



# **Security Audit Report**

## **Liquid Staking Celestia Intregation**

Authors: Marko Juric, Mirel Dalcekovic

Last revised 30 January, 2024

# Table of Contents

<b>Audit Overview .....</b>	<b>1</b>
The Project	1
Scope of this report	1
Conducted work	1
Conclusions	2
Further Increasing Confidence	2
Disclaimer	2
<b>Audit Dashboard .....</b>	<b>3</b>
Severity Summary	3
Resolution Status Summary	4
<b>Threat Inspection .....</b>	<b>5</b>
<b>Technical Specification Review .....</b>	<b>6</b>
Invariants:	6
Conclusions	6
<b>Findings .....</b>	<b>11</b>
Minor Issues Identified in x/staketia	13
Assumptions in Operator Processes	15
Enhancements in the Design of Active and Archived Stores	19
Fee Account Initialization in x/staketia Module	21
Liveness Issue in Liquid Staking Arising from the Inability to Unpack IBC Acknowledgment Messages in Stride	23
Implement Redemption Rate Change Validation in LiquidStakeAndDistributeFees	25
Monitoring Suggestions	27
Operator's Ability to Insert Records in Active Store for Future Epochs	28
Authorization Grants Setup for Mainnet Accounts	30
Static Analysis Results by Gosec Tool for x/staketia Module	32
Halting Claim Distribution for Unbonded Users in the Absence of Adequate Funds	33
Exclusion of DELEGATED_COMPLETED Records from Active Store	35
<b>Appendix: Vulnerability Classification .....</b>	<b>37</b>
Impact Score	37
Exploitability Score	37

Severity Score

# Audit Overview

## The Project

In January 2024, Stride extended its collaboration [Informal Systems](#) for a partnership and security audit of the stTIA Multisig solution, integrating liquid staking with Celestia.

The scope of this phase includes:

- the x/staketia module,
- stTIA Multisig technical specification, and
- authorization of grants to multi-signature accounts in the system.

## Scope of this report

The agreed work plan included the following tasks:

1. Technical specification review,
2. x/staketia codebase review, spanning multiple commit hashes and pull requests and helping with reaching the best design and solution, and
3. Analysis of multisig account permissions setup on the mainnet.

## Conducted work

During the kick-off meeting, Informal Systems and Stride team representatives established the audit process and devised an optimal project organization approach, which includes:

1. Synchronizing regularly, as often as possible.
2. Enhancing the availability and dedicating overtime of Informal team members.
3. Deciding that Informal auditing team will share draft versions of findings and communicate potential threats or issues over Slack promptly. Acknowledging the possibility of false positives, the emphasis is on a collaborative approach to save time, avoiding the necessity for 100% issue confirmation.

The Stride team provided onboarding materials in the form of Loom videos.

## Timeline Summary:

1. **2024.01.18 (Thursday):** Kickoff meeting covered scope, timeline, organization, and an introduction to the project. Planned two days for reviewing technical specifications and another two for code inspection tasks.
2. **2024.01.20 (Saturday):** Meeting with Stride team representative. Informal asked questions and clarified expected goals.
3. **2024.01.23 (Tuesday):** Sync meeting discussed the review of technical documentation. Conclusions and additional failure cases communicated over Slack. It was agreed that the Informal auditing team should continue with their work during the entirety of Thursday. The Stride team planned to merge the audited private repository branch in the main of the public repository, by the end of Thursday, January 25th.
4. **2024.01.25 (Thursday):** Sync meeting discussed code inspection task results. Communication of possible issues and concerns shared over Slack between meetings.
5. **2024.01.26 (Friday):** Review and analysis of agreed additional tasks for Informal Systems, including sharing possible solutions for the fee-account module account creation.
6. **2024.01.29 (Monday):** Draft report generated and shared with the Stride team.
7. **2024.01.30 (Tuesday):** Final report generated, after consulting the Stride team about the form and internal Informal review.

## Conclusions

Considering time constraints and the revised collaborative code review approach during the development sprint, the Informal Systems team believes they've maximized efforts to accelerate development and enhance the design of the newly established multisig approach for liquid staking integration with Celestia.

The solution is temporary and will be abandoned upon Celestia's integration of the ICA host submodule. Subsequently, migration to the existing solution utilizing ICAs will take place. Despite potential flaws in the multisig approach's design, the trade-offs involved prioritized minimal new code for essential functionality and reduced complexity.

The off-chain operator process drives the entire inter-chain liquid staking and unstaking, holding significant power to address issues arising from its irregular and unexpected operations. The primary audit goal was to ensure the correctness of the on-chain logic ([x/staketia](#) implementation) without requiring coverage of additional use cases.

## Further Increasing Confidence

To enhance confidence in the design and implementation, a recommended step would be to revisit the audited scope with standard auditing process Informal uses: focusing on the fixed commit hash and the latest technical specifications. Given the substantial impact on the overall liquid staking integration with Celestia, consideration should be given to extending the scope to include the operator process.

While this audit concentrated on technical specification review, manual code review, and manual protocol analysis and reconstruction, we usually suggest increasing the confidence through additional formal measures. This may involve more rigorous steps such as automated model checking and model-based adversarial testing.

Considering the temporary nature of this solution, the Stride team may determine that a reevaluation is unnecessary, and formal model-based auditing may not be deemed essential.

## Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an "as is" basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bug free status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an "endorsement", "approval" or "disapproval" of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client's business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.

# Audit Dashboard

## Target Summary

- **Type:** Specification and Implementation
- **Platform:** Go
- **Artifacts:**
  - [stTIA Multisig Tech Spec](#) - adaptations were done during the development.
  - private repository [stride-staketia](#) branch [sttia](#).
  - **Commit hash/tag:**
    - not fixed
    - several commit hashes: [fb0d964](#), [bc303a1](#), [43156c1](#)
    - PRs:
      - [ConfirmUndelegation](#)
      - [Removed archive status from delegation records](#)
      - [Removed archive status from unbonding records](#)
      - [Add HaltZone function](#)
      - [Changes to CLI txs](#)
      - [Audit batch 1](#)
      - [Liquid stake fees](#)
      - [Return estimated unbonding time](#)
      - [add authz scripts \(wrong addresses\)](#)
      - [Audit batch 2](#).
  - review of txs used to set up the grants to the multisig accounts created on Celestia mainnet.

## Engagement Summary

- **Dates:** 18.01.2024. to 29.01.2024.
- **Method:**
  - Manual code review and protocol analysis were carried out during the Stride development sprint, covering the code base as it underwent development.
- **Employees Engaged:** 2

## Severity Summary

!! The findings result from collaborative analysis between Informal Systems and Stride team representatives. A notable portion of the audit findings falls under "Unknown severity" due to potential issues arising from invalid operations of the operator process (which was not within the scope of this audit). The frequency and occurrence of these issues are uncertain, but if they arise, their impact is considered more significant.

Finding Severity	#
Critical	0
High	0
Medium	0
Low	1
Informational	6

<b>Finding Severity</b>	<b>#</b>
Unknown	5
<b>Total</b>	12

## Resolution Status Summary

<b>Resolution Status</b>	<b>#</b>
Acknowledged	6
Resolved	3
Functioning as Designed/Disputed	3
<b>Total</b>	12

## Threat Inspection

During the initial stage of the audit the auditing team spent time onboarding to the project's scope and identifying potential failure cases and threats.

Analysis of the possibility of realization of those threats was done during threats (code) inspection and during discussions with Stride team. Our primary objective was to validate the design's correctness and identify any issues in the code base that could result in the realization of these threats. To achieve this, we held sync meetings with the Stride team to discuss possible flaws in the system and gain insights into the current behavior of the system. Through these efforts, we were able to validate that the identified threats could not/could be realized.

The following analysis within this page and under the [Tech spec review](#) and [Operator process assumptions](#) contain our conclusions and the explanation of the audited product design and implementation on how those threats are mitigated.

In addition to our threat inspection analysis, we have documented our results in [Findings](#).

### Threats:

- Could depositing directly into the deposit account potentially trigger reaching the redemption rate bounds?
  - **Attack vector:** The risk involves the funds entering the Stride deposit account without passing through Liquid Staking. This could cause a sudden jump in the delegated balance, leading to the redemption rate being calculated as out of defined outer and/or inner bounds and consequently halting Liquid Staking.
  - **Conclusion:** A malicious user would need to send a non-refundable, likely substantial amount of TIAs to disrupt liquid staking. There is a [ResumeHostZone](#) action available, introducing a delegation and undelegation delay of at most one day. Monitoring might detect this transaction before halting the zone. Furthermore, once the host is resumed, it could result in increased yield for all users.
- IBC transfer callbacks correctness. In case that IBC acknowledgment can not be unmarshald, liquidity staking will be halted and need the manual intervention.
  - **Attack vector:** Malicious Celestia chain, versions upgrade on Celestia app - marshaling changed.
  - **Conclusions:** Described more with [Liveness Issue in Liquid Staking: Poorly Constructed IBC Acknowledgment Message on Host Side](#).



## Technical Specification Review

During the technical specification [stTIA Multisig Tech Spec](#) review, the primary focus for the Informal auditing team was to identify potential failure cases. The team categorized the potential root causes that could result in failure, and the subsequent analysis was centered around these identified factors:

1. operator actions ORDERING → **ORDERING ISSUES**
2. compromised operator keys → **COMPROMISED KEY ISSUES**
  - a. possibility of staking/unstaking the wrong amount
  - b. sending the wrong amount of undelegated tokens back to Stride
3. errors in operator code → **OPERATOR SCRIPT BUGS**
4. time delays between on-chain operations and on-chain operator triggering actions and signatures collected → **TIME-DELAY ISSUES**
5. records manipulation on Stride - updates and state changes by the operator process were incorrect → **INVALID RECORDS**
  - a. SAFE account will probably have the possibility to update records directly!
  - b. Verification of record status changes as described in the specification (implementation to follow, at the moment of reviewing the spec).

The analysis was conducted with the following guiding principles:

1. **No assumptions:** The examination was carried out without making any presumptions, requiring the identification of recovery scenarios for each potential failure case.
2. **Assume operator does everything wrong:** The analysis considered a scenario where the operator makes every possible mistake, leading to potential outcomes such as:
  - Loss of funds
  - Loss of yield

As per the information shared by the Stride team based on their experience, from the user standpoint, delayed unbondings result in frustration after 10 days, and delayed yield after more than 2 days contributes to user dissatisfaction.

### Invariants:

- Tokens are exclusively located in either the deposit account or the delegation records (undelegated balance on Stride), it is of crucial importance not to keep dual tracking of tokens to avoid impacting the Redemption Rate (RR).
- The sum of all delegation records and the rewards equals the total amount of TIA tokens held by the Stride protocol.
- The redemption rate calculation should precede the epochs triggering delegation and undelegation.
- Only one delegation IBC transfer is allowed to be in progress at any given time.
- If an IBC transfer is in progress, the delegation record status should be designated as "TRANSFER\_IN\_PROGRESS."
- Upon the successful completion of an IBC transfer, a corresponding record is updated to the next status.

## Conclusions

The following flows were analyzed for potential failure cases. This paragraph encompasses discussion notes from Informal and Stride sync meetings and includes references to specific failure cases outlined by the Stride team in the tech spec document.

1. **Staking flow:**
  - IBC transfer fails → delegation record is deleted

```
def TransferCallback(success: bool, recordId: int):
    """
    Callback for the transfer of native tokens to the host zone
    """
    # If successful, flag the record as DELEGATION_QUEUE
    if success:
        k.UpdateDelegationRecordStatus(recordId, DELEGATION_QUEUE)
    else:
        # Otherwise remove the record
        # Tokens will be refunded to the module account and
        # Re-tried next epoch
        k.RemoveDelegationRecord(recordId)
```

#### Questions:

- How is the retrieval of stTIAs from the user and their exchange for TIA handled?
- In the edge case where there are no TIAs in the balance when redemption is triggered by the user, will the transaction fail? →  
Monitoring this situation and taking action, once the cause of IBC transfer failures is addressed, all deposited tokens will eventually be delegated to Celestia. The only consequence would be that users will be temporarily excluded from yield.
- Is there an option for users to cancel the liquid stake? →  
The Stride team noted that this feature is not currently implemented and would significantly increase complexity.

#### 2. Reinvestment subflow:

- it is not clear from the specification will there be a record created for rewards sent to the Stride and ready to be reinvested?
- Will it be possible for the user to claim the rewards on Stride right away or **they are automatically staked?**
- redemption rate is calculated with delegated records and amounts held within these records on chain - so there must be some new record created or the redemption rate will be invalid? (check) → Riley, answer: there will be **new delegation record created, same as for the user staked amounts.**

#### 3. Redemption rate is calculated on daily epochs, but the amounts impacting it are, as shown:

$$\text{Redemption rate} = \frac{(\text{DepositAccount.Balance} + \text{DelegationRecord.TRANSFER\_IN\_PROGRESS} + \text{DelegationRecord.DELEGATION\_QUEUE} + \text{HostZone.Delegated Balance})}{\text{stToken supply}}$$

Rewards are collected on the Celestia delegation account, but claimed daily by the operator process and moved to the Celestia rewards MS account.

Since only daily, these rewards will be reinvested/delegated once again from the Stride chain to Celestia validators.

#### Question:

It seems that the redemption rate calculation could not include rewards amount if they are on the rewards account in the moment of the calculation? Or the `HostZone.DelegatedBalance` will somehow include the rewards account balance as well?

**Conclusion:** This is correct but, the updated redemption rate is important only prior to delegation and undelegations are triggered. All three happen on an epoch and the RR update is first, so delegations and undelegations will be created with the appropriate RR.

In the event of a liquid staking zone halt, the SAFE multisig account has the capability to execute `RefreshRedemptionRate` as required. It's important to note that the protocol does not incorporate rewards amounts into redemption rate calculations until they are reinvested or delegated.

#### 4. **UndelegateFlow:** `MsgConfirmUndelegation`

The Stride team asked Informal auditing team to focus on this flow.

```
def MsgConfirmUndelegation(recordId: int, txHash: str): // MS tx from CONTROLLER
    """
    Confirms that an undelegation has been submitted and updates
    the corresponding record status from UNBONDING_QUEUE -> IN_PROGRESS
    """

    # Update the record to UNBONDING_IN_PROGRESS and set the unbonding time
    unbondingRecord = k.GetUnbondingRecord(recordId)
    unbondingRecord.Status = IN_PROGRESS
    unbondingRecord.UnbondingTime = ctx.BlockTime() + hostZone.UnbondingPeriod
    unbondingRecord.UndelegationTxHash = txHash
    k.SetUnbondingRecord(unbondingRecord)

    # Burn stTokens
    k.BankKeeper.BurnCoins(unbondingRecord.StTokenAmount)

    # Decrement total delegated balance
    hostZone = k.GetHostZone()
    hostZone.DelegatedBalance -= unbondingRecord.NativeAmount
    k.SetHostZone(hostZone)
```

Undelegation should burn stTIA and reduce delegated amount in `HostZone.DelegatedBalance`.

**Possible Issues:** `DelegatedBalance` is less than `unbondingRecord.NativeAmount` → Redemption rate (RR) impact, so RR could move outside the bounds and produce a Stride liquid staking halt.

**Reason:** somehow the delegated records (balance) were “set off”:

- records were updated with SAFE MS account to hold less tokens
- so the **DelegatedBalance goes into negative** → Redemption rate calculated could also be negative?

**Fix to this potential failure case would be:** update delegated records with issuing the **slash** with a positive amount, equal to the diff between the correct slash and wrong slash amount.

**Comment:** Informal team is not sure if this is possible! Maybe due to wrong slashing amount sent to the Stride. While this failure case is not explicitly included in the table in the tech spec document, it is detailed in the notes under 15).

**The Stride team comment:** there is a check that this undelegate value can not be negative in the code base! This check should be added prior to issuing the slash tx. This is also shared in <https://informalsystems.atlassian.net/wiki/spaces/S2QDI/pages/edit-v2/232522531> and it was resolved in the meantime.

#### 5. **CheckUnbondingFinished (BeginBlocker execution)**

Set unbondingTime upon confirmation that unbonding is triggered on Celestia:

```
unbondingRecord.UnbondingTime = ctx.BlockTime() + hostZone.UnbondingPeriod
```

**Concern:** is it possible to mark unbonding record as UNBONDED, but the unbonding isn't finished on the Celestia?

**Conclusion:** Only if operator sends the tx to confirm the `MsgConfirmUndelegation` prior to actually executing the unbonding tx on Celestia → The amount of stTIA would be burn, native tokens reduced on `HostZone.DelegatedBalance` and with `CheckUnbondingFinished` it could easily happen that the unbonding is not finished still (period has not passed on Celestia) but the record will be marked as UNBONDED.

```
def CheckUnbondingFinished(): // begin blocker
    """
    Checks if any of the unbonding records have finished unbonding
    by checking against the current block time.
    Updates the status from IN_PROGRESS -> UNBONDED if so
    """
    for unbondingRecord in k.GetAllUnbondingRecordsWithStatus(IN_PROGRESS):
        if ctx.BlockTime() > unbondingRecord.UnbondingTime:
            unbondingRecord.Status = UNBONDED
            k.SetUnbondingRecord(unbondingRecord)
```

**FIX to this potential failure case would be:** Once the actual unbonding happens the records do not have to be updated.

**Comment:** This failure case is not in the table, but there are comments pointing to this failure case in the notes. (13. Multisig updates records and marks them as "unstaked" before they were unstaked).

6. `MsgConfirmUnbondedTokensSwept` sent prior to the actual tokens are sent back to the Stride, so the claimable amount could not be present in the `hostZone.ClaimAddress`.

Distribution of claims will fail: `def DistributeClaims(): // begin blocker`

```
def MsgConfirmUnbondedTokensSwept(recordId: int, txHash: str): // MS tx from CONTROLLER
    """
    Confirms that the unbonded tokens have been transferred back to stride
    and updates the record status from UNBONDED -> CLAIMABLE
    """
    unbondingRecord = k.GetUnbondingRecord(recordId)
    if unbondingRecord.Status != UNBONDED:
        return error

    unbondingRecord.Status = CLAIMABLE
    unbondingRecord.SweepTxHash = txHash
    k.SetUnbondingRecord(unbondingRecord)
```

Could the protocol end up in a situation that we are always late with having the needed amount to claim - since the new records are marked as claimed over and over again?

**FIX to this potential failure case would be:** go through claimable unbonding records - check txHashes to confirm the sweep has happened and then revert statuses for those not covered with tx.

**Comment:** This is the #8 failure case from the failures table in the tech spec document.

7. **Spec misses the design details about the reward & stride fees split**

Will the SAFE MS account send the fee to the fee module account on Stride and the rewards flow remains the

same (as explained: after claimed by the SAFE MS acc, IBC transfer rewards to Stride deposit acc)?

**The Stride team comment:** This part needs to be designed and added to the spec.

9. **Ordered operator steps and CELESTIA MS DELEGATION MS ACC?**

What can be found on the CELESTIA MS DELEGATION MS ACC? - Case when: Unbonding is finished on Celestia and there is amount of tokens waiting to be delegated as well as some rewards collected. Operator should:

1. claim the rewards from CELESTIA REWARDS acc → **Stride team, update:** MAYBE THIS WOULD go LAST!
2. IBC transfer the undelegated amount to the Stride and mark the
3. delegate the amount from delegation account (Celestia - should be same as from the delegation records)
4. start undelegate

**The reasoning about the ordering:**

- the biggest problem is when we delegate more than we should.
- users want their yields on time (with no delay > 2days)
- unbonding shouldn't be late more than >10days (in case we undelegate less then we should, we might have some time to fix this)
- delegation is anyways waited for at least 4 days - enough time to fix the issue if wrong delegation amount was bonded.

**The Stride team comment:** This would be designed and spec out when implementing operator process.

10. **Higher amount of rewards could influence a redemption rate to get out of bounds when transferred to a Deposit account**

Considering the case when rewards are IBC transferred from HostZone (Celestia) -> Stride we have thought about a possible scenario which could be triggered by operator, apart from the situation where it can somehow send rewards to a completely random address (perhaps no way back in that case):

- a. technically a higher amount of rewards could influence a redemption rate to get out of bounds when transferred to a Deposit account
- b. especially if operator didn't claim rewards for some time and reward balance got higher

**FIX to this potential failure case would be:** To introduce a calculation of "potential" redemption rate value before actually triggering sending rewards to Deposit account? And if the risk to get out of bounds is high, maybe smaller amount of rewards could be sent, and the other portion in the next epoch?

**The Stride team comment:** Acknowledged, and this will be added to the operator offchain process as a check performed prior to sending the rewards back to the Stride deposit account.

**Comment:** Also added in the [Assumptions in Operator Processes](#) finding.

11. Informal team also reviewed failure cases detected by the Stride team.

## Findings

Title	Type	Severity	Impact	Exploitability	Status	Issue
Minor Issues Identified in x/staketia	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	RESOLVED	<a href="https://github.com/Stride-Labs/stride-staketia/pull/61">https://github.com/Stride-Labs/stride-staketia/pull/61</a>
Enhancements in the Design of Active and Archived Stores	PROTOCOL	0 INFORMATIONAL	0 NONE	0 NONE	ACKNOWLEDGED	
Fee Account Initialization in x/staketia Module	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	RESOLVED	<a href="https://github.com/Stride-Labs/stride/pull/1091">https://github.com/Stride-Labs/stride/pull/1091</a>
Implement Redemption Rate Change Validation in LiquidStakeAndDistributeFees	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	DISPUTED	
Static Analysis Results by Gosec Tool for x/staketia Module	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	ACKNOWLEDGED	
Exclusion of DELEGATED_COMPLETE Records from Active Store	PROTOCOL IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE	RESOLVED	<a href="https://github.com/Stride-Labs/stride-staketia/pull/69">https://github.com/Stride-Labs/stride-staketia/pull/69</a>
Authorization Grants Setup for Mainnet Accounts	IMPLEMENTATION	1 LOW	1 LOW	1 LOW	ACKNOWLEDGED	
Assumptions in Operator Processes	PROTOCOL	04 UNKNOWN	3 HIGH	04 UNKNOWN	ACKNOWLEDGED	
Liveness Issue in Liquid Staking Arising from the Inability to Unpack IBC Acknowledgment Messages in Stride	PROTOCOL	04 UNKNOWN	3 HIGH	04 UNKNOWN	ACKNOWLEDGED	

Title	Type	Severity	Impact	Exploitability	Status	Issue
Monitoring Suggestions	PROTOCOL	04 UNKNOWN	3 HIGH	04 UNKNOWN	ACKNOWLEDGED	
Operator's Ability to Insert Records in Active Store for Future Epochs	IMPLEMENTATION	04 UNKNOWN	2 MEDIUM	04 UNKNOWN	DISPUTED	
Halting Claim Distribution for Unbonded Users in the Absence of Adequate Funds	PROTOCOL	04 UNKNOWN	2 MEDIUM	04 UNKNOWN	DISPUTED	

## Minor Issues Identified in x/staketia

<b>Title</b>	Minor Issues Identified in x/staketia
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	0 INFORMATIONAL
<b>Impact</b>	0 NONE
<b>Exploitability</b>	0 NONE
<b>Status</b>	RESOLVED
<b>Issue</b>	<a href="https://github.com/Stride-Labs/stride-staketia/pull/61">https://github.com/Stride-Labs/stride-staketia/pull/61</a>

### Involved artifacts

- [x/staketia/types/messages.go](#)
- [x/staketia/keeper/redemption\\_rate.go](#)
- [x/staketia/keeper/delegation.go](#)
- [proto/stride/staketia/tx.proto](#)
- [x/staketia/types/staketia.pb.go](#)
- [x/staketia/keeper/hooks.go](#)
- [x/staketia/keeper/unbonding.go](#)
- [x/staketia/keeper/msg\\_server.go](#)

### Description

The following minor issues or suggestions for in-line code comments improvements do not pose a security threat, but could improve the code quality and clarity.

During analysis we were analyzing:

- private repository [stride-staketia](#) branch [sttia](#)

**Shared on January 23rd, for commit hash: [fb0d964](#)**

- Error messages should be unified in case of `MsgLiquidStake` [here](#) and `MsgRedeemStake` [here](#), when minimum amount is not valid. In case of `MsgLiquidStake` the error is copied from `x/stakeibc` module message.
- Outer bounds [check](#) inside `CheckRedemptionRateExceedsBounds` seems unnecessary. `ValidateRedemptionRateBoundsIntialized` already makes sure that inner bounds cannot be set beyond outer bounds. There cannot be a situation where check will fail for outer bounds but not for inner bounds.



- Seems like there are separate `DelegationRecords` stores: active and archive. There are several [places](#) where delegation record is updated in active store in `SetDelegationRecord` but then removed from active store with `ArchiveDelegationRecord`. Was the intended scenario to update the field from valid active store record, but store it in archive store and remove from active store? Currently, seems like input values like `txHash` will be lost.

**Stride team comment:** acknowledged and resolved in [Audit batch 1](#).

**Shared on January 24th for commit hash:** [bc303a1](#)

- APIs differ for `x/stakeibc` and `x/staketia` `MsgRedeemStake` - there is no possibility of defining receiver for `staketia MsgReedemStake` while stakeibc has this [argument](#). This is not an issue, but not sure if this option needs to be provided.

**Stride team comment:** implemented as designed, on purpose. No changes will be implemented.

**Shared on January 25th for commit hash** [43156c1](#)

- fix in-line code comments, for clarity:
  - days vs secs: [here](#)
  - add mint epoch and fee distribution step [here](#)
  - fix comment: [here](#)
  - `AdjustDelegatedBalance` -> Suggestion: `rename msg.Operator -> msg.Signer / Creator`, since the SAFE multi signature account will be the one signing the broadcasting of this message.

**Informal team note:** Currently, comments remain unaltered at the time of report generation. These modifications are considered beneficial but optional, findings will be marked as resolved.

## Recommendation

As suggested above.

## Assumptions in Operator Processes

<b>Title</b>	Assumptions in Operator Processes
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	PROTOCOL
<b>Severity</b>	04 UNKNOWN
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	04 UNKNOWN
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Failure cases & operator monitoring to prevent those

This problem encapsulates several failure cases identified by the Informal team during the analysis of the technical specification. The Stride team's analysis already encompassed a substantial number of failure scenarios that might arise if the operator process behaves unpredictably or if there is a time delay between the execution of messages on the Celestia (host) chain and the corresponding confirmations sent to the Stride (controller chain).

The ones listed below are additional to those already identified:

Higher amount of rewards could influence a redemption rate to get out of bounds when transferred to a Deposit account

Scenario:

Considering the case when rewards are IBC transferred from HostZone (Celestia) back to the Stride chain. We have thought about a possible scenario which could be triggered by operator, apart from the situation where it can somehow send rewards to a completely random address (perhaps no way back in that case):

1. technically a higher amount of rewards could influence a redemption rate to get out of bounds when transferred to a Deposit account
2. especially if operator didn't claim rewards for some time and reward balance got higher

Did you consider to introduce a calculation of "potential" redemption rate value before actually triggering sending rewards to Deposit account? And if the risk to get out of bounds is high, maybe smaller amount of rewards could be sent, and the other portion in the next epoch?

Operator process:

Prior to sending the IBC transfer rewards to Stride deposit account:

1. calculate the "expected next redemption rate" off-chain,
2. and alert if there's a high jump leading to zone halt,

- then, the BOUNDS multi signature account can adjust the inner bounds up, to accommodate the upcoming jump. The redemption rate should only be recalculated once per day, so we'd have some time to catch it.

#### Additional things to think off:

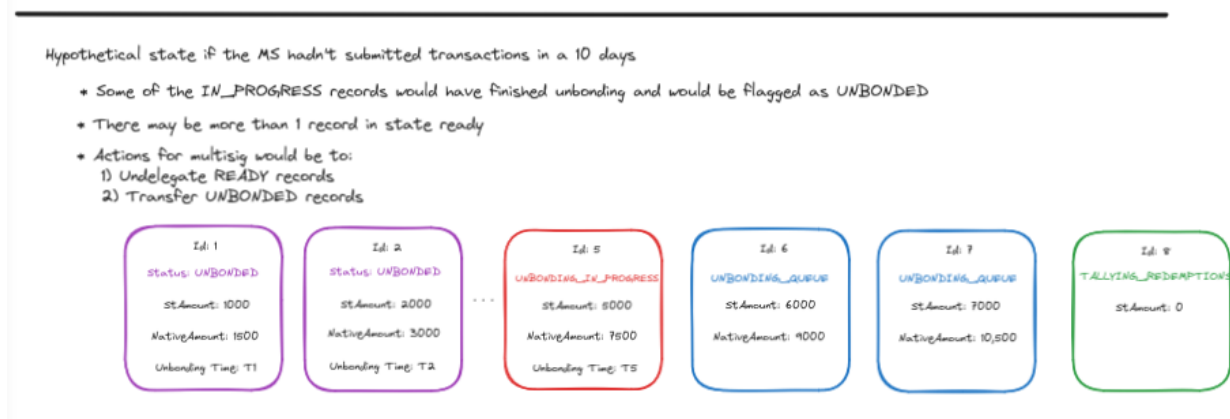
- Could the adaptation of inner bounds hit the outer bounds limit so it is impossible to update them without gov proposal changes?
  - Do we want these?
  - Do we split rewards in these cases?
- Monitor and track off-chain the amount of rewards transferred to the Stride deposit account. This is also listed as possible Monitoring improvement finding.

#### Stride team Comment:

Team plans to add monitoring or rewards influence to redemption rate jumps to their monitoring stack.

More than one accumulating records is status **UNBONDING\_QUEUE**

Scenario:



- There was no unbonding on Celestia for more than 4 days - accumulating records will remain in the **UNBONDING\_QUEUE**.
- Once the operator triggers the unbonding - it will probably read all the accumulating records and could
  - Trigger the unbonding for each of them, which means this could introduce the issue with having the limit of max 7 concurrent unbondings in parallel limit - leading to some of the upcoming undelegations will have to wait.
  - Batch up the already batched up accumulating records in **UNBONDING\_QUEUE**.

#### Operator process:

- create only one unbonding tx summing up all the **UNBONDING\_QUEUE**
- mapping of batched accumulating records processed with one transaction?
  - accumulating records have record ids equal to the unbonding epoch id - this is used as identifier for manipulation further in the undelegation flow
  - upon undelegation is finished on Celestia, prior to sending the **ConfirmUndelegation** back to Stride - unpack the batch processed
    - operator should send back to the Stride all the epochs processed should - **ConfirmUndelegation**

one confirm per accumulating record id

#### Additional things to think off:

- Operator monitors the amount of tokens unbonded per recordId.
- `ConfirmUndelegation` accepts the `unbondedAmountOnCelestia` (name e.g.) as an argument and performs validation check? If the amounts are not the same this could be the indication of a potential operator manipulation error.

Stride team comments:

More than one accumulating records is status `UNBONDING_QUEUE`.

- To clarify some of the terminology:
  - We refer to the records as **unbonding** records. The **accumulating** record is an unbonding record in status `ACCUMULATING_REDEMPTIONS` which is the status the record is originally created in
  - If there are no unbondings for an epoch, the "accumulating" record is archived
  - There should only ever be one record in status `ACCUMULATING_REDEMPTIONS` at a time
  - There can be multiple `UNBONDING_QUEUE` records at a time if the operator has not submitted the undelegations
- With respect to the issue:
  - Because we submit unbondings every 4 days, we'll actually only have 5 or 6 concurrent unbondings active at once, which gives us a buffer of 4-8 days
  - **If we do notice that we are behind by more than 2 records, we can batch them up off chain and submit one undelegation tx, but then multiple confirm txs.** Given how unlikely it is that we fall that far behind, I think this approach is desirable to making accommodations on chain.

#### Possible ways of DelegatedBalance going to negative values

Scenario:

- after unbonding
- after slashing

```
def MsgConfirmUndelegation(recordId: int, txHash: str): // MS tx from CONTROLLER
    """
    Confirms that an undelegation has been submitted and updates
    the corresponding record status from UNBONDING_QUEUE -> IN_PROGRESS
    """

    # Update the record to UNBONDING_IN_PROGRESS and set the unbonding time
    unbondingRecord = k.GetUnbondingRecord(recordId)
    unbondingRecord.Status = IN_PROGRESS
    unbondingRecord.UnbondingTime = ctx.BlockTime() + hostZone.UnbondingPeriod
    unbondingRecord.UndelegationTxHash = txHash
    k.SetUnbondingRecord(unbondingRecord)

    # Burn stTokens
    k.BankKeeper.BurnCoins(unbondingRecord.StTokenAmount)

    # Decrement total delegated balance
    hostZone = k.GetHostZone()
    hostZone.DelegatedBalance -= unbondingRecord.NativeAmount
    k.SetHostZone(hostZone)
```

Operator:

- A pre-check occurs before sending the ConfirmUndelegation Msg to Stride by the operator, ensuring that the delegated balance doesn't go into negative.
- Before sending the slash transaction to Stride, verify whether the slashing amount would result in a negative delegated balance.

#### Stride team Comment:

Regarding the negative delegated balance after slash - this is acknowledged and is resolved in [“audit batch 1” commit](#).

## Enhancements in the Design of Active and Archived Stores

<b>Title</b>	Enhancements in the Design of Active and Archived Stores
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	PROTOCOL
<b>Severity</b>	0 INFORMATIONAL
<b>Impact</b>	0 NONE
<b>Exploitability</b>	0 NONE
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Involved artifacts

- Active and archived stores and statuses of the records (redemption, delegation records) design - from:
  - [stTIA Multisig Tech Spec](#)
  - implementation in private repository [stride-staketia](#) branch [sttia](#)

### Description

Current design was analyzed by the Informal systems team. Even though we could not find any issues, the solution could be improved.

The current solution is based on the following:

- The stTIA multisig approach will be in use for less than 6 months, after which it will transition to the standard liquid staking solution.
- There are no justified concerns about state bloats, with fewer than 100 redemption requests per day (typically between 5 and 50, as communicated by the Stride team).
- The development team aims to make minimal changes and additions to the code base.

Also, there was an additional issue: <https://informalsystems.atlassian.net/wiki/spaces/S2QDI/pages/238682120/Active+store+should+not+contain+DELEGATED+COMPLETED+records> reported closely related to this one.

### Recommendation

Suggestions for a more elegant approach:

- Utilize a single store.
- Expand keys to consist of: key prefix + record status + record ID.
- When iterating and querying records with a specific status, there will be no need to iterate through a large number of records.
- Purge records older than a set number of epochs, determined by the Stride team based on how long they believe records should be on the chain to address failure cases through operator process.

## Resolution

The Stride team has acknowledged the suggested approach for potential future designs.

## Fee Account Initialization in x/staketia Module

<b>Title</b>	Fee Account Initialization in x/staketia Module
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	0 INFORMATIONAL
<b>Impact</b>	0 NONE
<b>Exploitability</b>	0 NONE
<b>Status</b>	RESOLVED
<b>Issue</b>	<a href="https://github.com/Stride-Labs/stride/pull/1091">https://github.com/Stride-Labs/stride/pull/1091</a>

### Involved artifacts

- [stTIA Multisig Tech Spec](#)
- fee account implementation in [PR](#)

### Description

During the analysis it was unclear will the fee account be module account or the MS account? [ref](#)  
The auditing team was confused with the following:

- no fee address in `DefaultGenesis`,
- [here](#) on `InitGenesis` we create module account if there is no `FeeAddress` on `HostZone`,
- [here](#), `ValidateGenesis` checks if `FeeAddress` is defined - invalid state if not.

The Stride team mentioned that they considered both options during the design phase, but ultimately settled on the module account as the final solution and asked for suggestions on how to set this up.

### Recommendation

The current approach of the Stride team involves using custom Stride [utils](#) functions to create module subaccounts. These subaccounts are employed for mapping ICA accounts from the host zones to the x/stakeibc module (sub)accounts on Stride.

Upon analyzing the current implementation, we found it wasn't deemed faulty. However, we've identified a simpler method to initialize the module account with the upgrade, specifically for the newly added `x/staketia` module

- use `GetModuleAccount(ctx, "new-module-address-name")`,
- and add the appropriate permissions for the `"new-module-address-name"` module account [here](#).



## Resolution

The Stride team implemented the suggestion over the [PR](#). It was merged onto the main.

## Liveness Issue in Liquid Staking Arising from the Inability to Unpack IBC Acknowledgment Messages in Stride

<b>Title</b>	Liveness Issue in Liquid Staking Arising from the Inability to Unpack IBC Acknowledgment Messages in Stride
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	PROTOCOL
<b>Severity</b>	04 UNKNOWN
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	04 UNKNOWN
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Involved artifacts

- [x/staketia/keeper/ibc.go](https://x/staketia/keeper/ibc.go)

### Description

Informal team noticed an interesting place where the system could have liquid staking liveness issues, even though it was clear to both Stride and Informal teams, this potential issue was discussed in order to find a graceful solution.

IBC transfer callbacks - in `OnAcknowledgementPacket` , `UnpackAcknowledgementResponse` could error due to some not very likely system issues.

It was concluded that the only root cause would be if the marshaling changed on the Celestia side and it is not possible to unmarshal the acknowledgment on the Stride chain.

All the cryptographic proofs and version upgrades would be reflecting to the channel, and would not pose an issue in this case.

### Problem Scenarios

After [this](#) line of code, errors may cause a delegating record in the `TRANSFER_IN_PROGRESS` state to become stuck.

- Considering that there is an invariant allowing only one delegating record in `TRANSFER_IN_PROGRESS` status, having no delegations could occur.
- Moreover, even if this invariant is removed, it may not resolve the issue, as the failure would persist deterministically until manual intervention addresses the situation.

```

// OnAcknowledgementPacket success: Update the DelegationRecord's status to
DELEGATION_QUEUE
// OnAcknowledgementPacket failure: Delete the DelegationRecord
func (k Keeper) OnAcknowledgementPacket(ctx sdk.Context, packet channeltypes.Packet,
acknowledgement []byte) error {
    recordId, recordIdFound := k.GetTransferInProgressRecordId(ctx,
packet.SourceChannel, packet.Sequence)
    if !recordIdFound {
        return nil
    }

    // Parse whether the ack was successful or not
    isICATx := false
    ackResponse, err := icacallbacks.UnpackAcknowledgementResponse(ctx,
k.Logger(ctx), acknowledgement, isICATx)
    if err != nil {
        return err
    }

    // Grab the delegation record
    record, found := k.GetDelegationRecord(ctx, recordId)
    if !found {
        return errorsmod.Wrapf(err, "record not found for record id %d", recordId)
    }

    // If the ack was successful, update the record id to DELEGATION_QUEUE
    if ackResponse.Status == icacallbacktypes.AckResponseStatus_SUCCESS {
        record.Status = types.DELEGATION_QUEUE
        k.SetDelegationRecord(ctx, record)
    } else {
        // Otherwise there must be an error, so archive the record
        err := k.ArchiveFailedTransferRecord(ctx, recordId)
        if err != nil {
            return err
        }
    }

    // Clean up the callback store
    k.RemoveTransferInProgressRecordId(ctx, packet.SourceChannel, packet.Sequence)

    return nil
}

```

## Recommendation

There is no way to gracefully overcome this situation, so the Informal Systems team is simply highlighting this highly unlikely possibility.

## Implement Redemption Rate Change Validation in LiquidStakeAndDistributeFees

<b>Title</b>	Implement Redemption Rate Change Validation in LiquidStakeAndDistributeFees
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	0 INFORMATIONAL
<b>Impact</b>	0 NONE
<b>Exploitability</b>	0 NONE
<b>Status</b>	DISPUTED
<b>Issue</b>	

### Involved artifacts

- [x/staketia/keeper/delegation.go](https://github.com/celestiaorg/celestia-node/blob/main/x/staketia/keeper/delegation.go)

### Description

The `LiquidStakeAndDistributeFees` function distributes fees during the `mintEpoch` and is currently configured to run daily. We examined the potential consequences of running this function more frequently.

The audit team determined that there might be a risk of hitting redemption rate bounds if the mint epoch is shorter than the daily epoch, as the redemption rate bounds are checked at the beginning of each daily epoch.

### Problem Scenarios

The auditing team suggests implementing a check to verify that the redemption rate does not exceed the defined bounds - something similar to the existing `check` for the unbonding process.

```
// check ratio against bounds
if ratio.LT(hostZoneAfter.MinRedemptionRate) ||
ratio.GT(hostZoneAfter.MaxRedemptionRate) {
    return types.ErrRedemptionRateOutsideSafetyBounds
}
```

## Recommendation

Add a similar check as for:

- [VerifyImpliedRedemptionRateFromUnbonding](#).

## Resolution

The Stride team shared they do not feel this check is needed.

## Monitoring Suggestions

<b>Title</b>	Monitoring Suggestions
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	PROTOCOL
<b>Severity</b>	04 UNKNOWN
<b>Impact</b>	3 HIGH
<b>Exploitability</b>	04 UNKNOWN
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Involved artifacts

- [stTIA Multisig Tech Spec](#)

### Monitoring Recommendations:

The following suggestions emerged from the Informal systems tech spec analysis and code review. Given that most failure cases in the liquid stake TIA process result from the operator process not functioning as expected, and since the process itself addresses the consequences, off-chain monitoring must be vigilant to detect such situations:

- One potential issue is that rewards may affect higher redemption rate changes, potentially causing the zone to halt.  
This concern was highlighted in Informal System's [Assumptions in Operator Processes](#) finding, and the Stride team has concurred on closely monitoring this possibility off-chain before the operator process triggers the rewards IBC transfer back to the Stride chain.
- Rewards sent back to the Stride chain are not recorded on the chain, and there is no information about fees and reinvestment amounts stored.  
In discussions with the Stride team, it was decided that on-chain records are unnecessary as they won't be utilized. However, the development team confirmed the implementation of off-chain monitoring.
- Closely monitor the activity of operator when it comes to sending undelegation IBC transactions to the Celestia chain, to avoid situations where there are more than two unbonding queue records on Stride chain. This concern was also highlited in Informal System's [Assumptions in Operator Processes](#) finding. The Stride team confirmed the justification for monitoring these transactions:

*If we do notice that we are behind by more than 2 records, we can batch them up off chain and submit one undelegation tx, but then multiple confirm txs. Given how unlikely it is that we fall that far behind, I think this approach is desirable to making accommodations on chain*

## Operator's Ability to Insert Records in Active Store for Future Epochs

<b>Title</b>	Operator's Ability to Insert Records in Active Store for Future Epochs
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	04 UNKNOWN
<b>Impact</b>	2 MEDIUM
<b>Exploitability</b>	04 UNKNOWN
<b>Status</b>	DISPUTED
<b>Issue</b>	

### Involved artifacts

- [x/staketia/keeper/msg\\_server.go](#)

### Description

With the current implementation of overwrite functions, there's a chance that the overwrite messages from the SAFE MS account might create a new record with the future epoch as its ID.

We couldn't identify any scenarios where this would be necessary, as the record ID is a system-defined value and should not be generated on Stride with an incorrect epoch ID (unless there's an issue with epoch numbers).

### Problem Scenarios

We could envision two potential situations:

#### 1. Inserting Delegation Record:

- Inserting a delegation record with a status other than `TRANSFER_IN_PROGRESS` and an ID from a future epoch. This might temporarily pass the invariant check but would later encounter an issue due to the record already existing.  
While we understand this action is likely only performed by the trusted SAFE MS account and is therefore improbable, we also believe it should be safeguarded against as it is unnecessary.

#### 2. Inserting Redemption Record:

- In an unlikely scenario, if a redemption record with "future" ID is accidentally inserted, and assuming there's an updated (or old) value and redeemer in the record, when the future epoch arrives, an Unbonding Record with the same ID will be generated. Once the Unbonding Record reaches the "CLAIMABLE" status, the specified "redeemer" in the Redemption record could potentially receive some native tokens, even without actually redeeming. This would be at the expense of the protocol.

## Recommendation

Informal Systems recommends implementing a check wherein the record ID should be less than or equal to the current epoch ID when placing the overwrite message.

This precaution ensures that only records accidentally removed from the active store can be inserted or updated. Such operations are intended solely to rectify faulty record states, and the proposed check serves as a preventive measure against potential system issues in the event of an operator process error.

## Resolution

The Stride team decided that this check should not be added in favor of having maximum flexibility.



## Authorization Grants Setup for Mainnet Accounts

<b>Title</b>	Authorization Grants Setup for Mainnet Accounts
<b>Project</b>	Liquid Staking Celestia Intregation
<b>Type</b>	IMPLEMENTATION
<b>Severity</b>	1 LOW
<b>Impact</b>	1 LOW
<b>Exploitability</b>	1 LOW
<b>Status</b>	ACKNOWLEDGED
<b>Issue</b>	

### Involved artifacts

- [signed\\_safe\\_0\\_celestia.json](#)
- [signed\\_safe\\_1\\_celestia.json](#)
- [mainnet addresses](#)

- ```
# Celestia
celestia-appd tx broadcast signed_safe_0_celestia.json =>
484C2698CAA38677FC7622ED40A9D80317CBB2E8FD6BA24C14353C7880EA9787
celestia-appd tx broadcast signed_safe_1_celestia.json =>
A13F02FDBAA5410C3396ABD7129ECD31C512ED3F6E66E0022574F037E78C3E85

# Stride
strided tx broadcast signed_safe_0_stride.json =>
BD69910F95B44DA4AD57A56C9566C8576B8327684BCF041E9D27F88C6D64C0BA
strided tx broadcast signed_safe_1_stride.json =>
44BEFEE8E53C3A96D82897FC3AF6CF9B54AB70C690BB08B6C7E7193482BC70E3
strided tx broadcast signed_safe_2_stride.json =>
AFF1573599C2E2863B288536D76211118A2C9977265FD02DB6CC161C6D864D14
strided tx broadcast signed_safe_3_stride.json =>
```

### Description

After checking `signed_safe_0_celestia` txs (tx HASH:

[484C2698CAA38677FC7622ED40A9D80317CBB2E8FD6BA24C14353C7880EA9787](#)):

- The Informal team didn't find the grant for `MsgCancelUnbondingDelegation` in the spec. However, we deduced that this pertains to `MsgGrantAllowance` to the Celestia operator.

- Allowed messages in this grant were not ok:

```

"@type": "/cosmos.feegrant.v1beta1.MsgGrantAllowance",
"granter": "celestia1d6ntc7s8gs86tpdyn422vsqc6uaz9cejnxz5p5",
"grantee": "celestia1ghhu67ttgmrsyxljfl2tysyayswklvxzls400",
"allowance": {
  "@type": "/cosmos.feegrant.v1beta1.AllowedMsgAllowance",
  "allowance": {
    "@type": "/cosmos.feegrant.v1beta1.BasicAllowance",
    "spend_limit": [],
    "expiration": null
  },
  "allowed_messages": [
    "/cosmos.staking.v1beta1.MsgDelegate",
    "/cosmos.staking.v1beta1.MsgDelegate"
  ]
}

```

The Auditing team supposes that `MsgUndelegate` and `MsgCancelUnbondingDelegation` are missing amongst the `allowed_messages`.

## Problem Scenarios

This matter is relatively inconsequential, except in cases where the operator account lacks sufficient funds to execute the transaction broadcast.

## Recommendation

Address grants with an additional transaction on the mainnet - add the `MsgUndelegate` and `MsgCancelUnbondingDelegation` as allowed messages.

Additionally, document all grants held by MS accounts in the technical specification.

## Resolution

The Stride team has acknowledged this issue, which remains unresolved at present. It is a minor matter primarily concerning fees, and the associated authorization grants are unnecessary. The team plans to address this post-launch at some point in the future.

## Static Analysis Results by Gosec Tool for x/staketia Module

|                       |                                                             |
|-----------------------|-------------------------------------------------------------|
| <b>Title</b>          | Static Analysis Results by Gosec Tool for x/staketia Module |
| <b>Project</b>        | Liquid Staking Celestia Intregation                         |
| <b>Type</b>           | IMPLEMENTATION                                              |
| <b>Severity</b>       | 0 INFORMATIONAL                                             |
| <b>Impact</b>         | 0 NONE                                                      |
| <b>Exploitability</b> | 0 NONE                                                      |
| <b>Status</b>         | ACKNOWLEDGED                                                |
| <b>Issue</b>          |                                                             |

### Involved artifacts

- gosec analysis shared results [here](#)

### Description

The Informal team has reviewed the outcomes concerning the `x/staketia` module.

The remaining issues will be addressed in a subsequent partnership audit phase due to time constraints.

Regarding:

- potential overflow issues in `x/staketia` : all instances in the code involve operations with block times set far in the future.  
Nevertheless, we recommend implementing safe conversions and, in cases of overflow, considering panic or defer mechanisms.

### Recommendation

As shared above.

## Halting Claim Distribution for Unbonded Users in the Absence of Adequate Funds

|                       |                                                                                |
|-----------------------|--------------------------------------------------------------------------------|
| <b>Title</b>          | Halting Claim Distribution for Unbonded Users in the Absence of Adequate Funds |
| <b>Project</b>        | Liquid Staking Celestia Intregation                                            |
| <b>Type</b>           | PROTOCOL                                                                       |
| <b>Severity</b>       | 04 UNKNOWN                                                                     |
| <b>Impact</b>         | 2 MEDIUM                                                                       |
| <b>Exploitability</b> | 04 UNKNOWN                                                                     |
| <b>Status</b>         | DISPUTED                                                                       |
| <b>Issue</b>          |                                                                                |

### Involved artifacts

- [x/staketia/keeper/unbonding.go](https://x/staketia/keeper/unbonding.go)

### Description

Informal auditing team has considered the following scenario:

1. The operator initiates an IBC transfer of unbonded tokens from Celestia to the Claim MS on Stride.
2. Due to unforeseen circumstances such as slashing or operator errors, the amount of funds sent to the Claim MS is lower than initially anticipated.
3. Owing to operator delays, multiple unbonding records are pending confirmation, transitioning from "UNBONDED" to "CLAIMABLE" status.
4. It is assumed that the operator will execute the `ConfirmUnbondedTokenSweep` function for each unbonded record individually within a short timeframe.
5. Because each unbonding record individually having an amount less than the claim account balance, all records successfully reach the "CLAIMABLE" status.
6. For instance, if there are 100 tokens in the claim account balance and the sum of all unbonding records is 110 tokens, the `ConfirmUnbondedTokenSweep` function processes them separately, allowing each to reach the "CLAIMABLE" status.
7. However, during the `DistributeClaims` process, it is realized that the total sum of claimable unbonding records (110 tokens) exceeds the actual balance (100 tokens) available in the Claim MS.
8. As a result, the `DistributeClaims` process fails. Users, even those with valid claims, may experience delays as there are insufficient funds to cover the total amount of tokens for all users.

The Stride team has conducted an internal review of the protocol design and deliberated on the following scenario:

1. **Atomicity of Distribution:**

- (a) Whether the entire distribution across all unbonding records should be atomic, or
- (b) Whether the distribution for an individual unbonding record should be atomic.

2. **Exclusion of Non-Atomic Operations:**

- The team ruled out option (c) of having no atomicity, as this was deemed unfair to users. It could result in some users receiving their funds while others do not, even if they redeemed at the same time. Such a situation would indicate a potential bug in the system.

3. **Resolution for Overlapping Claims:**

- When considering the scenario of having two unbonding records (A and B), each with 100 tokens, but only a total of 110 tokens in the account, the team debated whether to distribute to record A (and not B) since there are sufficient funds to cover it and it was processed first.

4. **Decision on System Integrity:**

- The team concluded that in such a scenario, there might be an issue with the system. Instead of proceeding with partial operations, they opted to prioritize rectifying the underlying problem. Acknowledging and addressing the issue first was considered more prudent than continuing with partial distributions.

5. **Addressing Potential Causes:**

- Furthermore, the team noted that such a scenario could likely manifest when insufficient funds are sent to the claim account. They emphasized that this issue is relatively straightforward to fix, assuming it is the sole concern.

In summary, the Stride team's decision prioritizes the integrity of the system and user fairness. They recognized potential scenarios indicating system issues and preferred addressing them promptly over proceeding with partial operations that might compromise user experience.

## Exclusion of DELEGATED\_COMPLETED Records from Active Store

|                       |                                                                                                                             |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Title</b>          | Exclusion of DELEGATED_COMPLETED Records from Active Store                                                                  |
| <b>Project</b>        | Liquid Staking Celestia Intregation                                                                                         |
| <b>Type</b>           | PROTOCOL IMPLEMENTATION                                                                                                     |
| <b>Severity</b>       | 0 INFORMATIONAL                                                                                                             |
| <b>Impact</b>         | 0 NONE                                                                                                                      |
| <b>Exploitability</b> | 0 NONE                                                                                                                      |
| <b>Status</b>         | RESOLVED                                                                                                                    |
| <b>Issue</b>          | <a href="https://github.com/Stride-Labs/stride-staketia/pull/69">https://github.com/Stride-Labs/stride-staketia/pull/69</a> |

### Involved artifacts

- [x/staketia/keeper/redemption\\_rate.go](#)

### Description

Since delegated records in active store should not be able to have DELEGATED\_COMPLETED status, these checks [here](#) are slightly confusing.

Active store could end up with such records due to operator process activities, after changes implemented with [Audit batch 1](#).

The following recommendation were shared with the team, regarding the present solution:

1. It seems that it is reasonable to filter out the DELEGATED\_COMPLETED records on Get function. This seems like the easiest solution, considering time bounds and the fact that existing design was well tested by the developemtn team.
2. The other approach would be to make it impossible save DELEGATED\_COMPLETED records in active store and to add one more overwrite function "MoveToArchive" for moving the record from the active store and placing it to the archived, with DELEGATED\_COMPLETED.

### Problem Scenarios

The suggestions were provided solely to enhance clarity in the code implementation.

### Recommendation

As described above.

## Resolution

The Stride team has implemented the first suggested improvement with [Audit batch 2](#).





## Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of [Common Vulnerability Scoring System \(CVSS\) v3.1](#), which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the [Impact score](#), and the [Exploitability score](#). The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ [CVSS Qualitative Severity Rating Scale](#), and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

### Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

| Impact Score                                                                                      | Examples                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <b>High</b>   | Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic                                                                                                                                            |
|  <b>Medium</b> | Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x) |
|  <b>Low</b>    | Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)                                                               |
|  <b>None</b>   | Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation                                                                                                                                                           |





### Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be



- *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
- *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)


| Exploitability Score                                                                            | Examples                                                                                                                              |
|-------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
|  <b>High</b>   | illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors |
|  <b>Medium</b> | illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors |
|  <b>Low</b>    | illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors               |
|  <b>None</b>  | illegitimate actions taken in a coordinated fashion by all actors                                                                     |





## Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

| Severity Score                                                                                      | Examples                                                             |
|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
|  <b>Critical</b> | Halting of chain via a submission of a specially crafted transaction |

| Severity Score                                                                                         | Examples                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <b>High</b>          | Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers                                           |
|  <b>Medium</b>        | Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users |
|  <b>Low</b>           | 2x increase in node computational requirements via coordinated withdrawal of all user tokens                                                                                                            |
|  <b>Informational</b> | Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary         |