Audit Report

Liquidity Staking Cosmos SDK Modules

v1.0

July 20, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Interchain Foundation to perform a security audit of the Liquidity Staking Cosmos SDK modules.

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/iqlusioninc/liquidity-staking-module

Commit hash: `faf413d4624af0a6fa8aac2c359d1b1b4f6adbc9`

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 liquidity staking Cosmos SDK modules extend Cosmos SDK's staking, distribution, and slashing modules by functionality that allows tokenization of a delegation, redemption of such tokens, and reward withdrawal. The audited modules have also removed epoch operations within the staking, distribution, and slashing modules.

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.

Summary of Findings

No	Description	Severity	Status
1	Using non-prefixed addresses in storage keys can lead to key collisions, allowing exploits to overwrite data	Critical	Resolved
2	Tokenization of a delegation and selling the shares allows evasion of slashes	Major	Acknowledged
3	Withdrawal of tokenized share record rewards is unbounded, owner can be grieved by an attacker	Major	Partially resolved
4	Several errors are not checked	Major	Resolved
5	Tokenize share record reward query does not include automatically withdrawn rewards	Major	Resolved
6	Share token denoms may be all lower- or all upper-cased, depending on user input	Minor	Resolved
7	Usage of panics for control flow is bad practice	Minor	Acknowledged
8	Addresses are not properly validated, which may cause panics and unexpected behaviour	Minor	Resolved
9	Different tokens for share records of the same validator can have different exchange rates after slashing	Informational	Acknowledged
10	Staking rewards are decoupled from share token holders	Informational	Acknowledged
11	Lack of event emission is bad practice	Informational	Resolved
12	Tokenized share record related transactions and queries are only available via CLI, not REST	Informational	Acknowledged
13	Storing the `Id` in `TokenizeShareRecord` is inefficient	Informational	Acknowledged
14	Storing the `ShareTokenDenom` in `TokenizeShareRecord` is inefficient	Informational	Resolved
15	Specification is outdated	Informational	Acknowledged
16	Unused code negatively impacts maintainability	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Low-Medium**	High-level documentation is present, but lower-level documentation is minimal. Within the spec directories, some documentation has been appended, but the existing documentation has not been adjusted, for example for removed epoch operations and queues.
Test coverage	**Medium**	-

Detailed Findings

1. Using non-prefixed addresses in storage keys can lead to key collisions, allowing exploits to overwrite data

Severity: Critical

The key returned by the `GetTokenizeShareRecordIdByOwnerAndIdKey` function in `x/staking/types/keys.go` contains an address that is not length prefixed. Since addresses could have variable lengths, key collisions are possible if the storage key contains a non-prefixed address followed by another component. This could lead to accidental collisions and could be exploited by an attacker to deliberately overwrite existing data at a target storage location.

Recommendation

We recommend length-prefixing addresses used in storage keys if these addresses are followed by other components in the `tokenize_share_record`.

Status: Resolved

2. Tokenization of a delegation and selling the shares allows evasion of slashes

Severity: Major

When a delegation is tokenized in the `TokenizeShares` function in `x/staking/keeper/msg_server.go`, no `UnbondingDelegationEntry` or `Redelegation` entry is created, but the current delegation is immediately unbonded and a new delegation is created from the module address associated with the `TokenizeShareRecord`. This implies that the previous delegator will not be subject to slashing for infractions that happened before the tokenization. Instead, slashes are now applied to the module address associated with the `TokenizeShareRecord`.

Since Cosmos SDK applies slashes for past infractions to all currently bonded tokens, this is not an issue, as long as the previous delegator owns the created share tokens. The previous delegator might have an incentive though to sell share tokens quickly if they are aware of a pending slashing event. If there is information asymmetry between the share token holder and a potential buyer, a delegator can use this mechanism to evade slashing. An example of this could be a validator that equivocated but has not published the double-signed block.

Rational buyers would therefore not buy any token shares before the `UnbondingTime` passed – but not all buyers can be assumed to be rational. We consider this issue to be major, because it can be used to circumvent one of the fundamental features of proof of stake blockchains.

Recommendation

We recommend protecting buyers of share tokens through one of the following mechanisms:

a) Only allow new delegations to be tokenized, not existing ones.
b) Split tokenization into two stages: First, a new `TokenizeShareRecord` is created. Second, after `UnbondingTime`, token shares can be minted.
c) Prevent transfers of token shares until `UnbondingTime` has passed.

Additionally, we recommend adding a creation timestamp or block height to a `TokenizeShareRecord` such that other protocols can check whether records are older than the `UnbondingTime`.

Lastly, we recommend clearly documenting this potential misalignment of incentives for validators.

Status: Acknowledged

The client states: "The trade-offs that allow transferring accountability for slashing to counter-parties is an intentional design choice without which liquid staking will fail to accomplish its core design goals. We will document this better."

3. Withdrawal of tokenized share record rewards is unbounded, owner can be grieved by an attacker

Severity: Major

The `WithdrawTokenizeShareRecordReward` function in `x/distribution/keeper/keeper.go` contains an unbounded iteration over all `TokenizeShareRecord` of an owner, which may run out of gas if too many entries exist. This may happen unintentionally.

Even worse, since it is possible to transfer ownership of a record to any other address, an attacker can create many low-value `TokenizeShareRecord`s and transfer ownership of them to a target owner. While the target owner can transfer relevant `TokenizeShareRecord`s to another account to resolve this issue, it allows the attacker to grieve the owner.

The same issue exists in the `TokenizeShareRecordReward` query function in `x/distribution/keeper/grpc_query.go`.

Recommendation

We recommend paginating the `WithdrawTokenizeShareRecordReward` and `TokenizeShareRecordReward` functions. We also recommend limiting the number of

`TokenizeShareRecord`s per owner and preventing the creation of new ones as well as transfers of existing ones to an owner that hits that limit.

Status: Partially resolved

The issue still exists in the `TokenizeShareRecordReward` query function.

4. Several errors are not checked

Severity: Major

In several places in the codebase, errors are not checked, and execution continues even if operations failed. This may cause an inconsistent state – a message handler may succeed and write changes to storage even though some operations failed.

Instances of not checked errors are in:

a) `x/distribution/module.go:73`: Error return value of `types.RegisterQueryHandlerClient` is not checked
b) `x/distribution/module.go:146`: Error return value of `cfg.RegisterMigration` is not checked
c) `x/distribution/types/msg.go:183`: Error return value of `sdk.AccAddressFromBech32` is not checked
d) `x/staking/genesis.go:43`: Error return value of `keeper.SetValidatorByConsAddr` is not checked
e) `x/staking/module.go:79`: Error return value of `types.RegisterQueryHandlerClient` is not checked
f) `x/staking/module.go:143`: Error return value of `cfg.RegisterMigration` is not checked
g) `x/distribution/types/msg.go:183`: Error return value of `sdk.AccAddressFromBech32` is not checked
h) `x/staking/keeper/msg_server.go:100`: Error return value of `k.SetValidatorByConsAddr` is not checked
i) `x/staking/keeper/msg_server.go:457`: Error return value of `k.ValidateUnbondAmount` is not checked
j) `x/staking/keeper/msg_server.go:477`: Error return value of `k.AddTokenizeShareRecord` is not checked
k) `x/staking/keeper/msg_server.go:550`: Error return value of `k.DeleteTokenizeShareRecord` is not checked
l) `x/staking/keeper/msg_server.go:598`: Error return value of `sdk.AccAddressFromBech32` is not checked
m) `x/staking/types/msg.go:380`: Error return value of `sdk.AccAddressFromBech32` is not checked
n) `x/staking/types/msg.go:414`: Error return value of `sdk.AccAddressFromBech32` is not checked

o) `x/staking/types/msg.go:442`: Error return value of `sdk.AccAddressFromBech32` is not checked
p) `x/slashing/genesis.go:19`: Error return value of `keeper.AddPubkey` is not checked
q) `x/slashing/module.go:81`: Error return value of `types.RegisterQueryHandlerClient` is not checked
r) `x/slashing/module.go:144`: Error return value of `cfg.RegisterMigration` is not checked
s) `x/slashing/keeper/hooks.go:35`: Error return value of `k.AddPubkey` is not checked
t) `x/slashing/keeper/hooks.go:69`: Error return value of `h.k.AfterValidatorCreated` is not checked

Recommendation

We recommend checking errors and aborting execution of message handlers if necessary.

Status: Resolved

5. Tokenize share record reward query does not include automatically withdrawn rewards

Severity: Major

The `TokenizeShareRecordReward` query function in `x/distribution/keeper/grpc_query.go` does not account for automatically withdrawn rewards that have been sent to the `TokenizeShareRecord`'s module account.

Recommendation

We recommend adding the current balance of each `TokenizeShareRecord`'s module account to the response. These balances should also be added if the validator has been removed and `val` in line `271` is `nil`.

Status: Resolved

6. Share token denoms may be all lower- or all upper-cased, depending on user input

Severity: Minor

The `getShareTokenDenom` function in `x/staking/keeper/msg_server.go` creates the denom for new share tokens through concatenation of the user-supplied validator address and a unique ID. The validator address is taken without transformations from the user's input. Since Bech32 addresses are valid in either all upper or all lower case, there can be two valid denoms for the same share tokens.

While this has not been found to cause any security issues in the current implementation, it may create problems for applications integrating with this module or lead to issues when further logic is added in the future.

Recommendation

We recommend transforming the user-supplied validator address by lower-casing it when creating the share token denom.

Status: Resolved

7. Usage of panics for control flow is bad practice

Severity: Minor

In several places in the codebase, panics are used for control flow. While this is no security issue since transactions recover from panics in Cosmos SDK, this may lead to problems in the future. For example, a caller of keeper functions may try to handle errors, while the function panics instead.

Instances of panic usage for control flow can be found in:

a) `x/distribution/keeper/allocation.go:33`
b) `x/distribution/keeper/delegation.go:35`, `40`, `48` and `127`
c) `x/distribution/keeper/hooks.go:50` and `97`
d) `x/distribution/keeper/invariants.go:87`
e) `x/distribution/keeper/keeper.go:38`
f) `x/distribution/keeper/store.go:51` and `70`
g) `x/distribution/keeper/validator.go:70`, `80` and `92`
h) `x/distribution/types/keys.go:65`, `79`, `94`, `108`, `122` and `136`
i) `x/distribution/types/msg.go:35`, `72`, `107` and `146`
j) `x/slashing/keeper/infractions.go:19` and `25`
k) `x/slashing/keeper/signing_info.go:109`, `121` and `125`
l) `x/slashing/types/msg.go:28`
m) `x/staking/keeper/alias_functions.go:68`
n) `x/staking/keeper/delegation.go:96`, `108`, `202`, `208`, `219`, `225`, `400`, `407`, `411`, `459`, `464`, `468`, `574`, `582`, `593`, `614` and `717`
o) `x/staking/keeper/invariants.go:63` and `106`
p) `x/staking/keeper/keeper.go:42`, `46` and `67`
q) `x/staking/keeper/pool.go:23` and `31`
r) `x/staking/keeper/querier.go:493`, `497` and `502`
s) `x/staking/keeper/query_utils.go:26`, `47`, `112` and `116`
t) `x/staking/keeper/slash.go:29`, `54`, `70`, `128`, `132`, `135`, `208`, `251`, `256`, `259`, `271`, `276`, `287`, `292` and `296`

u) `x/staking/keeper/val_state_change.go:30, 41, 45, 67, 71, 75, 139, 165, 233, 241, 249, 258, 267, 278` and `319`
v) `x/staking/keeper/validator.go:29, 50, 158, 162, 167, 316, 414, 427, 431` and `435`
w) `x/staking/types/authz.go:44`
x) `x/staking/types/delegation.go:51, 66, 73, 151` and `246`
y) `x/staking/types/historical_info.go:33`
z) `x/staking/types/keys.go:96`
aa) `x/staking/types/msg.go:75, 80, 179, 236, 290` and `345`
bb) `x/staking/types/params.go:94`
cc) `x/staking/types/validator.go:154, 262, 276, 381, 396, 400, 426` and `466`

Recommendation

We recommend replacing panics for control flow with returned errors.

Status: Acknowledged

The client decided to keep this pre-existing design pattern of the original staking module.

8. Addresses are not properly validated, which may cause panics and unexpected behavior

Severity: Minor

The `ValidateBasic` method, in `x/distribution/types/msg.go:47`, includes basic address validation. This validation only checks and raises an error if the `DelegatorAddress` or the `WithdrawAddress` provided is empty, but does not validate whether the input is a correctly formatted `Bech32` string. As an example, if `msg.DelegatorAddress=' '`, this check would not return an error, producing further panics and expected behavior.

Recommendation

We recommend properly validating that both addresses are valid `Bech32` strings using the SDK's method `AccAddressFromBech32`.

Status: Resolved

9. Different tokens for share records of the same validator can have different exchange rates after slashing

Severity: Informational

The staking module creates distinct `TokenizeShareRecord`s whenever a user tokenizes a delegation. These share records have all unique denoms, and their tokens are not fungible with each other, even if the validator is the same. While this simplifies the architecture of the module, it implies that the exchange rate between such records starts to diverge over time, specifically when tokens are minted after slashing occured. The reason for this is that during the `TokenizeShares` function in `x/staking/keeper/msg_server.go:454`, shares are minted without considering the current exchange rate of previous share records.

As an example, suppose `1000 ATOM` are delegated, of which `500 ATOM` are tokenized in record 1. There exist now `500 val...1` tokens. Now suppose a 10% slash happens, such that there are only `900 ATOM` in the delegation, and the `500 val...1` tokens are only worth `450 ATOM`. If now another `200 ATOM` of the same delegation are tokenized in record 2, there will be `200 val...2` tokens. As a result:

- `val...1` tokens have an exchange rate of `0.9 ATOM/val...1`, while
- `val...2` tokens have an exchange rate of `1 ATOM/val...2`.

This is not a direct security concern, but may confuse users and lead to problems if protocols built on top of the Liquidity Staking Cosmos SDK module do not consider these differences properly.

Recommendation

We recommend minting share record tokens that have the same value across all share records of the same validator. This can be achieved by applying the current ATOM/share record exchange rate when minting new share tokens.

Status: Acknowledged

10. Staking rewards are decoupled from share token holders

Severity: Informational

The current architecture separates holders of a tokenized delegation and the recipient of rewards for that delegation. A holder of share record tokens is not eligible for any rewards, instead these rewards accumulate in one single account per tokenized share record. The reward recipient is specified as the TokenizedShareOwner in the TokenizeShares function. It can be transferred using the TransferTokenizeShareRecord function.

While this is not a security concern, it may be unexpected that share record tokens do not contain eligibility for rewards.

Recommendation

We recommend clearly documenting this architecture.

Status: Acknowledged

The client states: "The reason for separating rewards from the liquid staking token is that it is by far the simplest design that is compatible with lazy computation of rewards that is fundamental to reward distribution design.

Alternative designs seem to require reward epoch and much more complex record keeping. We anticipate that Lido and Quicksilver will program rewards for participants to be sent a cosmwasm contract or or interchain account for each of their user interfaces.

For users who make use of the liquid staking tokens directly for an OTC trade then they can start claiming rewards just by redeeming their tokens back into shares."

11. Lack of event emission is bad practice

Severity: Informational

The `TokenizeShares`, `RedeemTokens` and `TransferTokenizeShareRecord` message handlers in `x/staking/keeper/msg_server.go` do currently not emit any events. The `Unjail` message handler in `x/slashing/keeper/msg_server.go` does emit an event, but does not include information about the action (unjailing).

Emitting events is a best practice, since it allows off-chain subscribers/indexers to track events.

Recommendation

We recommend adding events to these modules.

Status: Resolved

12. Tokenized share record related transactions and queries are only available via CLI, not REST

Severity: Informational

Transaction and query functionality for tokenized share records have been added to the CLI in `x/distribution/client/cli/tx.go`, `x/distribution/client/cli/query.go`, `x/staking/client/cli/tx.go` and `x/staking/client/cli/query.go`, but not to the REST interfaces in `x/distribution/client/rest/tx.go`, `x/distribution/client/rest/query.go`, `x/staking/client/rest/tx.go` and `x/staking/client/rest/query.go`.

Recommendation

We recommend adding transaction and query functionality for tokenized share records to the REST interface as well.

Status: Acknowledged

The client decided not to implement REST interfaces since they are deprecated in Cosmos SDK v0.46.

13. Storing the `Id` in `TokenizeShareRecord` is inefficient

Severity: Informational

The `TokenizeShareRecord` contains the `Id` of the record. That is unnecessary since every lookup of the record implies that the ID is known, since it's part of the key.

Recommendation

We recommend removing the `Id` from the `TokenizeShareRecord`.

Status: Acknowledged

14. Storing the `ShareTokenDenom` in `TokenizeShareRecord` is inefficient

Severity: Informational

The `TokenizeShareRecord` contains the `ShareTokenDenom` of the record. That is unnecessary, since the `ShareTokenDenom` can be derived from the stored `Validator` and `Id`.

Recommendation

We recommend removing the `ShareTokenDenom` from the `TokenizeShareRecord`.

Status: Resolved

15. Specification is outdated

Severity: Informational

The specification is outdated in several ways:

a) `x/staking/spec/01_state.md` is missing an entry for `0x64` for `LastTokenizeShareRecordIdKey`.
b) `x/staking/spec/*` still describes queues, which have been removeg from the module.

Recommendation

We recommend updating the specification.

Status: Acknowledged

16. Unused code negatively impacts maintainability

Severity: Informational

The code base contains unused code. Unused code increases the code size and hence inhibits maintainability. Instances of unused code are:

a) The `NewStakeAuthorization` function in `x/staking/types/authz.go`.
b) The `AllInvariants` function in `x/distribution/keeper/invariants.go`.
c) The `AllInvariants` function in `x/staking/keeper/invariants.go`.
d) The `ValidatePowerReduction` function in `x/staking/types/params.go`.

Recommendation

We recommend removing unused code.

Status: Acknowledged