Audit Report

pSTAKE Native

v1.0

February 17, 2023

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 Technologies Pte Ltd to perform a security audit of pSTAKE Native.

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/persistenceOne/pstake-native

Commit hash: `8113b4a8260dc2470b470a0281f7e6a77fca1c3b`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted codebase features stkATOM, a non-custodial ATOM liquid staking solution on the Persistence Core-1 chain, bringing stkATOM directly into the Cosmos ecosystem.

The audit scope is restricted to the `x/lscosmos` Cosmos SDK module.

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	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	No technical documentation was provided.
Test coverage	Low-Medium	33.2% reported code coverage

Summary of Findings

No	Description	Severity	Status
1	Inconsistent state mutation when handling acknowledgments	Critical	Resolved
2	Unsuccessful `MsgTransfer` message could drain the relayer's funds	Major	Resolved
3	gRPC-Gateway routes are not registered	Major	Resolved
4	Module state can only be disabled through a coordinated chain upgrade	Major	Resolved
5	Host chain's fee account can reset the database when the module is disabled	Minor	Resolved
6	Incorrect validation on undelegation completion time delays a block from being processed	Minor	Resolved
7	`HostChainParams` cannot be updated	Minor	Resolved
8	`MintDenom` is not derived from BaseDenom	Minor	Resolved
9	Negative validator weights cause distribution failure	Minor	Resolved
10	`MempoolFeeDecorator` allows attackers to slow down or halt the chain	Informational	Resolved
11	Incorrect log messages and comments negatively affect maintainability	Informational	Resolved
12	Code inefficiencies	Informational	Partially Resolved
13	Duplicate checks can be removed	Informational	Resolved
14	Outstanding TODO comments in codebase	Informational	Acknowledged
15	`JumpStart` message does not validate the minimum deposit amount as positive	Informational	Resolved

Detailed Findings

1. Inconsistent state mutation when handling acknowledgments

Severity: Critical

In `x/lscosmos/keeper/handshake.go:298-320`, the delegator's reward withdrawal and undelegation message are only processed if they are at the first index of the `txMsgData.Data` slice. Only the first message is processed if multiple messages are provided when handling successful acknowledgments.

Besides that, the `handleAckMsgData` function in line `291` performs a mutation for undelegation messages. In line `421`, `SubtractHostAccountDelegation` is called to deduct the host account's delegation based on the undelegation message amount.

Suppose that a `MsgWithdrawDelegatorReward` and `MsgUndelegate` message is sent. The first loop would process the delegator's reward withdrawal. On the second loop, `handleAckMsgData` would be executed to subtract the host account's delegation amount. However, the undelegation message is not processed since it's on the second index.

As a result, in case of receiving multiple transaction messages in the `OnAcknowledgementPacket` callback function, only the first message of `MsgUndelegate` is executed. This causes a state inconsistency issue because matured undelegations are not recorded properly, causing users to be unable to retrieve their unbonded `ATOMs`.

This issue is also present in line `460`, where `stakingtypes.MsgUndelegate` will only be handled if it is the first index of the message.

Recommendation

We recommend handling all messages regardless of the current index value.

Status: Resolved

2. Unsuccessful `MsgTransfer` message could drain the relayer's funds

Severity: Major

In `x/lscosmos/keeper/handshake.go:488-500`, the function responsible for handling `Timeout` and `Acknowledgement_Error` responses (see lines `196` and `258`) originated by `MsgTransfer` messages try to resend them through `GenerateAndExecuteICATx`.

In a scenario where a transaction always has unsuccessful responses, the execution will end up performing a loop between the controller and controlled chains with the result of draining

the relayer's funds. For example, this could happen if the controlled chain is halted or is performing an update.

Additionally, an attacker may trigger multiple such transfers to congest and slow down the chain. As a result, users will find their transactions taking more time to process.

Recommendation

We recommend not resending the message through `GenerateAndExecuteICATx` and delegating the responsibility for the `BeginBlocker` to retry in the next block.

Status: Resolved

3. gRPC-Gateway routes are not registered

Severity: Major

In `x/lscosmos/module.go:75`, the HTTP handlers are not registered for service `Query` to "mux". This causes all gRPC queries to fail.

Recommendation

We recommend registering the gRPC-Gateway routes with the `RegisterQueryHandlerClient` function.

Status: Resolved

4. Module state can only be disabled through a coordinated chain upgrade

Severity: Major

The messages defined in `x/lscosmos/handler.go:21-35` can only be executed when the module state is enabled. If the module is disabled, the epoch logic and main messages, such as liquid staking, are prohibited. However, the current implementation only allows the module to be disabled from `InitGenesis`, as seen in `x/lscosmos/genesis.go:16`. This implies that all validators would need to coordinate to halt and upgrade the chain in order to set the module to disabled.

Coordinating an upgrade with validators is manual and time-intensive progress, which prevents swift reactions to security incidents. For example, suppose the ATOM chain halts, and the module needs to be temporarily disabled.

Recommendation

We recommend implementing the following suggestions:

- Document conditions under which the module should be disabled.
- Add logic to conditionally disable the module.
- Allow governance to disable the module, potentially with a custom quorum.
- Optionally allow a number of whitelisted admins to temporarily disable the module, ideally with a timelock rather than a boolean value.

Status: Resolved

5. Host chain's fee account can reset the database when the module is disabled

Severity: Minor

In `x/lscosmos/keeper/msg_server.go:469-474`, the `JumpStart` message can be executed by the host chain's fee account as long the module is disabled. That message will remove the existing delegation state. As a result, active delegations and undelegations in the storage state would be removed, causing a loss of funds for delegators.

We classify this as a minor issue since only the admin can cause it.

Recommendation

We recommend ensuring the `JumpStart` message can only be called once. As an alternative, consider using a governance proposal instead of a `JumpStart` message.

Status: Resolved

6. Incorrect validation on undelegation completion time delays undelegations by one block

Severity: Minor

In `x/lscosmos/keeper/delegation_state.go:197`, the `ctx.BlockTime().After(undelegation.CompletionTime)` condition does not include the case when the current block time equals the completion time. This implied that mature delegations would be delayed by a block.

Recommendation

We recommend replacing the `ctx.BlockTime().After(undelegation.CompletionTime)` with

`!ctx.BlockTime().Before(undelegation.CompletionTime)` to prevent the potential delay.

Status: Resolved

7. `HostChainParams` cannot be updated

Severity: Minor

In `x/lscosmos/keeper/governance_proposal.go:18-21`, when handling a `RegisterHostChainProposal`, the guard ensures that `HostChainParams` are not already configured.

This implies that there is no way for governance to update existing `HostChainParams`. Since the administrator likely configures the host chain parameters with the `JumpStart` message, this proposal will always fail.

Recommendation

We recommend removing the guard to allow governance to update `HostChainParams` if needed. Alternatively, we recommend removing the proposal handler if it is not used.

Status: Resolved

8. `MintDenom` is not derived from `BaseDenom`

Severity: Minor

In `x/lscosmos/keeper/msg_server.go:485-488`, during the handling of the `JumpStart` message, the defined guard only checks that `mint` and `base` denoms are not equal.

By design, `MintDenom` should be derived from `BaseDenom` with the `stk` prefix, so the mint denom should be derived from the base denom in the code.

Recommendation

We recommend enforcing that `MintDenom` is constructed from `BaseDenom` and the `stk` prefix.

Status: Resolved

9. Negative validator weights cause distribution failure

Severity: Minor

In `x/lscosmos/types/lscosmos.go:34-L45`, the `Valid` functionality attempts to verify the validator's weight sums to one decimal. As decimals values can be negative, it is possible to configure validators with a negative weight value. On the other hand, a validator can possibly have a weight higher than one decimal as long as the total of all validators equals one.

This can cause the `distributeCoinsAmongstValSet` function to fail in `x/lscosmos/keeper/delegation_strategy.go:175` since coins cannot contain negative values.

Recommendation

We recommend ensuring the validator's weight is non-negative.

Status: Resolved

10. `MempoolFeeDecorator` allows attackers to slow down or halt the chain

Severity: Informational

The `MempoolFeeDecorator AnteHandler` defined in `ante/fee.go:20-75` allows whitelisted message types to have a gas price less than `MinGasPrices`. The handler performs an `O(n)` iteration through all messages included in the transaction to verify that each type is whitelisted. This unbounded iteration allows an attacker to slow down block production. If the `BroadcastTxCommit` timeout is hit, the node may not be able to process further `ABCI` messages such that it has to pause and contact peers to get the latest correct blocks.

An attacker could craft a large slice of messages where all are of a whitelisted type except the last one. This would iterate through all of them, return false, and cause the transaction run out of gas. With this approach, the attacker can overload nodes by performing a computationally costly iteration without spending gas.

Another possible attack could be crafting a large slice of whitelisted messages to spam the network by leveraging the fact that the following guard

$$gas \leq \#msgs \cdot maxBypassMinFeeMsgGasUsage$$

has not a cumulative hard cap but depends on the number of messages.

We classify this issue as informational since the code that causes this issue is out of scope of this audit.

Recommendation

We recommend enabling only a subset of whitelisted addresses to execute whitelisted messages with a lower gas price.

Status: Resolved

11. Incorrect log messages and comments negatively affect maintainability

Severity: Informational

There are some places that use incorrect log messages and comments. This reduces maintainability and makes debugging more difficult.

In `x/lscosmos/keeper/abci.go:27`, the function processes matured undelegations, while the log message misleadingly states `k.Logger(ctx).Error("Unable to Delegate tokens with ", "err: ", err)`.

In `x/lscosmos/keeper/abci.go:64`, the function name is `ProcessMaturedUndelegation`, while the comments refer to all the matured delegations.

In `x/lscosmos/keeper/governance_proposal.go:53`, the function registers an interchain account in the host chain using the `RegisterInterchainAccount` function, while the comment is incorrect.

Recommendation

We recommend fixing the incorrect log messages and comments.

Status: Resolved

12. Code inefficiencies

Severity: Informational

There are several parts of the codebase that can be optimized to reduce gas consumption and computational resources:

a) In `x/lscosmos/keeper/abci.go:37`, the `GetHostAccounts` function is called in every `BeginBlocker`, but it is ineffective when either `delegatableAmount` is 0 or the length of `allowListedValidators` is 0. It would lead to lower gas consumption if the function was called after the validation.
b) In `x/lscosmos/keeper/abci.go:68-69`, the `GetDelegationState` and `GetHostAccounts` functions are called in every `BeginBlocker`, but they are

ineffective when the length of `maturedUndelegations` is 0. It would lead to lower gas consumption if the function was called after the validation.

c) In `x/lscosmos/keeper/delegation_strategy.go:21`, the `GetDelegationState` function is previously called in the `DoDelegate` function. It would reduce gas consumption if the delegation state was passed rather than read again from the store.

d) In `x/lscosmos/keeper/delegation_strategy.go:246`, the `GetHostChainParams` function is called from the store and used for `BaseDenom`. It would reduce gas consumption if the base denom was passed from the calling function.

e) In `x/lscosmos/keeper/delegation_strategy.go:264-266`, it would be more efficient if the for loop was combined with lines `251` to `253`.

f) In `x/lscosmos/keeper/delegation_strategy.go:270-272`, it would be more efficient if the for loop was combined with lines `258` to `260`.

g) In `x/lscosmos/keeper/msg_server.go:405-442`, instead of calling `SendCoinsFromModuleToAccount` at each iteration, it would be more efficient if an accumulator was used.

h) In `x/lscosmos/keeper/hooks.go:167`, it is safer to use `!remainingDelegationBalance.IsAllPositive` rather than `remainingDelegationBalance.Empty` because that would check the length of coins and perform coin amount validation.

i) In `x/lscosmos/keeper/handshake.go:193-236`, two switch statements perform different operations based on the same case statement. As the operations can be combined inside a single case statement, consider modifying the implementation only to use a single switch statement. For instance, lines `223` to `228` can be included inside lines `201` to `205` to avoid the extra switch statement.

Recommendation

We recommend applying the above optimizations to reduce gas and computational resource consumption.

Status: Partially Resolved

13. Duplicate checks can be removed

Severity: Informational

The `msg.Amount.IsValid` function is called in the following locations:

- `x/lscosmos/keeper/msg_server.go:35`
- `x/lscosmos/keeper/msg_server.go:148`
- `x/lscosmos/keeper/msg_server.go:213`
- `x/lscosmos/keeper/msg_server.go:289`
- `x/lscosmos/keeper/msg_server.go:388`
- `x/lscosmos/keeper/msg_server.go:460`

These are duplicated invocations as they are being checked in `types/msgs.go` file. Moreover, `ctx.IsZero` is not needed as it checks the `MultiStore` interface is `nil`, which cannot be `nil` as long as the `StoreKey` is registered in the `NewKVStoreKeys` function in the `app.go` file.

Recommendation

We recommend removing `ctx.IsZero` and `msg.Amount.IsValid` function calls as they are duplicated invocations.

Status: Resolved

14. Outstanding TODO comments in codebase

Severity: Informational

In several instances of the codebase, some TODO comments exist:

- `x/lscosmos/keeper/governance_proposal.go:40`
- `x/lscosmos/keeper/handshake.go:96`
- `x/lscosmos/keeper/handshake.go:180`
- `x/lscosmos/keeper/msg_server.go:486`
- `x/lscosmos/keeper/msg_server.go:505`
- `x/lscosmos/types/governance_proposal.go:96`

Recommendation

We recommend resolving or removing TODO comments.

Status: Acknowledged

15. `JumpStart` message does not validate the minimum deposit amount as positive

Severity: Informational

The `JumpStart` message in `x/lscosmos/keeper/msg_server.go:457` does not verify the provided minimum deposit amount is a positive number. A negative minimum deposit amount configured allows users to provide a very minimal amount to liquid stake. Consequently, this might cause profit loss due to higher operating expenses than the accrued fees.

We consider this an informational issue since it can only be caused by the host chain's fee account address.

Recommendation

We recommend validating the minimum deposit amount to be a positive value.

Status: Resolved