calculates the fee amount by multiplying the total rewards by this rate. If the rate exceeds 100% (represented as 1.0 in `Decimal`), the calculation would produce a fee larger than the available rewards. When the contract attempts to subtract this fee from the total rewards, an arithmetic overflow error occurs because the fee exceeds the total available amount.

Additionally, even if rates are set below 100%, a compromised owner could divert nearly all rewards meant for users to the fee recipient address.

Recommendation

We recommend implementing a maximum fee rate cap (such as 30%) and validating all fee rates against this cap during both contract instantiation and parameter updates.

Status: Resolved

6. Missing parameter bounds validation

Severity: Minor

The contract does not check bounds for `epoch_length` and `unstaking_period` in both instantiation and parameter updates at `contracts/lst_staking_hub/src/contract.rs:30-52` and update handlers.

A short `epoch_length` can cause unstaking transactions to fail when maximum unbonding entries are reached, while extreme values for either parameter could lead to serious operational issues.

Recommendation

We recommend adding reasonable max/min value constants and implementing the following validations:

- `unstaking_period <= MAX_UNSTAKING_PERIOD`
- `epoch_length <= unstaking_period`
- `epoch_length >= MIN_EPOCH_LENGTH`

Additionally, create an off-chain deployment script to validate parameters against the chain's staking module configuration since direct querying is not possible from contracts on Babylon.

Status: Resolved

7. Query vs execution logic mismatch for withdrawable unstaked funds

Severity: Minor

The `query_get_finished_amount` function at `contracts/lst_staking_hub/src/query.rs:93-108` only checks if the unstaking period has passed (`history.time < block_time`) but does not verify the batch's released flag that is checked in the execution logic at `contracts/lst_staking_hub/src/state.rs:68-78`. Additionally, the query does not account for possible slashing events that could affect the actual withdrawable amount, potentially causing users to see incorrect withdrawal estimates.

Recommendation

We recommend updating `query_get_finished_amount` to use the same conditions as the execution logic by checking the `released` flag instead of just comparing timestamps. Also, consider adding documentation that notes the returned value is an estimate that could be affected by slashing events.

Status: Acknowledged

Client states that is the expected approach. All the histories with `time > unstaking` period are potential candidates for withdrawal. But to make the actual withdrawal, the history has to be processed and released to get the updated withdrawal rate. Thus there is a difference in the check in the query and execution logic.

8. Inconsistent exchange rate update

Severity: Minor

In `contracts/lst_staking_hub/src/stake.rs:65-77`, there is an inconsistency in exchange rate updates between staking types. Regular staking (`StakeType::LSTMint`) only updates `total_lst_token_amount` without recalculating the exchange rate, while reward staking (`StakeType::StakeRewards`) both updates the amount and recalculates the exchange rate.

This can lead to a slightly stale exchange rate after regular staking operations due to rounding errors in integer division, even though theoretically, the rate should remain unchanged with proportional minting.

Recommendation

We recommend adding the `update_lst_exchange_rate()` call to the `LSTMint` case for consistency. While the practical impact is minimal (the exchange rate is only slightly lower than reality and self-corrects when staking rewards are accounted for), consistent behavior would improve code maintainability and prevent subtle bugs in future modifications.

Status: Resolved

9. Arbitrary increment of slashed amount reduces user withdrawals

Severity: Minor

In `contracts/lst_staking_hub/src/unstake.rs:299-305`, the `calculate_new_withdraw_rate` function arbitrarily adds 1 to the slashed amount when non-zero (`slashed_amount_of_batch += Uint256::one()`). This artificial increase of the slashing effect results in users receiving slightly less than their fair share during withdrawals.

The effect compounds across multiple withdrawals, gradually reducing the total amount users can withdraw. This appears to be an unnecessary safeguard against rounding errors, as Rust's integer math already floors division results, providing natural protection against overestimation.

Recommendation

We recommend removing the line that increments `slashed_amount_of_batch` by one, allowing users to receive their full fair share of withdrawals without artificial reduction.

Status: Resolved

10. Pause mechanism is a centralization risk

Severity: Minor

The protocol's pause mechanism at `contracts/lst_staking_hub/src/config.rs:65-88` and enforced in `contracts/lst_token/src/contract.rs:66-72` represents a significant centralization risk. A single admin can indefinitely freeze all protocol operations, including token transfers, with no timeout constraints or governance oversight.

Unlike similar systems like Cosmos Hub's TokenFactory that separate token transfer functionality from minting/burning operations during pauses, this implementation freezes all activity, including basic token transfers, potentially trapping user funds indefinitely.

Recommendation

We recommend implementing a tiered pause system that exempts basic token transfers from the global pause facility while including an automatic timeout that expires after a predefined period (e.g., 72 hours).

Status: Resolved

11. Validator removal lacks tracking of failed redelegations

Severity: Minor

In `contracts/lst_validators_registry/src/contract.rs:61-97`, the `remove_validator` function has a design flaw where it first removes a validator from storage and then attempts to redelegate funds. If funds cannot be fully redelegated due to cooling periods, slashing, or other restrictions, the function silently succeeds without redelegating any funds or recording the failure.

While not permanently lost (the contract owner can retry the removal later), funds remain delegated to validators not tracked in the registry until manual intervention occurs. This creates an inconsistent state and exposes delegations to additional risk without any notification mechanism.

Recommendation

We recommend modifying the validator removal function to maintain a "pending redelegations" list that tracks validators whose funds could not be fully redelegated, including details on amounts and timestamps, with a permissionless function to retry these redelegations when cooling periods expire.

Status: Resolved

12. Owner is a single point of failure

Severity: Minor

In `contracts/lst_staking_hub/src/config.rs:108-114`, the contract relies on a single owner address for all administrative functions, verified through the `is_authorized_sender` check. This creates a central point of failure and requires users to place full trust in a single entity that can unilaterally change critical parameters, pause the system, or update contract configurations without oversight or time delays.

Recommendation

We recommend deploying the contract with a multisig wallet or DAO as the owner address instead of an individual account.

Status: Acknowledged

Client states that the team has internal OpSec guidelines, best practices, and processes to ensure that the owner's address is secured. Contract admin (set for all contracts, a CosmWasm feature) can override migration. This allows owner recovery, preventing a single point of failure for the contract.

13. Lack of sanity checks when reconciling the actual `lst_staking_hub` contract state

Severity: Minor

Within `contracts/lst_staking_hub/src/contract.rs:185-195`, the `query_actual_state` function retrieves the current state from `STATE` and obtains actual current staking delegations via `deps.querier.query_all_delegations(env.contract.address)`. If the returned delegations list is empty, the function returns the stored state without verifying that `State::total_lst_token_amount` is zero.

This omission could result in the contract operating with an inconsistent state, for instance, if a single validator is repeatedly slashed or if undelegations occur that are for some reason not reflected in the stored `State::total_lst_token_amount`.

Recommendation

If the delegations list is empty, the contract should verify that `State::total_lst_token_amount` is also zero. In cases where it is not, it is advisable to implement a circuit breaker mechanism (e.g., pausing the contract) until the discrepancy is investigated and resolved.

Status: Acknowledged

The Client states that Babylon Genesis implements its own epoched staking module through `x/epoching` rather than using the default Cosmos SDK's `x/staking` module. Therefore, there can be cases where the `total_staked_amount` is non-zero but delegations can be empty. This state is expected behavior.

14. Dangerous LST token address updates possibility

Severity: Minor

The protocol allows the `lst_staking_hub` contract owner to freely update the LST token address at any time without restrictions or validations. This design choice is problematic as it might lead to potential integration issues, enabling a direct path to complete fund theft.

A compromised owner can exploit this vulnerability through an attack flow described below:

1. Some unstaking operations need to be triggered to ensure the contract has pending unstake requests with non-zero requested token amounts.
2. Update of the LST token address needs to be performed to point to a new, owner-controlled token contract.
3. Staking of a nominal amount of tokens to mint some of the new LST tokens has to be done, effectively gaining control of the entire token supply.
4. The attacker is unstaking their newly minted tokens, which would trigger the undelegation of all staked tokens from validators.

After the unbonding period, potential attackers could claim all funds from the protocol, effectively stealing the entire staked amount from legitimate users.

Recommendation

We recommend removing the ability to update the LST token address after the initial contract deployment. The LST token contract address should be set during initialization and treated as immutable thereafter.

Status: Resolved

15. Missing event attributes for validator management operations

Severity: Informational

In `contracts/lst_validators_registry/src/contract.rs`, the `add_validator` and `remove_validator` functions return responses without attributes detailing the operation that was performed.

Unlike other operations in the codebase that emit detailed attributes (like `update_params`, which include specific parameter values), validator management operations lack this emission.

Recommendation

We recommend adding appropriate attributes to `add_validator` and `remove_validator` functions.

Status: Resolved

16. “Migrate only if newer” pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the “migrate only if newer” pattern defined in the CosmWasm documentation.

Status: Resolved

17. Simplification of decimal arithmetic using `cosmwasm_std` math improves code maintainability

Severity: Informational

In `contracts/lst_staking_hub/src/math.rs`, the functions `decimal_division`, `decimal_multiplication`, and `decimal_multiplication_256` implement decimal arithmetic using manual scaling and conversions. The current implementations perform operations with custom multiplication and division using `Uint256` and a scaling factor, whereas they could be simplified by leveraging well tested functions from `cosmwasm_std`.

This simplification would reduce code complexity, lower the risk of precision or conversion errors, and result in a more robust and maintainable implementation.

Recommendation

We recommend refactoring these functions to utilize the corresponding `cosmwasm_std` math operations as demonstrated in Appendix A.

Status: Acknowledged

Client states that he agrees, and while the current implementation works as expected, the contracts will be updated in the next iteration to use simplified arithmetic for decimal scaling, allowing for further gas optimization.

18. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The `lst_staking_hub` contract implements a pausing mechanism, which is in line with best practices. However, all of the administrative functions of the contract are centralized in the admin role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn on and off the pausing mechanism.

Status: Acknowledged

Client acknowledges the recommendations and has operational processes to ensure multi-sig ownership when the Babylon Genesis mainnet is live.

19. Outdated version convention

Severity: Informational

In `Cargo.toml:7-8`, the package version is set to "0.0.1" and the edition to “2021”, which according to SemVer 2.0 conventions indicates extremely experimental/unstable code not ready for production use. This versioning may misrepresent the project's actual maturity level.

Recommendation

We recommend updating the version number to follow SemVer 2.0 conventions. For pre-release software, consider "0.1.0" to indicate initial development, or "1.0.0-alpha.1" for alpha testing. For production-ready software, use "1.0.0" or higher. Ensure the edition year is also updated.

Status: Resolved

20. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

Client states that contract admin (set for all contracts, a CosmWasm feature) has the omni-ability to override by migration; this logic is app-level. 2-step ownership transfer is mostly redundant for CosmWasm contracts with the admin set.

21. Unused files and dead code occurrences

Severity: Informational

In `contracts/lst_staking_hub/src/error.rs` and `contracts/lst_staking_hub/src/msg.rs`, the defined logic is never used throughout the codebase. These files contain error definitions and message structures that are not referenced by any other contract's logic.

Additionally, in `contracts/lst_common/src/errors.rs`, the `InvalidHookMsg` error variant is defined but never used in any error handling flow. Similarly, in `contracts/lst_common/src/lib.rs`, the `to_canonical_addr` function is defined but never called

Recommendation

We recommend either removing the unused files or integrating them into the active codebase if they're intended for future functionality. Additionally, we recommend removing or refactoring the unused error type and utility function to reduce maintenance burden and improve code clarity.

Status: Resolved

22. Optimize functions by returning early

Severity: Informational

The `calculate_delegations` function in `contracts/lst_common/src/delegation.rs:8-61` and `calculate_undelegations` on lines `63-124` can return early if the `amt_to_delegate` and `amt_to_undelegate` parameters respectively, are equal to zero.

The `calculate_new_withdraw_rate` on `contracts/lst_staking_hub/src/unstake.rs:290-336` can return the `withdraw_rate` parameter unchanged if the `slashed_amount` parameter is equal to zero, which will be the case most of the times the function is called.

Recommendation

We recommend refactoring the abovementioned functions to use early returns.

Status: Resolved

23. Unnecessary access granted to a protected function

Severity: Informational

In the `add_validator` handler on `contracts/lst_validators_registry/src/contract.rs:82`, the execution is authorized if the sender is the contract owner or the `lst_staking_hub` contract.

However, the `lst_staking_hub` contract does not have any functionality to issue the `ExecuteMsg::AddValidator` message to the `lst_validators_registry` contract.

Recommendation

We recommend removing the unnecessary access by only allowing the contract owner to issue this message. This would also make the access control consistent with the `ExecuteMsg::RemoveValidator` message handler.

Status: Resolved

24. Miscellaneous comments

Severity: Informational

- There is a spelling error in the error message for `HubError::RewardDispatcherNotSet` at `contracts/lst_common/src/errors.rs:65`. The word "dispatcher" is incorrectly spelled as "discpather".
- There is a typographical error in the function name `to_canoncial_addr` at `contracts/lst_common/src/lib.rs:30`. The correct spelling should be `to_canonical_addr` (swapped "i" and "c").
- In `contracts/lst_staking_hub/src/unstake.rs:83`, the comment preceding the `store_unstake_wait_list` function declaration incorrectly describes the implementation, as no `HashMap` with specified parameters orders is used.
- The comment on `contracts/lst_common/src/delegation.rs:54-55` is misleading, it reads as “it is impossible to completely delegate the amount” but this is incorrect.
- Documentation should be added to clarify that the handling of the `QueryMsg::State` and `QueryMsg::ExchangeRate` message uses the `query_actual_state` function in `contracts/lst_staking_hub/src/query.rs:37-39`, which is potentially not the same as the `STATE` storage item until the `check_slashing` function is called in an `ExecuteMsg` handler.
- Consider using a less misleading name for the `State::total_lst_token_amount` field defined on `contracts/lst_common/src/hub.rs:99`, for example, `total_delegations` better describes the value it is tracking.

Recommendation

We recommend resolving the abovementioned issues.

Status: Resolved

Appendix A: Suggestions

1. Alternative implementation of `decimal_division`

```
pub fn decimal_division(numerator: Uint128, denominator: Decimal) -> Uint128 {
    // convert the numerator to a Decimal256
    Decimal256::from_ratio(numerator, Uint256::one())
        // divide by the denominator
        .checked_div(denominator.into())
        // panic on divide by zero
        .unwrap()
        // convert result to Uint256, discarding the fractional part
        .to_uint_floor()
        // convert result to Uint128
        .try_into()
        // panic if conversion would overflow
        .unwrap()
}
```

2. Alternative implementation of `decimal_multiplication`

```
pub fn decimal_multiplication(a: Uint128, b: Decimal) -> Uint128 {
    // convert a to a Decimal256
    Decimal256::from_ratio(a, Uint256::one())
        // multiply by b
        .checked_mul(b.into())
        // panic if multiplication overflows
        .unwrap()
        // convert result to Uint256, discarding the fractional part
        .to_uint_floor()
        // convert result to Uint128
        .try_into()
        // panic if conversion would overflow
        .unwrap()
}
```

3. Alternative implementation of `decimal_multiplication_256`

```
pub fn decimal_multiplication_256(a: Uint256, b: Decimal256) -> Uint256 {
    // convert a to a Decimal256
    Decimal256::from_ratio(a, Uint256::one())
        // multiply by b
        .checked_mul(b)
        // panic if multiplication overflows
        .unwrap()
        // convert result to Uint256, discarding the fractional part
        .to_uint_floor()
}
```