Audit Report

pSTAKE Native Auto-Compounding and Rebalancing

v1.0

January 26, 2024

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 PSTAKE Sub One Ltd to perform a security audit of pSTAKE Native Auto-Compounding and Rebalancing.

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/persistenceOne/pstake-native
Commit	`b020194a6cb98e5c5756ff0e1d7fbe895372063d`
Scope	Only the `x/liquidstakeibc` module was in the scope of this audit.
Fixes verified at commit	`c7cc855e4cf2abd525f00a979981093f6bb27c5f` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

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 `x/liquidstakeibc` module enables base tokens on other chains to be liquid-staked via IBC and interchain accounts. Auto-compounding and rebalancing features are available to reinvest delegation rewards and rebalance delegated funds over time. Additionally, the Liquid Staking Module (LSM) allows users to skip the undelegation period and transform their existing delegations into stkATOM, achieving instant ATOM liquidity.

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 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 complexity is flagged as high due to multiple entry points with complex state transitions (e.g., epoch hooks, ICQ callbacks, and IBC packet flow) and integrations with custom modules implemented by the client, such as persistence-sdk, cosmos-sdk, and ibc-go.
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	Documentation is available in `x/liquidstakeibc/spec/README.md` and https://blog.pstake.finance/category/product/stkatom/.
Test coverage	**Medium-High**	go cover reports a test coverage of `75.2%`, excluding proto-generated files. For more information on how we compute the coverage, please see this section in the appendix.

Summary of Findings

No	Description	Severity	Status
1	Incorrect bookkeeping of validator's delegated amount upon redelegation	Critical	Resolved
2	Attackers can halt the chain by performing numerous minimal unstaking requests from different accounts	Critical	Resolved
3	Attackers can prevent users from liquid-staking funds by removing the `Deposit` entry	Critical	Resolved
4	Risk of gas exhaustion for unbounded ICA messages	Critical	Resolved
5	The `AfterEpochStart` hook consumes excessive computational resources	Critical	Acknowledged
6	Outdated validator exchange rates could result in inflated minting of liquid-staked tokens	Major	Acknowledged
7	The rewards account balance is not updated on `OnChanOpenAck`	Major	Resolved
8	The `BeforeEpochStart` hook consumes excessive computational resources	Major	Acknowledged
9	The `DoRedeemLSMTokens` function consumes excessive computational resources in the `BeginBlocker`	Major	Resolved
10	Validators are incorrectly set to non-delegable for default bond factor values	Major	Resolved
11	Incorrect comparison between shares and tokens	Major	Resolved
12	The bond factor is not set to the default value when registering host chains	Minor	Resolved
13	A significant decrease in `CValue` could disable the host chain and require a global parameters update	Minor	Resolved
14	`LowerCValueLimit` can be misconfigured as a negative value	Minor	Resolved
15	Incomplete `HostChain` validation	Minor	Resolved
16	The `RegisterHostChain` function allows to	Minor	Resolved

	overwrite existing `HostChain`		
17	Fees lack validation	Minor	Resolved
18	Matured undelegations are delayed by one block	Minor	Acknowledged
19	Unhandled errors in the codebase	Minor	Resolved
20	Host denom can be registered with multiple host chains	Minor	Resolved
21	The total validator weight is not validated to be 100%	Minor	Acknowledged
22	Genesis validation incorrectly fails for sent deposits	Informational	Resolved
23	Low `AutoCompoundFactor` results in unutilized funds	Informational	Acknowledged
24	Only the last broken invariants are reported	Informational	Resolved
25	Unnecessary module owner updates	Informational	Resolved
26	Unused error in the codebase	Informational	Resolved
27	Missing simulation features in the `liquidstakeibc` module	Informational	Acknowledged
28	Usage of magic numbers decreases maintainability	Informational	Resolved
29	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
30	Incorrect `existing_delegation` attribute emitted	Informational	Resolved
31	Relayers are not incentivized to relay ABCI-initiated IBC transactions	Informational	Acknowledged

Detailed Findings

1. Incorrect bookkeeping of validator's delegated amount upon redelegation

Severity: Critical

In `x/liquidstakeibc/keeper/ica_handlers.go:370`, the `HandleMsgBeginRedelegate` function updates the source and destination validators' delegated amount after `MsgBeginRedelegate` is successfully executed in the host chain.

The issue occurs in line `425`, where the total amount delegated for the source validator is incorrectly increased. This is problematic because redelegations will transfer the bonded tokens from the source to the destination validator. Hence, the total amount delegated for the source validator should be reduced, not increased.

Consequently, the `DelegatedAmount` state for the source validator will be inflated, preventing the protocol from working as intended. For example, the redelegation workflow in `x/liquidstakeibc/keeper/rebalance.go:27` relies on the validators' total delegated amount to compute the ideal rebalancing weight. Since the total delegated amount is inflated, the redelegated validators' weight will be incorrect.

Recommendation

We recommend modifying `x/liquidstakeibc/keeper/ica_handlers.go:425` to subtract the redelegated amount from the source validator.

Status: Resolved

2. Attackers can halt the chain by performing numerous minimal unstaking requests from different accounts

Severity: Critical

In `x/liquidstakeibc/keeper/abci.go:120`, the `DoClaim` function attempts to refund failed undelegation requests and distribute completed requests. This function is called in every block as part of the `BeginBlock` function in line `34`. Since there is no minimum amount for unstaking requests, external actors can influence the computation to slow down and potentially halt the chain.

For example, the attacker can acquire liquid-staked tokens and distribute them in small amounts to different accounts under their control. The attacker can then call `LiquidUnstake` messages from different addresses to generate a significant amount of `UserUnbonding` elements that want to undelegate a minimal amount. Once the undelegation matures, the `DoClaim` function must process all the undelegations

simultaneously in the `BeginBlock` function, causing the chain to halt after a block production timeout.

Exacerbating the issue, due to the truncated operation during the fee calculation in `x/liquidstakeibc/keeper/msg_server.go:636` and the small amount to be unstaked, the unstaking fee will not be charged.

Recommendation

We recommend implementing a minimum amount for `LiquidUnstake`, similar to `x/liquidstakeibc/keeper/msg_server.go:330`.

Status: Resolved

3. Attackers can prevent users from liquid-staking funds by removing the `Deposit` entry

Severity: Critical

In `x/liquidstakeibc/keeper/deposit.go:80-96`, the `AdjustDepositsForRedemption` function is called when a user invokes the `Redeem` message to reduce the deposited amount from the `Deposit` entry.

Specifically, the function retrieves deposits in the `Deposit_DEPOSIT_PENDING` state for the designated `HostChain` through the `GetRedeemableDepositsForHostChain` function in line `80`. The function then iterates through the deposits and tries to subtract the `redeemAmount` in line `88`. An edge case scenario is that if the `redeemAmount` equals the deposit amount, the redeemed amount is reduced, and the `Deposit` is deleted.

This is problematic because once the `Deposit` is deleted, subsequent `LiquidStake` messages will fail in `x/liquidstakeibc/keeper/msg_server.go:366`. Users must wait until the next epoch (i.e., the next day) so a new `Deposit` instance is created in `x/liquidstakeibc/keeper/hooks.go:89` to liquid-stake their funds.

An attacker can exploit this by calling `LiquidStake` at the beginning of the epoch to deposit little funds after the `Deposit` entry is created. After that, the attacker calls the `Redeem` message with a liquid-staked token amount such that, after fee deduction and division with the redemption rate, resulting in an amount that equals the deposited amount.

Consequently, the `Deposit` entry will be deleted in `x/liquidstakeibc/keeper/deposit.go:95`, halting all liquid staking operations until the commencement of the subsequent epoch that occurs the next day. The attacker can then repeat the same exploit in future epochs, effectively disabling liquid-staking operations for users.

Recommendation

We recommend modifying `x/liquidstakeibc/keeper/deposit.go:81` to be `depositsAmount.LTE(redeemAmount.Amount)`.

Status: Resolved

4. Risk of gas exhaustion for unbounded ICA messages

Severity: Critical

In `x/liquidstakeibc/keeper/ica.go:16`, the `GenerateAndExecuteICATx` function is utilized across various sections of the code to submit ICA transactions. The function takes a slice of `Message` elements intended for execution on the host chain, serializes them, and dispatches an ICA transaction.

However, there is no upper bound defined for the cardinality of messages that can be dispatched, with the function lacking any logic to limit their number. There is a potential risk of the relayer encountering an out-of-gas error in the host chain due to the significant amount of messages. This scenario is plausible in functions like the `DoRedeemLSMTokens` in `x/liquidstakeibc/keeper/abci.go:305`, which generates a `Message` for each `DepositLSM` and bundles them into a single ICA transaction.

Since ICA channels are `ORDERED`, future messages cannot be dispatched until the previous ones are completed. The `BeginBlocker` will then recreate it and try to dispatch the same ICA transaction, which will be a more significant number of messages due to the newly submitted `DepositLSM` elements, ultimately causing the transaction to fail again due to an out-of-gas error.

Recommendation

We recommend introducing mechanisms within the `GenerateAndExecuteICATx` function to impose limits on the number of messages per transaction to mitigate gas-related challenges and ensure the smooth execution of subsequent transactions by the relayer on the host chain.

Status: Resolved

5. The `AfterEpochStart` hook consumes excessive computational resources

Severity: Critical

In `x/liquidstakeibc/keeper/hooks.go:117-124`, the `AfterEpochStart` function is defined as a hook of the epoch module's `BeginBlocker`. It's essential to highlight that

this function operates within the `BeginBlocker`, necessitating a cautious approach to computational resource consumption.

However, it currently engages in extensive iterations, which could pose a risk of halting the chain.

Let's delve into specific concerns:

- The `DepositeWorkflow` function iterates through all `Deposits`, with an expected count of one for every `HostChain` plus reminders from previous epochs.
- The `LSMWorkflow` function iterates through all `LSMDeposits`, resulting in one iteration for every LSM deposit per `HostChain`.
- The `ValidatorUnderdelegationWorkflow` function is called for each `HostChain`, iterates all validators with `O(N^2)` complexity, and sends each of them an ICA transaction. This process can lead to significant computation and communication overhead.
- The `UnderdelegationWorkflow` function iterates through all `HostChains`, sending an ICA transaction for each.
- The `RewardsWorkflow` function is called for each `HostChain` and iterates through all validators, triggering an ICA transaction for each.
- The `RebalanceWorkflow` function is called for each `HostChain` and iterates through all validators, involving sorting and reversing operations on slices. This operation could impose a heavy computational load.

Similar to the previous issue, the cardinality of some of those slices depends on user interaction, opening the possibility for attackers to influence their length.

Consequently, the `BeginBlocker` execution will take significant time, potentially slowing block production and halting the chain.

Recommendation

We recommend not concentrating all the operations in a single `BeginBlocker` but executing the iterations in multiple batches.

Status: Acknowledged

The client states that they limited the amount of LSM deposits that can be processed at any stage of the flow, which is the only user-dependent way of increasing the resources consumed while processing every epoch.

6. Outdated validator exchange rates could result in inflated minting of liquid-staked tokens

Severity: Major

In `x/liquidstakeibc/keeper/msg_server.go:122`, the authority or admin can call the `UpdateHostChain` message specifying `KeyValidatorUpdate` to update the validator state after an ICQ callback.

One of the state updates is the exchange rate, which determines the rate of liquid-staked tokens. Any changes in the validator exchange rate (i.e., due to slashing) will not be reflected in the local chain status unless the authority or the admin calls the `UpdateHostChain` message periodically.

Consequently, this would cause the controller chain to use outdated exchange rates. For example, assume a host chain's validator gets slashed and the exchange rate decreases. The rate is not reflected immediately in the controller chain, resulting in users receiving more liquid-staked tokens than intended, causing a temporary loss of funds for existing liquid stakers.

Recommendation

We recommend allowing permissionless updates of the validator status and ensuring the exchange rate is not stale within a period.

Status: Acknowledged

The client states that the exchange rate is being updated via an off-chain process that sends `ValidatorUpdate` messages.

7. The rewards account balance is not updated on `OnChanOpenAck`

Severity: Major

In `x/liquidstakeibc/keeper/ibc.go:32`, the `OnChanOpenAck` function handles ICA open channel acknowledgments and uses the `QueryDelegationHostChainAccountBalance` function to update the balance of the delegation account through an ICQ callback.

However, the function does not update the balance of the rewards account. This would happen if the ICA channel is re-established, causing unutilized rewards to be stuck in the rewards account instead of being staked for the benefit of the users.

Recommendation

We recommend calling `QueryRewardHostChainAccountBalance` in `OnChanOpenAck`.

Status: Resolved

8. The `BeforeEpochStart` hook consumes excessive computational resources

Severity: Major

In `x/liquidstakeibc/keeper/hooks.go:86-97`, the `BeforeEpochStart` function is defined as a hook of the `epoch` module's `BeginBlocker`.

However, this function has a large computational footprint. The `UpdateCValues` function in `x/liquidstakeibc/keeper/keeper.go:257` iterates through all the registered `HostChain` elements, and for each of them, iterates all the related `LSMDeposit`, `Validators`, `Deposit`, `ValidatorUnbonding` and then executes other hooks defined in the `PostCValueUpdate` function.

Additionally, the cardinality of some of those slices depends on user interaction, opening the possibility for attackers to influence their length. For instance, the `LSMDeposit` slice can be extended by users by depositing LSM tokens, allowing attackers to maliciously influence the `BeginBlocker` execution time.

Consequently, the `BeginBlocker` execution will take significant time, potentially slowing block production and halting the chain.

Recommendation

We recommend not concentrating all the operations in a single `BeginBlocker` but executing the iterations in multiple batches.

Status: Acknowledged

The client states that they limited the amount of LSM deposits that can be processed at any stage of the flow, which is the only user-dependent way of increasing the resources consumed while processing every epoch.

9. The `DoRedeemLSMTokens` function consumes excessive computational resources in the `BeginBlocker`

Severity: Major

In `x/liquidstakeibc/keeper/abci.go:305`, the `DoRedeemLSMTokens` function is called in the `BeginBlocker` and iterates through all deposits in the

`LSMDeposit_DEPOSIT_RECEIVED` state retrieved by the `GetRedeemableLSMDeposits` function.

Since the cardinality of `LSMDeposits` depends on user interaction, attackers can influence their length by depositing many LSM tokens.

Consequently, the `BeginBlocker` execution will take significant time, potentially slowing block production and halting the chain.

Recommendation

We recommend limiting the iteration scope of the `DoRedeemLSMTokens` function to a reasonable number of `LSMDeposit` instances, ensuring that it does not process an unmanageable quantity of deposits in a single block.

Status: Resolved

10. Validators are incorrectly set to non-delegable for default bond factor values

Severity: Major

In `x/liquidstakeibc/keeper/host_chain.go:202`, the `ProcessHostChainValidatorUpdates` function computes the available amount of shares that can be bonded (indicated as the `bondRoom` variable) by multiplying the validator's self-bonded shares with `hc.Params.LsmBondFactor`. The bond factor can be configured to `-1`, which signifies that the bond factor validation is disabled and should be skipped, as seen in `x/liquidstakeibc/keeper/host_chain.go:128-129`.

The implementation is incorrect though, as the `bondRoom` variable will become a negative value when the bond factor is disabled, causing the validator to incorrectly be determined as non-delegable in line `212`.

Recommendation

We recommend skipping bond factor validation when `hc.Params.LsmBondFactor` is `-1`.

Status: Resolved

11. Incorrect comparison between shares and tokens

Severity: Major

In several instances of the codebase, comparisons between shares and tokens are performed. This is incorrect because the number of shares is generally lower than the tokens due to the exchange rate, causing the comparisons to be incorrect and ineffective.

Firstly, in `x/liquidstakeibc/keeper/host_chain.go:206`, the `ProcessHostChainValidatorUpdates` function compares `msgDelegate.Amount.Amount` (denominated in tokens) with `capRoom` and `bondRoom` (denominated in shares) to determine whether the validator has sufficient capacity for delegation. This is incorrect because the amount to delegate should be divided by the exchange rate before performing the comparison to ensure both denominations are equal.

Secondly, in `x/liquidstakeibc/keeper/msg_server.go:464`, the `LiquidStakeLSM` function ensures `delegation.Amount` (denominated in shares) is not less than `hc.MinimumDeposit` (denominated in tokens) when performing the minimum deposit validation. This is incorrect because the tokenized validator shares are denominated in shares, and multiplying with the exchange rate is required to compare the values correctly.

Lastly, in line `562`, the `LiquidStakeLSM` function emits the `input_amount` as `hc.HostDenom` coin denom (denominated in tokens) with `delegation.Amount` coin amount (denominated in shares). This is incorrect because the caller provided delegation amounts as shares denom, not tokens. Hence, the coin value should be `deposit.Amount` that denominates in tokens denom, as seen in line `478`.

Recommendation

We recommend comparing the values in equal denominations mentioned above.

Status: Resolved

12. The bond factor is not set to the default value when registering host chains

Severity: Minor

In `x/liquidstakeibc/keeper/msg_server.go:53-58`, the `RegisterHostChain` function does not set the value for `LsmBondFactor` in the `HostChainLSParams` struct to `-1`. This is problematic because the `LsmBondFactor` value will be set to zero, which is incorrect since the default value should be `-1`, as seen in `x/liquidstakeibc/keeper/host_chain.go:126-127`.

Consequently, newly registered host chains will be automatically configured with a bond factor of zero, causing all validators to be determined as non-delegable in lines `131` to `138`.

We classify this issue as minor severity because the admin or authority can update the `LsmBondFactor` value to recover from it.

Recommendation

We recommend setting `LsmBondFactor` to `-1` when registering new host chains.

Status: Resolved

13. A significant decrease in `CValue` could disable the host chain and require a global parameters update

Severity: Minor

In `x/liquidstakeibc/keeper/keeper.go:257`, the `UpdateCValues` function updates the `CValue` by computing the liquid-staked amount and the minted liquid-staked tokens. If the `CValue` falls below the `LowerCValueLimit`, which is a global parameter, the chain will be disabled, and user funds will be locked.

Suppose the authority or the admin decides to reduce the limit of `LowerCValueLimit` to re-enable the locked chain. In that case, other registered chains will also use the new `LowerCValueLimit` because it is a global parameter.

Recommendation

We recommend implementing local `CValue` parameters for each chain to have more granular control over each host chain.

Status: Resolved

14. `LowerCValueLimit` can be misconfigured as a negative value

Severity: Minor

In `x/liquidstakeibc/types/params.go:49-51`, the `Validate` function does not validate that the `LowerCValueLimit` is not a negative value. Since the C value redemption rate will not be a negative value, misconfiguring the `LowerCValueLimit` will always cause the `CValueWithinLimits` function in `x/liquidstakeibc/keeper/keeper.go:353` to evaluate `hc.CValue.GT(k.GetParams(ctx).LowerCValueLimit)` as `true`.

We classify this issue as minor because the `LowerCValueLimit` value can only be configured by the authority or the admin, which are both privileged roles.

Please see the `TestParams_NegativeCValue` test case in the appendix to reproduce the issue.

Recommendation

We recommend validating the `LowerCValueLimit` to ensure it is not a negative value.

Status: Resolved

15. Incomplete `HostChain` validation

Severity: Minor

In `x/liquidstakeibc/types/liquidstakeibc.go:78-85`, the `Validate` function validates the provided `HostChain` struct.

However, while most of its fields are validated, `ChainId`, `ConnectionId`, `HostDenom`, `ChannelID`, and `PortId` are not verified as non-empty strings. Similarly, `DelegationAccount` and `RewardsAccount` lack validation as valid `ICAAccount` instances and `RewardParams` remain unvalidated in cases where the destination address is defined.

Recommendation

We recommend validating `ChainId`, `ConnectionId`, `HostDenom`, `ChannelID`, and `PortId` as non-empty strings, `DelegationAccount` and `RewardsAccount` as valid `ICAAccount` instances, and `RewardParams` in cases where the destination address is defined.

Status: Resolved

16. The `RegisterHostChain` function allows to overwrite existing `HostChain`

Severity: Minor

In `x/liquidstakeibc/keeper/msg_server.go:34`, the `RegisterHostChain` function permits the `AdminAddress` and `Authority` to register a new `HostChain`.

However, if the `HostChain` is already registered, it does not return an error. This could result in the execution overwriting an existing `HostChain` and resetting all its fields, including the ICA accounts balances.

This situation poses a potential risk of losing the permission to manage deposits and undelegations on the other chain and stored data.

We classify this issue as minor severity since it can only be caused by the admin or authority, which is a privileged address.

Recommendation

We recommend returning an error if the chain ID for the host chain is already registered.

Status: Resolved

17. Fees lack validation

Severity: Minor

In `x/liquidstakeibc/types/liquidstakeibc.pb.go:460`, the `HostChainLSParams` struct features a deposit, restake, unstake, and redemption fee. The fee values are validated in `x/liquidstakeibc/types/msgs.go:122-152` to be positive and less than 100%.

Recommendation

We recommend enforcing reasonable upper bounds on the values of fee parameters.

Status: Resolved

18. Matured undelegations are delayed by one block

Severity: Minor:

In `x/liquidstakeibc/keeper/abci.go:231`, the `DoProcessMaturedUndelegations` function processes completed undelegations as indicated by `ctx.BlockTime().After(u.MatureTime)`. This comparison excludes undelegations that have matured in the current block, which should be processed together.

Recommendation

We recommend changing the logic in lines `237` and `273` to include undelegations that are complete in the current block time.

Status: Acknowledged

19. Unhandled errors in the codebase

Severity: Minor

In the following instances of the codebase, errors are ignored and not handled:

- `x/liquidstakeibc/keeper/abci.go:109, 352`
- `x/liquidstakeibc/keeper/hooks.go:649, 791`
- `x/liquidstakeibc/keeper/keeper.go:327`
- `x/liquidstakeibc/types/msgs.go:52-55`

If an error occurs in the above instances, it will fail silently without reverting the transaction.

For instance, if the `ica_messages` attribute emits an empty attribute, the gauge metric emits a zero redemption rate, and the fee validation in `NewMsgRegisterHostChain` will fail silently.

Recommendation

We recommend handling the error and reverting the transaction if needed.

Status: Resolved

20. Host denom can be registered with multiple host chains

Severity: Minor

In `x/liquidstakeibc/keeper/msg_server.go:34`, the `RegisterHostChain` function allows the `AdminAddress` and the `Authority` to register a new `HostChain`.

However, when a new host chain is registered, it is not verified that the provided `HostDenom` has not already been used by another `HostChain` in line `66`.

Since this parameter is used to compute the `MintDenom` in `x/liquidstakeibc/types/liquidstakeibc.go:21`, it could cause a situation where several chains use the same `MintDenom` to create the same liquid staking coins, causing the redemption rate to be computed incorrectly, as seen in `x/liquidstakeibc/keeper/keeper.go:263` and `287`.

We classify this issue as minor severity since it can only be caused by the admin or authority, which is a privileged role.

Recommendation

We recommend verifying that the `HostDenom` provided in the `RegisterHostChain` function is not used by other `HostChain`.

Status: Resolved

21. The total validator weight is not validated to be 100%

Severity: Minor

In `x/liquidstakeibc/types/liquidstakeibc.go:78-83`, the `Validate` function does not ensure the total validator weight (indicated as the `validator.Weight` variable) sums up to 100%. This is problematic because the weight is used to compute how many tokens should be bonded or unbonded to a specific validator, as seen in `x/liquidstakeibc/keeper/delegation_strategy.go:74`.

Consequently, misconfiguring the total validator weight to exceed 100% would cause the validators with excess weights to be left out in `x/liquidstakeibc/keeper/delegation_strategy.go:108-119`, while

misconfiguring the total validator weight to below 100% would cause unused tokens to remain undelegated in the host chain's delegation account.

This issue also happens when performing changes to the validators, such as adding a new validator, removing an existing validator, and updating a validator's weight, as seen in `x/liquidstakeibc/keeper/msg_server.go:142-172`, and `181-190`.

We classify this issue as minor severity since it can only be caused by the admin or authority, which is a privileged role.

Recommendation

We recommend validating the total weight of all validators to be 100% when validating the genesis state and performing validator changes.

Status: Acknowledged

The client states that the process that updates the weights does the calculation, and all messages are sent in one transaction.

22. Genesis validation incorrectly fails for sent deposits

Severity: Informational

In `x/liquidstakeibc/types/liquidstakeibc.go:51-59`, the `Validate` function validates the host chain's deposit state to be either `Deposit_DEPOSIT_PENDING`, `Deposit_DEPOSIT_RECEIVED`, or `Deposit_DEPOSIT_DELEGATING`. However, the `Deposit_DEPOSIT_SENT` state is not included despite being one of the valid deposit states, as seen in `x/liquidstakeibc/types/liquidstakeibc.pb.go:67`.

Consequently, deposits with their state set to `Deposit_DEPOSIT_SENT` in `x/liquidstakeibc/keeper/hooks.go:563` will be incorrectly determined as invalid genesis state, causing genesis validation to fail.

Please see the `TestDeposit_Validate_Deposit_Sent` test case in the appendix to reproduce the issue.

Recommendation

We recommend allowing `Deposit_DEPOSIT_SENT` to be one of the allowed deposit states.

Status: Resolved

23. Low `AutoCompoundFactor` results in unutilized funds

Severity: Informational

In `x/liquidstakeibc/keeper/icq.go:145`, the `RewardsAccountBalanceCallback` function sends funds to the rewards account to auto-compound delegation rewards. The amount to be transferred and auto-compounded is determined by the `AutoCompoundFactor`, which is set upon chain registration and can be updated by `Authority` or `Admin`. If this factor is set too low, this would result in unutilized funds in the rewards account.

Recommendation

We recommend ensuring that the `AutoCompoundFactor` is aligned with the expected APY.

Status: Acknowledged

24. Only the last broken invariants are reported

Severity: Informational

In `x/liquidstakeibc/keeper/invariants.go:27-34`, the `CValueLimits` function checks the invariants for the module and ensures that for each `HostChain`, the `CValue` is within the specified limits.

If the invariant checks fail and multiple chains with `CValue` are outside the limits, the function should return an error message for all affected chains to the string `str`. This is not the case, as the function only reports the last broken invariant in the returned `Invariat` struct.

Recommendation

We recommend reporting all the broken invariants.

Status: Resolved

25. Unnecessary module owner updates

Severity: Informational

In `x/liquidstakeibc/keeper/ibc.go:72` and `76`, the `OnChanOpenAck` function checks the port owner address and updates the `hc.DelegationAccount.Owner` and `hc.RewardsAccount.Owner` to the same address in lines `74` and `78`. This is unnecessary because the updated value will be the same.

Recommendation

We recommend removing `x/liquidstakeibc/keeper/ibc.go:74` and `78` to increase code readability.

Status: Resolved

26. Unused error in the codebase

Severity: Informational

The `ErrFailedICQRequest` error in `x/liquidstakeibc/types/errors.go:14` is unused.

Recommendation

We recommend removing unused errors.

Status: Resolved

27. Missing simulation features in the `liquidstakeibc` module

Severity: Informational

Cosmos SDK provides a comprehensive simulation framework that enables thorough fuzz-testing of every message defined by a module. This blockchain simulator evaluates the behavior of blockchain applications, simulating real-life circumstances by generating and sending randomized messages.

Recommendation

We recommend implementing simulation features for the `liquidstakeibc` module, as indicated in `x/liquidstakeibc/module.go:143`.

Status: Acknowledged

28. Usage of magic numbers decreases maintainability

Severity: Informational

In `x/liquidstakeibc/keeper/keeper.go:358`, hard-coded number literals without context or a description are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintainability.

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

29. Contracts should implement a two-step ownership transfer

Severity: Informational

In `x/liquidstakeibc/keeper/msg_server.go:874`, the `UpdateParams` function allows the `AdminAddress` param to be updated as 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 the 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 param 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.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

The client states that the second owner is the governance module address, hence the admin address can always be re-stated via governance.

30. Incorrect `existing_delegation` attribute emitted

Severity: Informational

In `x/liquidstakeibc/keeper/icq.go:117`, the `DelegationCallback` function emits the `existing_delegation` attributes as `validator.DelegatedAmount`. This is incorrect because the existing delegation represents the previously delegated amount.

Consequently, the emitted value will be incorrect since the `validator.DelegatedAmount` has been updated to the latest delegated amount in line `109`.

Recommendation

We recommend emitting the `existing_delegation` attribute as the previously delegated amount.

Status: Resolved

31. Relayers are not incentivized to relay ABCI-initiated IBC transactions

Severity: Informational

The app chain implements the ICS-29 specification in order to charge fees for IBC transactions. This is important because relayers are incentivized to relay packets from the controller chain to the host chain (and vice versa), and implementing fees would increase the cost for attackers to spam transactions that can drain relayers' funds.

However, the specification cannot be applied to IBC transactions initiated in the ABCI module, as seen in `x/liquidstakeibc/keeper/abci.go:18`. Since the relayers are not compensated, they will operate at a loss, potentially causing them to not relay transactions due to missing incentives.

We classify this issue as informational because relayers can be incentivized through other means, such as grants and compensation through a governance proposal.

Recommendation

We recommend implementing a fee granter account to incentivize relayers to relay ABCI-initiated transactions.

Status: Acknowledged

Appendix

1. Command to compute test coverage excluding proto-generated files

To compute the test coverage accurately without including code generated by protoc-gen-gogo and protoc-gen-grpc-gateway, file extensions that end with `.pb.go` or `.pb.gw.go` need to be excluded.

The following command is executed in the `x/liquidstakeibc` directory to compute the test coverage:

```
go test -coverprofile=coverage.out ./... && grep -vE ".pb.go|.pb.gw.go"
coverage.out | go tool cover -func=/dev/stdin | grep total | awk '{print $3}'
```

2. Test case for "Genesis validation incorrectly fails for sent deposits"

Please run the test case in `x/liquidstakeibc/types/liquidstakeibc_test.go`.

```
func TestDeposit_Validate_Deposit_Sent(t *testing.T) {
    type fields struct {
        ChainId     string
        Amount      sdk.Coin
        Epoch       int64
        State       types.Deposit_DepositState
        IbcSequenceId string
    }
    validCoin := sdk.NewInt64Coin("ibc/uatom", 1000)
    invalidCoin := validCoin
    invalidCoin.Amount = sdk.NewInt(-1)
    tests := []struct {
        name  string
        fields  fields
        wantErr bool
    }{
        {
                name: "valid",
                fields: fields{
                        ChainId:    "chain-1",
                        Amount:     validCoin,
                        Epoch:      0,
                        State:      1, // Deposit_DEPOSIT_SENT
                        IbcSequenceId: "",
                },
                wantErr: false,
        },
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
                deposit := &types.Deposit{
                        ChainId:    tt.fields.ChainId,
                        Amount:     tt.fields.Amount,
                        Epoch:      tt.fields.Epoch,
                        State:      tt.fields.State,
                        IbcSequenceId: tt.fields.IbcSequenceId,
                }
                if err := deposit.Validate(); (err != nil) != tt.wantErr {
                        t.Errorf("Validate() error = %v, wantErr %v", err,
tt.wantErr)
                }
        })
    }
}
```

3. Test case for "`LowerCValueLimit` can be misconfigured as a negative value"

Please run the test case in `x/liquidstakeibc/types/params_test.go`.

```
func TestParams_NegativeCValue(t *testing.T) {
    type fields struct {
        AdminAddress       sdk.AccAddress
        FeeAddress         sdk.AccAddress
        UpperCValueLimit sdk.Dec
        LowerCValueLimit sdk.Dec
    }
    tests := []struct {
        name  string
        fields  fields
        wantErr bool
    }{
        {
               name: "negative C value",
               fields: fields{
                      AdminAddress:      types.DefaultAdminAddress,
                      FeeAddress:        types.DefaultFeeAddress,
                      UpperCValueLimit: sdk.OneDec(),
                      LowerCValueLimit: sdk.NewDec(-1),
               },
               wantErr: false,
        },
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
               p := &types.Params{
                      AdminAddress:      tt.fields.AdminAddress.String(),
                      FeeAddress:        tt.fields.FeeAddress.String(),
                      UpperCValueLimit: tt.fields.UpperCValueLimit,
                      LowerCValueLimit: tt.fields.LowerCValueLimit,
               }
               if err := p.Validate(); (err != nil) != tt.wantErr {
                      t.Errorf("Validate() error = %v, wantErr %v", err,
tt.wantErr)
               }
        })
    }
}
```