Zellic

Prepared for
Eric Warehime
The Hub team at Informal Systems

Prepared by
Hojung Han
Jaeeu Kim
Zellic

March 26, 2025

Cosmos SDK Liquid Staking Module

Cosmos Application Security Assessment

Contents

DRAFT

DRAFT

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

DRAFT

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for the Hub team at Informal Systems from March 5th to March 12th, 2025. During this engagement, Zellic reviewed Cosmos SDK Liquid Staking Module's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are users able to get more tokens or shares than they are entitled?
- Can deposits or withdrawals be manipulated to unfairly reward or penalize users?
- Is it possible for any role to lock user funds?
- Are there any potential issues that could arise from migrating the liquid-staking–related code from Cosmos SDK to gaia?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

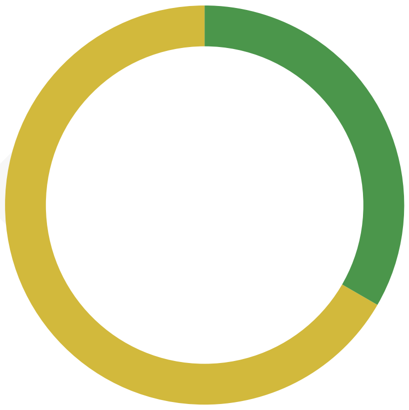
1.4. Results

During our assessment on the scoped Cosmos SDK Liquid Staking Module modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

DRAFT

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	2
🟩 Low	1
⬜ Informational	0



DRAFT

2. Introduction

2.1. About Cosmos SDK Liquid Staking Module

The Hub team at Informal Systems contributed the following description of Cosmos SDK Liquid Staking Module:

> It is an extension that enables liquid staking on the Cosmos Hub. Simply put, LSM allows the creation of semi-fungible liquid staking primitives, known as LSM shares. LSM shares are derivatives of delegated shares, linked to a delegator-validator pair, and represent the underlying delegated stake. By issuing LSM shares, the staked ATOM remains slashable while becoming 'liquid'. LSM shares function as tokens that can be utilized in various DeFi protocols and transferred between users or across chains via IBC. Since they are tied to specific delegator-validator pairs, LSM shares are non-fungible. Additionally, because they are issued directly by the Hub, they do not rely on the security of any entity outside of the Cosmos Hub itself.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the modules.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no

DRAFT

hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

DRAFT

2.3. Scope

The engagement involved a review of the following targets:

Cosmos SDK Liquid Staking Module Modules

Type	Cosmos
Platform	Cosmos

Target	Cosmos SDK
Repository	https://github.com/cosmos/cosmos-sdk ↗
Version	`9f2e1dac5d356e12cf43def67c075c8c965e1e6f`

Programs	`staking/*`
	`distribution/*`
	`slashing/*`

Target	gaia
Repository	https://github.com/cosmos/gaia ↗
Version	`f46b6b058caf9b9b2d8780e59e5f7ad30996f5b3`
Programs	`x/liquid/*`

DRAFT

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of one person-week. The assessment was conducted by two consultants over the course of 1.2 calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Hojung Han
Engineer
hojung@zellic.io ↗

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

DRAFT

2.5. Project Timeline

March 5, 2025	Kick-off call
March 5, 2025	Start of primary review period
March 12, 2025	End of primary review period

3. Detailed Findings

3.1. Missing enforcement of `MinSelfDelegation` allows zero self-delegation for validators

Target	x/staking		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

In the original staking module of Cosmos SDK, the `MsgCreateValidator` handler included a validation check to ensure that `MinSelfDelegation` was a positive value. However, when the liquid-staking functionality was integrated into the staking module, this validation was effectively deprecated, and the enforcement of `MinSelfDelegation` was removed.

Later, when the liquid-staking functionality was separated into a dedicated liquid module in the gaia repository, the staking module was restored without liquid staking. As a result, validators who created their validator during the period when `MinSelfDelegation` was not enforced may have unintentionally set `MinSelfDelegation` to `0`, effectively bypassing the original requirement. This means that some validators could currently be operating without any minimum self-delegation, which was previously required as a safeguard to ensure validators had a financial stake in the network.

Impact

Validators without any self-delegation may not have sufficient financial incentives to act honestly, increasing the risk of malicious behavior.

Recommendations

To avoid restricting validators' ability to configure their own settings, avoid enforcing a blanket update to all `MinSelfDelegation` values. The `EditValidator` function enforces an invariant that disallows lowering `MinSelfDelegation` once set, which could limit flexibility. Instead, consider updating the code so that staking message handlers fail when `MinSelfDelegation` is zero—excluding `MsgEditValidator` message handler within the staking module. This approach allows validators to define an appropriate `MinSelfDelegation` value at their discretion in the future.

Remediation

TBD

DRAFT

3.2. Inconsistent handling of liquid-staked token adjustments in `RedeemTokensForShares`

Target	x/liquid		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The `TokenizeShares` function increases liquid-staking–related values for all delegations by calling `SafelyIncreaseTotalLiquidStakedTokens` and `SafelyIncreaseValidatorLiquidShares`. However, in the `RedeemTokensForShares` function, the corresponding decrease functions, `DecreaseTotalLiquidStakedTokens` and `DecreaseValidatorLiquidShares`, are only executed when the delegator is not an interchain account (ICA). This inconsistency means that when ICA delegators redeem their tokenized shares, the total liquid-staked token count and validator liquid-share values are not adjusted accordingly.

Impact

The imbalance caused by this inconsistency may lead to an inaccurate representation of liquid-staked tokens within the system. If liquid-staking caps rely on these values, the limits may be incorrectly enforced, potentially allowing more liquid-staked tokens to exist than intended.

Recommendations

To maintain consistency, `RedeemTokensForShares` should call `DecreaseTotalLiquidStakedTokens` and `DecreaseValidatorLiquidShares` for all delegations.

Remediation

TBD

DRAFT

3.3. Blacklist bypass in reward withdrawal via `TransferTokenizeShareRecord`

Target	x/liquid, x/staking		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

In the `WithdrawSingleShareRecordReward` and `WithdrawAllTokenizeShareRecordReward` functions, rewards are not distributed if the owner is included in the blacklist. However, the `TransferTokenizeShareRecord` function allows users to bypass this restriction. A blacklisted user can transfer ownership of the `TokenizeShare` to an address that is not included in the blacklist and subsequently withdraw rewards from that new address.

Impact

This issue undermines the effectiveness of the blacklist mechanism intended to prevent certain users from receiving rewards. Malicious actors could exploit this loophole to evade restrictions, making the blacklist ineffective in enforcing restrictions on reward distribution.

Recommendations

To ensure the blacklist mechanism functions as intended, consider implementing additional checks in the `TransferTokenizeShareRecord` function to prevent blacklisted addresses from transferring ownership.

Remediation

TBD

DRAFT

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

4.1. Module: staking

The staking module processes various messages for validator and delegation management. Each message type has specific security considerations and validation requirements.

The following section outlines the recent updates to the staking module.

States

Added

- `TotalLiquidStakedTokens`
  - Tracks the total amount of liquid-staked tokens to monitor progress against the `GlobalLiquidStakingCap`.
- `PendingTokenizeShareAuthorizations`
  - Stores a queue of addresses undergoing reactivation/unlocking for tokenized shares.

The following item has been added to `Delegation`.

- `ValidatorBond`
  - Indicates whether the delegation is a validator bond.

The following items have been added to `Validator`.

- `ValidatorBondShares`
  - Represents the number of shares self-bonded by the validator.
- `LiquidShares`
  - Represents the number of shares either tokenized or owned by a liquid-staking provider.

BeginBlock

Added

- `RemoveExpiredTokenizeShareLocks`
  - Finds/releases all expired `TokenizeShareLocks` up to the current block time.

DRAFT

Chain parameters

Added

- `ValidatorBondFactor`
  - Defines the factor for validator bonds.
- `GlobalLiquidStakingCap`
  - Sets the global cap for liquid staking.
- `ValidatorLiquidStakingCap`
  - Sets the liquid-staking cap for individual validators.

Messages

Added

- `MsgUnbondValidator`
  - Sets a validator to be jailed and prepares it for unbonding.
- `MsgTokenizeShares`
  - Tokenizes a specified amount from an existing delegation.
  - Creates a new `TokenizeShareRecord`.
  - Unbonds the specified amount from the existing delegation.
  - Transfers the unbonded tokens to the `ModuleAddress` specified in the `TokenizeShareRecord`.
  - Assigns tokenized shares to the delegator corresponding to the unbonded tokens.
  - Creates a new delegation between the `ModuleAddress` in the `TokenizeShareRecord` and the validator.
- `MsgRedeemTokensForShares`
  - Redeems tokenized shares to reclaim the originally delegated assets.
  - Unbonds the specified amount from the delegation between the `ModuleAddress` and validator, deleting the delegation data if the entire amount is unbonded.
  - Transfers the share tokens to the `NotBondedPool` and burns them.
  - Transfers the corresponding original tokens to the delegator and updates delegation information between the validator and delegator.
- `MsgTransferTokenizeShareRecord`
  - Transfers ownership of tokenized delegation to another user.
  - Removes the relationship between the existing `TokenizeShareRecord` and the current owner.
  - Creates a new relationship between the `TokenizeShareRecord` and the new owner.
- `MsgDisableTokenizeShares`
  - Disables tokenization for a delegator's address.
  - Removes expired locks from `PendingTokenizeShareAuthorizations` if they

DRAFT

exist.
    - Creates a new permanent lock and adds it to `PendingTokenizeShareAuthorizations`.
- `MsgEnableTokenizeShares`
    - Initiates reauthorization for tokenization for a delegator's address.
    - Adds a lock to `PendingTokenizeShareAuthorizations` that will be released after the unbonding period to allow tokenization.
- `MsgValidatorBond`
    - Designates a specific delegation as a validator bond, allowing the validator to receive additional liquid-staking delegations.
    - Marks a specific delegation as the validator's `ValidatorBond`.

Modified

- `MsgCreateValidator`
    - Increased `TotalLiquidStakedTokens` by the amount of tokens delegated, if the delegator is a liquid staker.
    - Increased the validator's `LiquidShares` by the number of shares delegated, if the delegator is a liquid staker.
- `MsgEditValidator`
    - Deleted the `MinSelfDelegation` implementation.
- `MsgDelegate`
    - Increased `TotalLiquidStakedTokens` by the amount of tokens delegated, if the delegator is a liquid staker.
    - Increased the validator's `LiquidShares` by the number of shares delegated, if the delegator is a liquid staker.
    - Increased the validator's `ValidatorBondShares` by the newly delegated shares, if the existing delegation's `ValidatorBond` is true.
- `MsgRedelegate`
    - Added checks that `srcDelegation` exists.
    - Decreased the validator's `ValidatorBondShares` by the redelegated amount, if the source delegation's `ValidatorBond` is true.
    - Increased the destination validator's `LiquidShares` by the number of shares delegated, if the delegator is a liquid staker.
    - Decreased the source validator's `LiquidShares` by the number of shares delegated, if the delegator is a liquid staker.
    - Increased the destination validator's `ValidatorBondShares` by the redelegated amount, if the destination delegation's `ValidatorBond` is true.
- `MsgUndelegate`
    - Added checks that the target `delegation` exists.
    - Decreased the validator's `ValidatorBondShares` by the undelegated amount, if the specified delegation's `ValidatorBond` is true.
    - Decreased `TotalLiquidStakedTokens` by the amount of tokens undelegated, if the delegator is a liquid staker.

DRAFT

    - Decreased the validator's `LiquidShares` by the number of shares undelegated, if the delegator is a liquid staker.
- `MsgCancelUnbondingDelegation`
    - Increased `TotalLiquidStakedTokens` by the amount of tokens for which unbonding is canceled, if the delegator is a liquid staker.
    - Increased the validator's `LiquidShares` by the number of shares for which unbonding is canceled, if the delegator is a liquid staker.
    - Increased the validator's `ValidatorShares` by the number of shares for which unbonding is canceled, if the existing delegation's `ValidatorBond` is true.

Invariants

This section describes new/removed invariants compared to the existing staking module.

Validator-management messages

- `MsgEditValidator`
    - Validation for `msg.MinSelfDelegation` has been removed.
- `MsgUnbondValidator`
    - The `msg.ValidatorAddress` must have been registered through a prior `MsgCreateValidator`.
    - The `msg.ValidatorAddress` must not already be in jail.
- `MsgValidatorBond`
    - The `msg.ValidatorAddress` must have been registered through a prior `MsgCreateValidator`.
    - The `msg.DelegatorAddress` must have delegated self-staked tokens to the validator in advance.
    - The `msg.DelegatorAddress` must not be a liquid staker.
    - The delegation between `msg.ValidatorAddress` and `msg.DelegatorAddress` must not already be set as a validator bond.

Share-tokenization messages

- `MsgTokenizeShares`
    - The `msg.Amount.Amount` must be greater than zero and not nil.
    - The `msg.Amount.Denom` must match the regular expression `[a-zA-Z][a-zA-Z0-9/:._-]{2,127}`.
    - The `msg.Amount.Amount` cannot exceed the number of tokens owned by the delegator for `msg.Amount.Denom`.
    - The tokenize-share lock for `msg.DelegatorAddress` must not be in `TOKENIZE_SHARE_LOCK_STATUS_LOCKED` or `TOKENIZE_SHARE_LOCK_STATUS_LOCK_EXPIRING`.

DRAFT

  - A delegation must exist between the validator specified by `msg.ValidatorAddress` and the delegator specified by `msg.DelegatorAddress`.
  - The delegation's `ValidatorBond` must not be true.
  - The `msg.Amount.Denom` must match the denom specified in the chain parameters.
  - If the delegator specified by `msg.DelegatorAddress` is a vesting account, the tokenized shares must not exceed the amount available for use after the vesting period.
  - The delegator specified by `msg.DelegatorAddress` must not be in the middle of a redelegation process.
  - The shares to be unbonded by the delegator specified by `msg.DelegatorAddress` must not exceed the delegation's total shares.
- `MsgRedeemTokensForShares`
  - The `msg.Amount.Amount` must be greater than zero and not nil.
  - The `msg.Amount.Denom` must match the regular expression `[a-zA-Z][a-zA-Z0-9/:._-]{2,127}`.
  - The `msg.Amount.Amount` cannot exceed the number of tokens owned by the delegator for `msg.Amount.Denom`.
  - A `TokenizeShareRecord` related to `msg.Amount.Denom` must exist.
  - The validator in the `TokenizeShareRecord` must have been registered through a prior `MsgCreateValidator`.
  - A delegation relationship must exist between the `Module Address` in the `TokenizeShareRecord` and the validator.
  - The shares calculated for the `msg.Amount.Amount` based on the validator in the `TokenizeShareRecord` must not be zero.
  - The shares to be unbonded by the delegator specified by `msg.DelegatorAddress` must not exceed the delegation's total shares.
- `MsgTransferTokenizeShareRecord`
  - The `msg.Sender` must be the `record.Owner`.
- `MsgEnableTokenizeShares`
  - The tokenize-share lock related to `msg.DelegatorAddress` must not be in `TOKENIZE_SHARE_LOCK_STATUS_UNLOCKED`.
  - The tokenize-share lock related to `msg.DelegatorAddress` must not be in `TOKENIZE_SHARE_LOCK_STATUS_LOCK_EXPIRING`.
- `MsgDisableTokenizeShares`
  - The tokenize-share lock related to `msg.DelegatorAddress` must not be in `TOKENIZE_SHARE_LOCK_STATUS_LOCKED`.

4.2. Module: distribution

The distribution module is responsible for distributing rewards (block rewards, fees, etc.) between validators and delegators. It handles validator commissions, distributes rewards to delegators, and

DRAFT

manages community-pool funds, implementing the network's reward-distribution mechanism.

The following section outlines the recent updates to the distribution module.

Messages

Added

- `MsgWithdrawTokenizeShareRecordReward`
    - The `TokenizeShareRecord` owners can withdraw rewards from the `TokenizeShareRecord`.
- `MsgWithdrawAllTokenizeShareRecordReward`
    - The `TokenizeShareRecord` owners can withdraw rewards from every `TokenizeShareRecord`.

Invariants

Added

- `MsgWithdrawTokenizeShareRecordReward`
    - The validator of the specified `TokenizeShareRecord` must be a currently registered validator.
    - The owner address is not blocked address.
    - A delegation must exist between the `Module Address` of the specified `TokenizeShareRecord` and the validator.
    - The `Owner` of the specified `TokenizeShareRecord` must match the owner (signer) included in the transaction.
- `MsgWithdrawAllTokenizeShareRecordReward`
    - The validator of the specified `TokenizeShareRecord` must be a currently registered validator.
    - The owner address is not blocked address.
    - A delegation must exist between the `Module Address` of the specified `TokenizeShareRecord` and the validator.

DRAFT

4.3. Module: slashing

The slashing module enables Cosmos SDK–based blockchains to disincentivize any attributable action by a protocol-recognized actor with value at stake by penalizing them (slashing). The following section outlines the recent updates to the slashing module.

Messages

Modified

- `MsgUnjail`
    - Removed checking for the minimum `SelfDelegation` when validator is unjailed.

DRAFT

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Cosmos Chain.

During our assessment on the scoped Cosmos SDK Liquid Staking Module modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.

DRAFT