Tellor Cosmos Chain Review and Security Analysis

Client: Tellor

Date: 20 Feburary, 2025

Version: 1.0

Table of Contents

Table of Contents **2**
Security Review Information 9
Executive summary 9
Introduction **10**
Exclusions from This Review 10
Protocol Overview **11**
Key Features 11
Risks 12
Security Insights 12
Methodology **14**
Issue Severity Classification 14
Issue Status Definitions 14
Findings Summary **15**
Detailed Findings **19**
TS-C1 - Incorrect Handling of CommissionRate Range in DivvyingTips Function Enables Funds Theft 19
Classification 19
Description 19
Recommendation 19
TS-C2 - FeefromReporterStake Function Incorrectly Calculates The feeTracker's TokenOriginInfo.Amount 20
Classification 20
Description 20
Recommendation 20
TS-C3 - Lack of Validation in ProposeDispute Function Leads to Censorship and Possible Denial of Service 21
Classification 21
Description 21
Recommendation 21
TS-C4 - PrepareProposalHandler Is Susceptible to Replay Attacks 23
Classification 23
Description 23
Recommendation 23
TS-C5 - Inflated Voting Power by Transferring Tokens Between Accounts 24
Classification 24
Description 24
Recommendation 24
TS-H1 - Vote Extension Validation Failure Does Not Halt Block Proposal in ABCI++ PrepareProposalHandler 25
Classification 25

Description..... 25
Recommendation..... 25
TS-H2 - Inefficient Implementation of WeightedMode Function May Lead to a Chain Halt 26
Classification..... 26
Description..... 26
Recommendation..... 26
TS-H3 - Malicious Validator Can Force Vote Extensions to be Discarded..... 27
Classification..... 27
Description..... 27
Recommendation..... 27
TS-H4 - Multiple Issues In Reporter Module's AnteHandle Function..... 28
Classification..... 28
Description..... 28
Recommendation..... 28
TS-H5 - Missing Reporter Removal From Store If jailDuration is Max..... 30
Classification..... 30
Description..... 30
Recommendation..... 30
TS-H6 - Malicious Evidence Can Be Added to Prevent Deposit Claim..... 31
Classification..... 31
Description..... 31
Recommendation..... 31
TS-H7 - Missing Validation for Reporter Self-Removal as Selector..... 32
Classification..... 32
Description..... 32
Recommendation..... 32
TS-H8 - Incorrect Reporter Selection in SetVoterReporterStake Due to Current Block Query..... 33
Classification..... 33
Description..... 33
Recommendation..... 33
TS-H9 - Inaccurate Voting Power Calculation Due to Early BlockInfo Snapshot..... 34
Classification..... 34
Description..... 34
Recommendation..... 34
TS-H10 - Double Voting Power Inflation via MsgUpdateTeam..... 35
Classification..... 35
Description..... 35
Recommendation..... 35
TS-H11 - Overwriting DisputeFeePayer Entries Causes Loss of Funds in Refunds..... 36
Classification..... 36
Description..... 36
Recommendation..... 36
TS-H12 - Incorrect Pool Allocation in ReturnSlashedTokens Function..... 37

Classification 37
Description 37
Recommendation 37
TS-H13 - AddDisputeRound does not record DisputeFeePayer for payer 38
Classification 38
Description 38
Recommendation 38
TS-H14 - Invalid Minimum Bonded Tokens Check 39
Classification 39
Description 39
Recommendation 39
TS-H15 - Bypassing Stake Change Validation Using MsgExec from the authz Module 40
Classification 40
Description 40
Recommendation 40
TS-H16 - Incomplete and Faulty Dispute Rounds Logic Leading to Multiple Issues 41
Classification 41
Description 41
Recommendation 42
TS-M1 - Possible Out-of-Gas Errors When Claiming Deposits 43
Classification 43
Description 43
Recommendation 43
TS-M2 - QueryDecodeValueRequest Is Susceptible to Denial-of-Service Attack 44
Classification 44
Description 44
Recommendation 44
TS-M3 - FeefromReporterStake Function Is Biased Towards Penalizing Validators With Low Address Byte-string Representation 45
Classification 45
Description 45
Recommendation 45
TS-M4 - UpdateDispute Function Can Lead to Chain Halt 46
Classification 46
Description 46
Recommendation 46
TS-M5 - FeeRound Function Is Susceptible To Out-Of-Gas Errors 47
Classification 47
Description 47
Recommendation 47
TS-M6 - CheckOpenDisputesForExpiration Function Can Lead to Chain Halt 49
Classification 49
Description 49
Recommendation 49

TS-M7 - SetAggregatedReport Function Can Cause Chain Halt....50
Classification....50
Description....50
Recommendation....50
TS-M8 - Lack of Minimum Tip Requirement Leads to Disproportionally Low Storage Costs. 51
Classification....51
Description....51
Recommendation....51
TS-M9 - Proposer Can Provide Arbitrary Number of Votes Due To Missing Check....52
Classification....52
Description....52
Recommendation....52
TS-M10 - Bonded Tokens Snapshot Does Not Take Into Account Slashing Events....53
Classification....53
Description....53
Recommendation....53
TS-M11 - The Reporter Module Does Not Implement Bonding-related Hooks....54
Classification....54
Description....54
Recommendation....54
TS-M12 - MsgWithdrawTip Bypasses 5% Stake Change Restriction....55
Classification....55
Description....55
Recommendation....55
TS-M13 - Potentially Expensive FeefromReporterStake execution....56
Classification....56
Description....56
Recommendation....56
TS-M14 - Possible Out-of-Gas Error in MsgVote Due to Retrieving All Voters....58
Classification....58
Description....58
Recommendation....58
TS-M15 - Inaccurate Voting Power Due to Dynamic Total Supply in the Current Block....59
Classification....59
Description....59
Recommendation....59
TS-M16 - Vulnerable cometbft package....60
Classification....60
Description....60
Remediation....60
Resources....60
TS-L1 - Incomplete ValidateBasic Function Implementation for MsgSubmitValue....61
Classification....61

Description..... 61
Recommendation..... 61
TS-L2 - Missing Sanitization And Validation In UpdateDataSpec..... 62
Classification..... 62
Description..... 62
Recommendation..... 62
TS-L3 - Incomplete ValidateBasic Implementation for MsgCreateReporter..... 63
Classification..... 63
Description..... 63
Recommendation..... 63
TS-L4 - Chain Halt Due To Empty Cyclelist..... 64
Classification..... 64
Description..... 64
Recommendation..... 64
TS-L5 - Missing Cyclelist Validation in UpdateCyclelist Function..... 65
Classification..... 65
Description..... 65
Recommendation..... 65
TS-L6 - Block Stuffing Due To Unlimited queryData in PreventBridgeWithdrawalReport function..... 66
Classification..... 66
Description..... 66
Recommendation..... 66
TS-L7 - WithdrawTip Function Bypasses the Staking Module..... 67
Classification..... 67
Description..... 67
Recommendation..... 67
TS-L8 - Permissionless RegisterSpec Function Leads to Malicious Behaviour..... 68
Classification..... 68
Description..... 68
Recommendation..... 68
TS-L9 - Possible Invalid Handling of Voting Power..... 69
Classification..... 69
Description..... 69
Recommendation..... 69
TS-L10 - Incomplete Export Genesis..... 70
Classification..... 70
Description..... 70
Recommendation..... 70
TS-L11 - Potentially Incorrect Implementation of MsgRemoveSelector handler..... 72
Classification..... 72
Description..... 72
Recommendation..... 72
TS-L12 - Incomplete Validation of the MsgProposeDispute..... 73

Classification....73
Description....73
Recommendation....73
TS-L13 - Incomplete Implementation of ValidateBasic of MsgRegisterSpec....74
Classification....74
Description....74
Recommendation....74
TS-L14 - Missing Validation on Params in Registry Module....75
Classification....75
Description....75
Recommendation....75
TS-L15 - Incomplete Validation for GenesisState in Registry Module....76
Classification....76
Description....76
Recommendation....76
TS-L16 - Incomplete validation for MsgTip in Oracle module....77
Classification....77
Description....77
Recommendation....77
TS-L17 - UnjailReporter Function Fails to Reset JailedUntil Field....78
Classification....78
Description....78
Recommendation....78
TS-L18 - Reporter Should Not Be Allowed to Pay From Bond in Self-Created Disputes....79
Classification....79
Description....79
Recommendation....79
TS-L19 - Missing Cyclelist validation in GenesisState of the Oracle module....80
Classification....80
Description....80
Recommendation....80
TS-L20 - Missing validation for Params in Oracle module....81
Classification....81
Description....81
Recommendation....81
TS-L21 - params.MinCommissionRate Not Enforced in CreateReporter Function....82
Classification....82
Description....82
Recommendation....82
TS-L22 - Chain Halt Risk When Minted Coins Are Not layer.BondDenom....83
Classification....83
Description....83
Recommendation....83
TS-L23 - CalculateBlockProvision Mints DefaultBondDenom Instead of Minter.BondDenom

84
Classification....84
Description....84
Recommendation....84
TS-L24 - Vulnerable cosmossdk.io/math package....85
Classification....85
Description....85
Remediaiton....85
Resources....85
TS-I1 - Miscellaneous Comments....86
Classification....86
Description....86
Recommendation....87
TS-I2 - Redundant Team Vote Tally Computation in Voting Logic....88
Classification....88
Description....88
Recommendation....88
Disclaimer....89

Security Review Information

Repository	tellor-layer
Commit	0da020d31681cd8c43545b44666fbec19c95dacd
Scope	app/* x/bridge/* x/dispute/* x/oracle/* x/mint/* x/registry/* x/reporter/*
Version	Final Report (v. 1.0)
Date	20th January 2025

Executive summary

As of Feburary 20, 2025, our comprehensive security review of the Tellor Chain has been concluded. Initially, the assessment identified **5** critical-severity, **16** high-severity, **16** Medium-severity, and **24** low-severity vulnerabilities. All critical issues were resolved, **15** high severity issues were resolved and **1** high severity issue was partly-resolved, **11** medium-severity issues were resolved, **4** acknowledged and **1** mitigated. As for low-severity issues: **23** were resolved and **1** acknowledged.

Introduction

Torii Security has been commissioned by Tellor to conduct a comprehensive security review of their Cosmos chain, focusing on the robustness, security, and efficiency of the chain implementation. The review aims to critically evaluate the program's architecture and codebase, with the following specific objectives:

- **Verify Protocol Integrity**: Assess the program's operation against its design specifications to ensure it functions correctly and efficiently within the Solana ecosystem. This includes evaluating its interaction with other protocols and services on the Solana network.
- **Identify Security Vulnerabilities**: Uncover potential security weaknesses that could be exploited by malicious actors, including but not limited to, flaws in program logic, transaction handling, and external dependencies.
- **Detect Program Bugs**: Identify bugs and glitches in the code that may result in unintended or erratic program behavior, potentially compromising its performance or security.
- **Provide Improvement Recommendations**: Offer actionable advice to enhance the program's security posture, efficiency, and code clarity, aiming to fortify it against current and future security threats while improving maintainability and scalability.

Exclusions from This Review

While the security review conducted by Torii Security on behalf of Tellor provides a comprehensive analysis of the Cosmos chain architecture, codebase, and security posture, certain aspects are beyond the scope of this audit. These exclusions are critical for stakeholders to understand, as they may require separate consideration and evaluation. The following areas have not been verified as part of this security review:

- **Operational Aspects:** Procedures, runbooks, and other operations-related workflows not covered by the on-chain code.
- **Daemons:** Off-chain components responsible for automated tasks and data handling.
- **APIs:** External interfaces and integrations not included in the on-chain security review.
- **Economic Design:** Game-theoretic and market-based mechanisms that were beyond the audit’s technical scope.
- **EVM Contracts:** External Ethereum-based smart contracts and side-chain contracts interacting with the Cosmos chain.

Protocol Overview

Tellor Layer is a standalone Layer 1 (L1) blockchain built using the Cosmos SDK. It aims to achieve decentralized consensus on subjective data by leveraging Tendermint (CometBFT) consensus and a network of staked participants (reporters and validators). These participants are economically incentivized to honestly report requested data—such as price feeds—by staking Tellor's native TRB token and risking slashing in case of dishonest reporting or malicious behavior.

A key component of Tellor Layer is a trustless bridge module that connects the new Cosmos-based L1 with Tellor's existing Ethereum protocol. This enables a robust two-way flow of data and tokens (TRB) between Ethereum and Tellor Layer, enhancing Tellor's overall ability to provide cross-chain data feeds and services.

Key Features

- **Dual Delegation for Reporting and Validating:** Tellor Layer adopts Tendermint's delegated proof-of-stake (dPoS) for validation, but also introduces a separate delegation mechanism specifically for reporting. Each TRB token can be simultaneously delegated toward both validation (securing the chain) and reporting (submitting data). This approach decouples the roles of validators (capped set due to Tendermint constraints) and reporters (uncapped, to maximize data provider decentralization).
- **Optimistic and Robust Data Handling**:
    - **Robust Data**: Achieved when at least two-thirds (⅔) of reporters and validators agree on a submitted value. It can then be used immediately, provided the user trusts the validator set.
    - **Edge Data**: Data that does not reach ⅔ agreement is handled optimistically (i.e., consumers may require a dispute window before final acceptance). This ensures data can still be submitted for less popular or more obscure requests, albeit with extra caution needed by consumers.
- **Flexible Dispute and Slashing Mechanism**: Any participant can raise a dispute against a reporter they believe submitted incorrect data. Depending on severity (warning, minor infraction, major infraction), disputes can result in partial or full slashing. Disputes also impose escalating fees to discourage spam. During an active dispute, both the disputing party's fee and the disputed reporter's staked amount are locked until the system reaches a resolution.
- **Trustless Bridging**:
    - **Token Bridge:** TRB tokens are seamlessly transferred between Ethereum and Tellor Layer via a light client bridge, supporting a two-way exchange of assets. Security is prioritized through extended challenge periods (13 hours to finalize a deposit and 28 days unbonding period before withdrawals).
    - **Data Bridge:** Values aggregated on Tellor Layer are signed by validators and can be relayed to other chains. External protocols can consume these validated data points by verifying signature proofs from the Tellor Layer validator set.

Risks

The security review of the Tellor chain has identified several risks. It is crucial to understand that the risks identified in this section of the report are associated with operational and procedural aspects of the Tellor chain that were not within the purview of our security review.

1. **Economic Incentive Risks:** The protocol's security hinges on the value of TRB and the willingness of participants to stake and dispute malicious data. If the token's value decreases or if insufficient staking participation exists, it may weaken the economic incentives to submit correct data or raise disputes.
2. **Bridge Finality and Delays:** Bridging tokens and data involves time delays (e.g., 13 hours to finalize deposits, 28 days unbonding). These windows protect against short-term attacks but also create user experience issues and potential vulnerabilities (e.g., if bridging transactions are not monitored, an attacker could attempt to exploit users unaware of finality constraints).
3. **Low Liquidity / Low Participation in Edge Queries**: Some queries that are not in the chain's "cycle list" or are less popular may receive limited reporter engagement. Fewer reporters can increase the risk of incorrect data slipping through if not properly disputed. Users consuming these "edge data" feeds must employ additional layers of security, such as dispute monitoring or fallback logic.
4. **Governance Attack:** Tellor Layer's governance structure (25% each to users, reporters, token holders, and team) could, in theory, be compromised if a single entity or colluding group gains majority control. Malicious governance proposals could result in unforeseen protocol parameter changes—such as inflation rates or dispute fee manipulations.
5. **Upgrades and Forks:** While upgrades (soft upgrades) follow a measured process, critical security issues may require a chain fork. Coordinating validator set updates and bridging logic under severe time pressure can be challenging. Improperly managed forks could lead to chain splits, confusion over canonical data, and security gaps in the bridging contracts.

Security Insights

Validation Architecture Issues

The current implementation shows inefficiencies in its validation approach. Instead of leveraging *ValidateBasic* functions for preliminary transaction screening, validation logic is primarily implemented within transaction processing functions. This architectural decision leads to unnecessary resource consumption, as transactions that could be rejected early are allowed to consume block space. The *ValidateBasic* functions should be utilized to perform initial validation checks before transactions enter the mempool, filtering out obviously invalid transactions and reducing unnecessary network load.

Testing Coverage Concerns

The codebase demonstrates insufficient test coverage, particularly in critical paths and edge cases. This limitation poses risks to the network's stability and security. A comprehensive test suite should be developed to include:

- Unit tests for individual components
- Integration tests for cross-module interactions
- Stress tests for performance bottlenecks
- Edge case scenarios that could potentially disrupt network operations

Performance Optimization Needs

A concern lies in the prevalent use of for loops in various operations. This implementation pattern can lead to significant performance degradation, particularly as the network scales. In worst-case scenarios, these loops could cause:

- Block processing delays
- Increased transaction latency
- Potential chain halts during high-load periods
- Memory consumption issues

We highly recommend reviewing the validation performed during transaction processing and moving as much of it into *ValidateBasic* functions. Additionally, performing an overall architectural review can be beneficial for reducing the number of required loops. If, the loops are deemed necessary, batching is highly encouraged to reduce the maximal number of iterations executed. Apart from that, we encourage allocating some development effort towards designing and implementing a thorough testing suite that would cover as much production code as possible - introducing fuzz tests is also recommended.

Methodology

Issue Severity Classification

This report differentiates identified issues into distinct severity levels, each reflecting the potential impact on the system's security and overall functionality.

Severity	Description
Critical	Issues that present an immediate and severe threat, such as significant financial loss, irreversible locking of funds, or catastrophic system failure. These vulnerabilities require urgent remediation.
High	Bugs or vulnerabilities that could disrupt the correct operation of the system, potentially leading to incorrect states or temporary denial of service. Prompt attention and corrective action are necessary.
Medium	Issues that indicate deviations from best practices or suboptimal use of system primitives. While they may not pose immediate security threats, these issues could lead to vulnerabilities or inefficiencies if unaddressed.
Low	Minor concerns that have anegligible impact on system security or functionality. These may include inefficiencies or minor deviations from best practices that are unlikely to affect the system's operation significantly.
Informational	Suggestions related to design decisions, potential enhancements, or optimizations that do not have adirect impact on security. Implementing these recommendations may improve aspects such as usability or code readability but is not essential for system security.

Issue Status Definitions

Each issue is assigned astatus reflecting its current resolution stage.

Status	Description
Pending	The issue has been identified but not yet reviewed or addressed by the development team.
Acknowledged	The development team has recognized the issue but has not completed its resolution.
Mitigated	The issue has been resolved through fixes or procedures implemented outside the codebase.
Resolved	The issue has been fully addressed, with implemented changes verified for effectiveness.

Findings Summary

ID	Title	Severity	Status
TS-C1	Incorrect Handling of CommissionRate Range in DivvyingTips Function Enables Funds Theft	Critical	Resolved
TS-C2	FeefromReporterStake Function Incorrectly Calculates The feeTracker's TokenOriginInfo.Amount	Critical	Resolved
TS-C3	Lack of Validation in ProposeDispute Function Leads to Censorship and Possible Denial of Service	Critical	Resolved
TS-C4	PrepareProposalHandler Is Susceptible to Replay Attacks	Critical	Resolved
TS-C5	Inflated Voting Power by Transferring Tokens Between Accounts	Critical	Resolved
TS-H1	Vote Extension Validation Failure Does Not Halt Block Proposal in ABCI++ PrepareProposalHandler	High	Resolved
TS-H2	Inefficient Implementation of WeightedMode Function May Lead to a Chain Halt	High	Resolved
TS-H3	Malicious Validator Can Force Vote Extensions to be Discarded	High	Resolved
TS-H4	Multiple Issues In Reporter Module's AnteHandle Function	High	Resolved
TS-H5	Missing Reporter Removal From Store If jailDuration is Max	High	Resolved
TS-H6	Malicious Evidence Can Be Added to Prevent Deposit Claim	High	Resolved
TS-H7	Missing Validation for Reporter Self-Removal as Selector	High	Resolved
TS-H8	Incorrect Reporter Selection in SetVoterReporterStake Due to Current Block Query	High	Resolved
TS-H9	Inaccurate Voting Power Calculation Due to Early BlockInfo Snapshot	High	Resolved
TS-H10	Double Voting Power Inflation via MsgUpdateTeam	High	Resolved

ID	Title	Severity	Status
TS-H11	Overwriting DisputeFeePayer Entries Causes Loss of Funds in Refunds	High	Resolved
TS-H12	Incorrect Pool Allocation in ReturnSlashedTokens Function	High	Resolved
TS-H13	AddDisputeRound does not record DisputeFeePayer for payer	High	Resolved
TS-H14	Invalid Minimum Bonded Tokens Check	High	Resolved
TS-H15	Bypassing Stake Change Validation Using MsgExec from the authz Module	High	Resolved
TS-H16	Incomplete and Faulty Dispute Rounds Logic Leading to Multiple Issues	High	Partially Resolved
TS-M1	Possible Out-of-Gas Errors When Claiming Deposits	Medium	Resolved
TS-M2	QueryDecodeValueRequest Is Susceptible to Denial-of-Service Attack	Medium	Resolved
TS-M3	FeefromReporterStake Function Is Biased Towards Penalizing Validators With Low Address Byte-string Representation	Medium	Acknowledged
TS-M4	UpdateDispute Function Can Lead to Chain Halt	Medium	Resolved
TS-M5	FeeRound Function Is Susceptible To Out-Of-Gas Errors	Medium	Mitigated
TS-M6	CheckOpenDisputesForExpiration Function Can Lead to Chain Halt	Medium	Resolved
TS-M7	SetAggregatedReport Function Can Cause Chain Halt	Medium	Resolved
TS-M8	Lack of Minimum Tip Requirement Leads to Disproportionally Low Storage Costs	Medium	Resolved
TS-M9	Proposer Can Provide Arbitrary Number of Votes Due To Missing Check	Medium	Resolved
TS-M10	Bonded Tokens Snapshot Does Not Take Into Account Slashing Events	Medium	Acknowledged
TS-M11	The Reporter Module Does Not Implement Bonding-related Hooks	Medium	Acknowledged
TS-M12	MsgWithdrawTip Bypasses 5% Stake Change Restriction	Medium	Resolved

ID	Title	Severity	Status
TS-M13	Potentially Expensive FeefromReporterStake execution	Medium	Acknowledged
TS-M14	Possible Out-of-Gas Error in MsgVote Due to Retrieving All Voters	Medium	Resolved
TS-M15	Inaccurate Voting Power Due to Dynamic Total Supply in the Current Block	Medium	Resolved
TS-M16	Vulnerable cometbft package	Medium	Resolved
TS-L1	Incomplete ValidateBasic Function Implementation for MsgSubmitValue	Low	Resolved
TS-L2	Missing Sanitization And Validation In UpdateDataSpec	Low	Resolved
TS-L3	Incomplete ValidateBasic Implementation for MsgCreateReporter	Low	Resolved
TS-L4	Chain Halt Due To Empty Cyclelist	Low	Resolved
TS-L5	Missing Cyclelist Validation in UpdateCyclelist Function	Low	Resolved
TS-L6	Block Stuffing Due To Unlimited queryData in PreventBridgeWithdrawalReport function	Low	Resolved
TS-L7	WithdrawTip Function Bypasses the Staking Module	Low	Resolved
TS-L8	Permissionless RegisterSpec Function Leads to Malicious Behaviour	Low	Acknowledged
TS-L9	Possible Invalid Handling of Voting Power	Low	Resolved
TS-L10	Incomplete Export Genesis	Low	Resolved
TS-L11	Potentially Incorrect Implementation of MsgRemoveSelector handler	Low	Resolved
TS-L12	Incomplete Validation of the MsgProposeDispute	Low	Resolved
TS-L13	Incomplete Implementation of ValidateBasic of MsgRegisterSpec	Low	Resolved
TS-L14	Missing Validation on Params in Registry Module	Low	Resolved
TS-L15	Incomplete Validation for GenesisState in Registry Module	Low	Resolved

ID	Title	Severity	Status
TS-L16	Incomplete validation for MsgTip in Oracle module	Low	Resolved
TS-L17	UnjailReporter Function Fails to Reset JailedUntil Field	Low	Resolved
TS-L18	Reporter Should Not Be Allowed to Pay From Bond in Self-Created Disputes	Low	Resolved
TS-L19	Missing Cyclelist validation in GenesisState of the Oracle module	Low	Resolved
TS-L20	Missing validation for Params in Oracle module	Low	Resolved
TS-L21	params.MinCommissionRate Not Enforced in CreateReporter Function	Low	Resolved
TS-L22	Chain Halt Risk When Minted Coins Are Not layer.BondDenom	Low	Resolved
TS-L23	CalculateBlockProvision Mints DefaultBondDenom Instead of Minter.BondDenom	Low	Resolved
TS-L24	Vulnerable cosmossdk.io/math package	Low	Resolved
TS-I1	Miscellaneous Comments	Informational	Partially Resolved
TS-I2	Redundant Team Vote Tally Computation in Voting Logic	Informational	Resolved

Detailed Findings

TS-C1 - Incorrect Handling of CommissionRate Range in DivvyingTips Function Enables Funds Theft

Classification

Severity: **Critical**

Status: **Resolved**

Description

The *DivvyingTips* function tips that a reporter has earned from reporting in the oracle module amongst the reporters' selectors. The purpose of the height argument is only to pay out selectors who were part of the reporter at the height of the report. *DivvyingTips* distributes the reward among the reporter and its selectors based on their shares at the given height. When doing so, it computes the reporter commission by multiplying the reward by the *CommissionRate*. This would suggest that the commission rate is a number in the [0,1] range. However, in *CreateSelector* function defined here, the *CommissionRate* is required to be less-than-or-equal-to 100. Consequently, if the reporter sets it to 99, it would cause the commission to be 99 times the reward. This inflated value would ultimately influence the *SelectorTips*, which ultimately leads to a funds theft from the *TipsEscrowPool* here.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/distribution.go#L34-L35

Recommendation

Change the implementation so that the *CommissionRate* is treated as a percentage value. An exemplary code, would be:

```
//selector's commission = reporter's commission rate * reward
commission :=
reward.Mul(reporter.CommissionRate).Quo(math.LegacyNewDec(100))

// Calculate net reward
netReward := reward.Sub(commission)
```

Remediation

The vulnerability was fixed in the *09a0a0db24cc00d2c183a95f5d4a14bf90217b3a* commit.

TS-C2 - FeefromReporterStake Function Incorrectly Calculates The feeTracker's TokenOriginInfo.Amount

Classification

Severity: Critical

Status: Resolved

Description

The *FeefromReporterStake* allows a reporter to use part of its stake to pay the fee for a dispute. Its selectors and delegations are iterated to compute the actual stake, and then the fee is equally distributed between all selectors by performing partial unbondings. For every unbonding, the *feeTracker* is updated with a *TokenOriginInfo* to keep track of the fees. However, in the *else* branch, this *TokenOriginInfo* is provided with incorrect data. In fact, the *Amount* is set to *unbondAmt*, but since this value has already been subtracted in line *125*, this is not the actual amount paid with this delegation but the leftover. Consequently, an invalid amount is recorded, and an incorrect amount would be rebonded when executing *FeeRefund*.

For example, consider the following scenario: *unbondAmt = 100* and there are *[10, 40, 50]* delegations. The created *feeTracker* would be equal to *[90, 50, 0]*, which will rebond 140 tokens instead of 100.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/withdraw.go#L123-L142

Recommendation

Change the implementation so that *TokenOriginInfo.Amount* is set to *stakeWithValidator* instead of *unbondAmt*.

Remediation

The vulnerability was fixed in the *3faa8d4323d1c7ac069d32efe9c7fd26493557c2* commit.

TS-C3 - Lack of Validation in ProposeDispute Function Leads to Censorship and Possible Denial of Service

Classification

Severity: **Critical**

Status: **Resolved**

Description

The *ProposeDispute* functionality allows a proposer to submit a *Report* that includes the reporter's power at a specific block height. This reported power is used to calculate the fee and determine slashing penalties. However, no validation ensures the reported power matches the actual on-chain data. As a result, a malicious proposer can exploit this lack of verification to manipulate the process.

An exemplary attack scenario:

1. A malicious actor submits a new dispute with the power field in the Report set to 1.
2. The dispute message is constructed with a Fee of 10,000 loya.
3. The *disputeFee* is then calculated as:

$$disputeFee = powerReduction \times power \times \frac{slashingPercentage}{100}$$

For a slashing percentage of 1%, this results in a dispute fee of exactly 10,000 loya.

4. The *JailReporter* function is invoked, which directly jails the reporter.

The impact of this issue is twofold:

- Censorship: Legitimate reporters can be unfairly jailed, disrupting their ability to participate.
- Denial of Service (DoS): Repeated abuse could destabilize the system by jailing reporters en masse.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/msg_server_propose_dispute.go#L15-L44

Recommendation

Implement checks that will ensure the provided report actually exists on-chain and validate data. Additionally, as the power could have changed, the reporter's actual power should also be checked.

Remediation

The vulnerability was fixed in the *11d549fbdb87055e8692c61f6c05447ec4675528* commit.

TS-C4 - PrepareProposalHandler Is Susceptible to Replay Attacks

Classification

Severity: Critical

Status: Resolved

Description

During the execution of the ABCI++ *PrepareProposal* handler, the *CheckInitialSignaturesFromLastCommit* function is executed. This function performs various checks and then calls the *EVMAddressFromSignatures*. This function ensures that both the signatures (A and B) provided in *voteExt.InitialSignature* are the signatures of 2 hardcoded messages, and then the pubkey is derived as well as the EVM address. However, the message to be signed is static. Consequently, the whole authentication is vulnerable to replay attacks. A replay attack occurs when a malicious actor intercepts a valid transaction and retransmits it to the network. For example, if the signatures, operator addresses, or attestations included in the proposal are not bound to a unique context, such as the block height, they could be reused in the next blocks by anyone.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/app/proposal_handler.go#L280-L283

Recommendation

Change the design so that the signatures are dependent on a unique context. For example, the message could also contain the block height.

Remediation

The vulnerability was fixed in the *856a415fe23fd3513f862a07c1c4ed31139f2501* commit.

TS-C5 - Inflated Voting Power by Transferring Tokens Between Accounts

Classification

Severity: **Critical**

Status: **Resolved**

Description

The voting power of token holders is determined by their token balance (GetAccountBalance) in the current block. This logic allows users to exploit the system by transferring tokens between accounts within the same block:

1. User A transfers all their tokens to User B.
2. User B votes with the consolidated balance.
3. User B transfers the tokens back to User A (or another account).
4. Repeat the process to inflate the total voting power and manipulate the dispute outcome.

Code Location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/vote.go#L184

Recommendation

The user's voting power should be based on their token balance as of the past block height, not the current block height.

Remediation

The vulnerability was fixed in the a2a64b4ee013eda4f02b6046c828110893ae5c09.

TS-H1 - Vote Extension Validation Failure Does Not Halt Block Proposal in ABCI++ PrepareProposalHandler

Classification

Severity: High

Status: Resolved

Description

During the execution of the ABCI++ *PrepareProposalHandler*, the proposer node runs the *ValidateVoteExtensions* function to verify the vote extensions provided in the previous block (h-1). This process includes validating vote extensions against the 2/3 threshold required by *cometBFT*. However, when errors such as insufficient votes or invalid validator signatures occur, the function does not halt the execution of the proposer. Instead, it logs the error and continues the execution. This behavior could result in incorrect blocks being proposed and the missing enforcement of the 2/3 vote requirement. The proposer should abort if it is providing an incorrect *LocalLastCommit*.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/app/proposal_handler.go#L65-L69

Recommendation

Change the implementation to return the error instead of just logging the event. An exemplary code would be:

```
err := baseapp.ValidateVoteExtensions(ctx, h.valStore, req.Height,
ctx.ChainID(), req.LocalLastCommit)
if err != nil {
    h.logger.Info("PrepareProposalHandler: failed to validate vote
extensions", "error", err, "votes", req.LocalLastCommit.Votes)
    return err // stop execution in case of error
}
```

Remediation

The vulnerability was fixed in the *a925af3f01bc75c0a21250308dda9b6dffe1312d* commit.

TS-H2 - Inefficient Implementation of WeightedMode Function May Lead to a Chain Halt

Classification

Severity: High

Status: Resolved

Description

The *WeightedMode* function is executed in the *EndBlocker* of the oracle module in order to aggregate the values submitted by reporters. It constructs a *frequencyMap* and then iterates all the reports to populate it. However, it is inefficiently implemented and could lead to DoS. Specifically, to populate the map, the execution iterates *n* time, incrementing the *frequencyMap* at the *r.Value* where *n* is the power of the reporter. Consequently, it would lead to unnecessary large iterations, which would increase the processing time of the *EndBlocker*, leading to chain slowdown or, in the worst case, to a chain halt.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/weighted_mode.go#L9-L31

Recommendation

Replace the loop with:

```
frequencyMap[r.Value]+=r.Power
```

Remediation

The vulnerability was fixed in the *8b845cae37274a4e1221614eb30b723fb189f7e9* commit.

TS-H3 - Malicious Validator Can Force Vote Extensions to be Discarded

Classification

Severity: High

Status: Resolved

Description

In the *PrepareProposalHandler*, if invalid data is included in the vote extensions, errors may occur in *CheckInitialSignaturesFromLastCommit* or *CheckValsetSignaturesFromLastCommit*. In such cases, all vote extensions are discarded, and an empty *VoteExtTx{}* is appended to the proposed transactions. This allows the proposer to censor other validators' vote extension data by deliberately injecting a vote with invalid data. As a result, the new block is constructed using only mempool transactions, excluding any transactions from the vote extensions. Critically, the malicious proposer does not face any penalties, as the proposal is still considered valid and accepted by other validators.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/app/proposal_handler.go#L65-L93

Recommendation

Change the implementation so that votes are always required on processing proposal.

Remediation

The vulnerability was fixed in the *567b188a1b9d854f77b48e512d7598ed6987bce7* commit.

TS-H4 - Multiple Issues In Reporter Module's AnteHandle Function

Classification

Severity: High

Status: Resolved

Description

Multiple issues were identified regarding the reporter module's *AnteHandle* function:

In the reporter module's *AnteHandle* function, staking messages are intercepted, and a check ensures that bonded token variations do not exceed a safe threshold of 5%. To enforce this, a snapshot is taken, and the amount specified in the staking message summed with the *TotalBondedTokens* is compared against the threshold. However, the *MsgBeginRedelegate* message can be exploited to bypass this restriction.

Specifically, the amount in *MsgBeginRedelegate* is always treated as a positive value, regardless of its effect on the bonded tokens. This results in incorrect computations when tokens are redelegated from a bonded validator to an unbonded one. If more than 5% of tokens are effectively unstaked, the *MsgUndelegate* would fail, but a malicious actor could instead redelegate to an unbonded validator, bypassing the threshold and avoiding the intended restriction. This exposes the system to risks associated with unregulated bonded token variations.

Additionally, other staking messages may not be handled correctly either:

• If executing *MsgCancelUnbondingDelegation* on an undelegation from an unbonding validator, the amount should be treated as zero.
• Similarly, delegating to or undelegating from an unbonded validator should not affect the bonded tokens.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/ante/ante.go#L30-L66

Recommendation

Implement a correct handling of mathematical operation. If the stake is moved from bonded to unbonded validator, the amount should be treated as decreasing (subtraction). If the stake is moved from unbonded to bonded validator, the amount should be treated as increasing (addition). However, the amount should not be modified if the redelegation happens from bonded to bonded validator or from unbonded to unbonded validator.

Remediation

The vulnerability was fixed in the *40345a5c9f9fcb93645ba3e7e2c13cda3789c4aa* commit.

TS-H5 - Missing Reporter Removal From Store If jailDuration is Max

Classification

Severity: High

Status: Resolved

Description

The *JailReporter* function states that if a major jail duration is provided, there is no need to actually jail the reporter. In that case, the reporter is supposed to be removed from store. However, we did not identify this removal taking place.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/dispute.go#L184-L187

Recommendation

Implement the reporter removal mechanism according to the design.

Remediation

The vulnerability was fixed in the *1aad2ba723c8ae06aba1d3304268ebf5e5ab6bf4* commit.

TS-H6 - Malicious Evidence Can Be Added to Prevent Deposit Claim

Classification

Severity: High

Status: Resolved

Description

The *MsgAddEvidence* message allows users to add *MicroReport* to an open dispute. The *FlagAggregateReport* function will be called to set the *Aggregate.Flagged* value to true (FlagAggregateReport).

The issue is that there is no validation that the provided *MicroReport* aligns with the dispute opened. For example, users can provide *MicroReport* that is different from the dispute, giving useless evidence.

This can be used to prevent users from claiming their deposits in ClaimDeposit by flagging the aggregates maliciously. If the aggregate is flagged, a *types.ErrAggregateFlagged* error will be returned. This causes a denial of service issue as the user should be able to claim their deposits as no disputes were opened for that particular deposit query ID.

Code location:

https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/msg_server_add_evidence.go#L15

Recommendation

Add validation to ensure the provided *MicroReport* corresponds to the dispute. This can be achieved by checking the first part of the dispute's *HashId*, which is made up from (*MicroReport*, *DisputeCategory*) in SetNewDispute. This will ensure that the dispute's evidence should only contain the relevant micro reports, and irrelevant evidence will be rejected to prevent flagging incorrect aggregates.

Remediation

The vulnerability was fixed in the 37088df75a86335b7428cd6239d468741004d5b6.

TS-H7 - Missing Validation for Reporter Self-Removal as Selector

Classification

Severity: High

Status: Resolved

Description

The system allows a reporter to be removed as their own selector without proper validation. This happens during the removal process where the Selectors.Remove function is called without checking if the selectorAddr corresponds to the reporter's own address.

Reporters are registered as their own selectors by default during registration.

If a reporter is removed as their own selector, they will no longer be eligible to receive rewards, as rewards are distributed through the selector logic.

Without validation, this can lead to significant loss of funds for the reporter, either due to accidental misconfiguration or malicious action.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L219

Recommendation

Add validation to ensure that a selector cannot be removed if it is the reporter's own address. Return an error in such cases to prevent the operation.

Remediation

The vulnerability was fixed in the f33454507abce04cf37f0dcdc10b87260e228476.

TS-H8 - Incorrect Reporter Selection in SetVoterReporterStake Due to Current Block Query

Classification

Severity: High

Status: Resolved

Description

The SetVoterReporterStake function determines the selector's reporter based on the current block state. However, selectors can switch their reporters using MsgSwitchReporter. This introduces a problem: if a selector switches their reporter after a dispute is created, the reporter queried will be the new one, not the reporter that was associated with the selector at the time of the dispute's creation.

This mismatch leads to incorrect computation of voting power, as the reporter's stake at the time of the dispute block height is not accurately reflected in the calculations.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/vote.go#L116

Recommendation

Modify *SetVoterReporterStake* to query the selector's reporter as of the dispute block height, not the current block height.

Remediation

The vulnerability was fixed in the 98fc88e541ed184be277a01f63ae5543dca928b4.

TS-H9 - Inaccurate Voting Power Calculation Due to Early BlockInfo Snapshot

Classification

Severity: High

Status: Resolved

Description

The *BlockInfo* used for calculating voting power percentages for Users and Reporters is recorded immediately after a dispute is created. This snapshot stores *k.reporterKeeper.TotalReporterPower* and *k.oracleKeeper.GetTotalTips*.

If a dispute is created in the same transaction where selectors increase their delegation or users add tips, the TotalReporterPower and *GetTotalTips* values in BlockInfo do not include the updated values. This results in an incorrect denominator when calculating voting power percentages, unfairly increasing the user's voting power relative to others.

This is possible by adding two messages within a transaction. The latter message executed by selectors/users will increase the *TotalReporterPower* and *GetTotalTips*, but the *BlockInfo* will not record it.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/dispute.go#L134

Recommendation

Move the *SetBlockInfo* logic from its current location to the *EndBlocker* so that *BlockInfo* records the correct values after all delegations and tip additions for the block have been processed.

Remediation

The vulnerability was fixed in the 69a044408c4837b8fb37e91d48bc90500ba07758.

TS-H10 - Double Voting Power Inflation via MsgUpdateTeam

Classification

Severity: High

Status: Resolved

Description

The *params.TeamAddress* is used to determine the team's address for voting. When the team votes, their voting power is increased by 25,000,000. However, if the team votes and then changes the *TeamAddress* using *MsgUpdateTeam*, the new address can vote again, inflating the voting power by another *25 000 000*.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/vote.go#L43

Recommendation

In the *MsgUpdateTeam* logic, ensure that any voting power already assigned to the previous TeamAddress is removed or transferred to the new *TeamAddress* before updating the address.

Remediation

The vulnerability was fixed in the 80153cb46c42d3693c143947fb70040b25d27f9b.

TS-H11 - Overwriting DisputeFeePayer Entries Causes Loss of Funds in Refunds

Classification

Severity: High

Status: Resolved

Description

The *MsgAddFeeToDispute* logic overwrites the existing *DisputeFeePayer* entry if a user adds fees multiple times to the same dispute. This results in the previous fee contribution being lost, and the user is unable to receive a refund for their earlier contributions when they call *MsgWithdrawFeeRefund*.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/msg_server_add_fee_to_dispute.go#L64

Recommendation

Accumulate the Amount in *DisputeFeePayer* for multiple fee additions while ensuring the *FromBond* value remains consistent to prevent refund misallocation.

Remediation

The vulnerability was fixed in the a90bd9ea59376365ae416ccc121f032f46435763.

TS-H12 - Incorrect Pool Allocation in ReturnSlashedTokens Function

Classification

Severity: High

Status: Resolved

Description

The *ReturnSlashedTokens* function inappropriately sends slashed tokens to the *BondedPoolName* assuming all funds are delegated to validators with bonded status. However, the *k.reporterKeeper.ReturnSlashedTokens* function may delegate these tokens to validators with Unbonded or Unbonding status.

This mismatch causes funds to be incorrectly allocated, potentially depleting the *NotBondedPoolName* pool.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/dispute_fee.go#L50

Recommendation

Update the *ReturnSlashedTokens* function to allocate tokens to the correct pool (*BondedPoolName* or *NotBondedPoolName*) based on their validator status.

Remediation

The vulnerability was fixed in the b94f6bde4d00031cc7e8a5eb786ffd73d53bddc7.

TS-H13 - AddDisputeRound does not record DisputeFeePayer for payer

Classification

Severity: High

Status: Resolved

Description

The *AddDisputeRound* function is called to add a new round to an existing unresolved dispute, and it increases the *dispute.FeeTotal* with the fee paid by the user. However, the state for *DisputeFeePayer* is not updated to record the fee amount paid by the user in this new round.

As a result, when the user attempts to call *WithdrawFeeRefund*, the state lacks information about their contribution to the dispute fees, and they are unable to receive a refund. This causes loss of funds for users who participate in the dispute process.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/dispute.go#L228

Recommendation

Ensure the *AddDisputeRound* function updates the *DisputeFeePayer* state to record the user's fee contribution for accurate refunds.

Remediation

The vulnerability was fixed in the e3e91cc9897e7540e27de85f6a4adc031dff97a9.

TS-H14 - Invalid Minimum Bonded Tokens Check

Classification

Severity: **High**

Status: **Resolved**

Description

The *CreateReporter* function includes a guard that requires a minimum amount of TRB tokens staked to create a reporter, intended to mitigate potential denial of service (DoS) attacks. However, the implementation erroneously compares the *minTRB* value with the raw integer value (Int) retrieved from the staking module. Since the staking module defines TRB in its smallest unit (*loya*), this results in a comparison mismatch. Specifically, the system compares minTRB against loya (1 TRB = 10^6 loya), causing the required stake to be 10^6 times less than intended. This miscalculation significantly reduces the economic threshold for creating a reporter, undermining the protective mechanism against spam or abuse.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L45

Recommendation

Update the *CreateReporter* function to calculate the minimal required TRB value expressed in *loya*. Then, use that value for comparison to ensure the minimal bonded tokens threshold is met.

Remediation

The vulnerability was fixed in the 589d54d4f1739730ca165f1ca5e1725217b365bf.

TS-H15 - Bypassing Stake Change Validation Using MsgExec from the authz Module

Classification

Severity: High

Status: Resolved

Description

The project implements a *TrackStakeChangesDecorator* to enforce a 5% threshold on staking-related changes (e.g., delegations, undelegations, or redelegations) within a single transaction. The decorator evaluates the transaction messages to determine if the total stake change exceeds the allowed threshold.

However, the decorator does not handle staking-related messages wrapped inside *MsgExec* from the authz module. The authz module allows one account (grantee) to execute messages delegated by another account (granter) using the *MsgExec* message type.

If a staking message is wrapped in *MsgExec*, the decorator does not process the inner messages, bypassing the 5% threshold validation. This allows malicious actors to perform stake changes beyond the allowed limit without being detected.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/ante/ante.go#L30

Recommendation

Update the *TrackStakeChangesDecorator* to handle *MsgExec* explicitly. This involves unwrapping the messages contained within *MsgExec* and applying the stake change validation logic to those messages recursively. Example on how evmos performs this check.

Remediation

The vulnerability was fixed in the 8940a6b0def303a81cc700e1e75783f136b25ff0.

TS-H16 - Incomplete and Faulty Dispute Rounds Logic Leading to Multiple Issues

Classification

Severity: High

Status: Partially Resolved

Description

The current implementation of dispute rounds has several flaws that deviate from the intended functionality described in the documentation, resulting in significant issues in the dispute process.

Single Dispute Round Limitation
The dispute mechanism does not support multiple rounds as intended. The *AddDisputeRound* function automatically invokes *CloseDispute*, which closes the dispute after the first round without providing any mechanism to reopen it. According to the documentation, disputes should continue through multiple rounds until consensus is reached, with dispute fees doubling after the initial 5% in each successive round. The absence of this functionality limits the system's ability to achieve the intended consensus-driven resolution.

Overflow Risk with BurnAmount + RoundFee
In the current implementation, a check is present to ensure *RoundFee* does not exceed *SlashAmount*. However, there is no validation to prevent the sum of *BurnAmount* and *RoundFee* from exceeding *SlashAmount*. This oversight introduces a risk of overflow, which could lead to a critical chain crash during the dispute resolution process. Proper validation is necessary to prevent this scenario and ensure the integrity of the system.

Unutilized RoundFee Leading to Incorrect Rewards
The dispute.BurnAmount is increased by the RoundFee, presumably to distribute the additional fees to reporters and selectors if the dispute resolves with a majority against. However, the calculation of *disputeFeeMinusBurn (SlashAmount - BurnAmount)* does not include *RoundFee*, effectively leaving it unutilized and stuck in the *x/dispute* module account. This results in reporters and selectors receiving less than their fair share of rewards.

Example:
If *SlashAmount* = 100, *BurnAmount* = 5, and *RoundFee* = 20, the correct *disputeFeeMinusBurn* should be 115. However, the current implementation calculates it as 75, meaning reporters and selectors receive less than intended, and the difference (20) remains locked in the module account.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/dispute.go#L233

Recommendation

Enable Multiple Rounds:
Refactor the dispute logic to allow multiple rounds of voting until consensus is achieved. Ensure disputes can remain open until the defined DisputeEndTime is reached.

Add Overflow Validation:
Include a check to ensure *BurnAmount* + *RoundFee* does not exceed *SlashAmount* before processing additional rounds.

Track RoundFee Separately:
Introduce a separate state field for *RoundFee* instead of increasing *BurnAmount*. Update the formula for *disputeFeeMinusBurn* and *disputeBurnAmountDec* to include the *RoundFee*.

Remediation

The vulnerability was partially fixed in the *5800e0b29d8b899daba49ee7a7aa8b92fa0205ec* commit. We also found “todo” comments indicating plans for a complete fix in a future release.

TS-M1 - Possible Out-of-Gas Errors When Claiming Deposits

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *ClaimDeposit* function calls the *GetAggregateByIndex* function when a user claims a deposit. The provided *queryId* will be iterated in the *Aggregates* state until the *currentIndex* matches the requested report index. If the index does not match, it will repeatedly increment until it finds the requested index, returning an error otherwise.

The issue is that if a deposit query ID grows large such that there are many *Aggregates* entries stored (e.g., many deposit requests pending to be claimed), an out-of-gas error may occur when iterating over the *Aggregates* entries. This will cause users to be unable to claim their deposits and, thus, unable to access their funds.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/aggregate.go#L242-L264

Recommendation

Change the implementation so the *Aggregates* map is accessed directly via the index instead of being looped over.

Remediation

The vulnerability was fixed in the *8e06a19e53ae4f2c74ac884790be92f6960ae55f* commit.

TS-M2 - QueryDecodeValueRequest Is Susceptible to Denial-of-Service Attack

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *QueryDecodeValueRequest* allows users to invoke the *DecodeValue* method with arbitrary value data. This data is processed by decoding it from hexadecimal to a string and subsequently parsing it as JSON. An attacker can exploit this behavior to execute a Denial-of-Service (DoS) attack. The attack could unfold as follows:

1. The attacker registers a specification with a data type containing an excessive number of delimiters (e.g., int,int,int,...) to create computationally intensive JSON structures.
2. The attacker sends a high volume of *QueryDecodeValueRequest* messages, each containing a large hexadecimal value.
3. The node expends significant computational resources decoding and parsing the data, leading to high overhead and a potential DoS condition.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/types/decoding.go#L19-L61

Recommendation

Define the maximal accepted size of provided data and check that caller-provided data does not exceed this limit.

Remediation

The vulnerability was fixed in the *d0c3e81de8d2ed6a71b0a039e43a18c9084d94e2* commit.

TS-M3 - FeefromReporterStake Function Is Biased Towards Penalizing Validators With Low Address Byte-string Representation

Classification

Severity: **Medium**

Status: **Acknowledged**

Description

The *FeefromReporterStake* function enables a reporter to utilize a portion of their stake to pay dispute fees. The process involves iterating through the reporter's selectors and their associated delegations to compute the reporter's total stake. Once the total stake is determined, the fee is proportionally distributed among the selectors. This distribution is achieved by performing partial unbonding operations.

To retrieve the delegations for each selector, the function uses the *IterateDelegatorDelegations* method. It iterates through delegations and appends them to the *selectorSharesList* in the order returned by the iterator. When unbonding funds, a saturation mechanism is applied, where funds are removed from delegations starting at the beginning of the *selectorSharesList*.

However, delegations are stored in the underlying data structure using a *(delegatorAddress, valAddr)* key. As a result:

1. The iterator keeps the *delegatorAddress* constant and iterates over the *valAddr* in lexicographical order.
2. This ordering disproportionately impacts validators with lower byte-string representations of their addresses, as they are the first to be depleted during unbending.

This approach introduces an unfair bias, penalizing certain validators purely based on the lexicographical ordering of their addresses rather than on any equitable or proportional basis.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/withdraw.go#L54-L144

Recommendation

Change the implementation so that it does not depend on the lexicographic ordering. The *FeefromReporterStake* function should treat all validators without bias.

Remediation

This finding was accepted.

TS-M4 - UpdateDispute Function Can Lead to Chain Halt

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *UpdateDispute* function is responsible for checking who the winner of the dispute is and sets the *VoteResult*. However, a certain edge case was identified. If the voting ends as a tie between two categories, the function returns a "no majority" error, and the dispute is not updated to any final result.

The *UpdateDispute* function is called during the *TallyVote* function execution. There are two cases in which *TallyVote* is called:

- During the execution of the *MsgVote* handler. In this case, the identified error would prevent users from voting due to execution revert.
- During the execution of *CheckOpenDisputesForExpiration*, which runs on each *BeginBlocker*. This case is significantly more impactful, as it would result in each *BeginBlocker* executing with an error, halting the whole chain.

It must be noted that although the impact of this issue is critical, the actual likelihood is relatively low. Hence, the overall severity is Medium.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/tally.go#L241-L277

Recommendation

Change the implementation so the *UpdateDispute* function does not cause a revert. A new *VoteResult* for tie should be introduced. Alternatively, this case could be a no-op execution.

Remediation

The vulnerability was fixed in the *4425b0b97f34c30ca510db53671cbf77d2ac0ab5* commit.

TS-M5 - FeeRound Function Is Susceptible To Out-Of-Gas Errors

Classification

Severity: **Medium**

Status: **Mitigated**

Description

The *FeeRefund* function is executed during the handling of the *MsgWithdrawFeeRefund* message. The function iterates over all entries in *trackedFees.TokenOrigins* and performs a delegation operation for each entry. Each delegation triggers the execution of all delegation hooks in the staking module.

The size of the *trackedFees.TokenOrigins* slice grows with each dispute creation and dispute round. Given the constraints of the Cosmos ecosystem:

- A reporter can have up to 200 selectors.
- Each selector can have up to 100 delegations (average number of validators in Cosmos chains).

In the worst case, the total number of elements in *TokenOrigins* is:

$$N \times 200 \times 100 = N \times 20,000$$

Where *N* represents the number of dispute rounds.

As the number of elements increases, the function may become not executable due to gas exhaustion. This would make it impossible to refund fees, potentially leading to operational disruptions.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/distribution.go#L141-L181

Recommendation

Rethink the storage design to try to decrease the number of iterations. If the storage cannot be redesigned to reduce the number of iterations, consider lowering the number of selectors.

Remediation

The vulnerability was mitigated in the *d38b5cef76966831a3370a2a1245d72f818dc77b* commit as the number of iterations are decreased.

TS-M6 - CheckOpenDisputesForExpiration Function Can Lead to Chain Halt

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *CheckOpenDisputesForExpiration* function is executed in the *BeginBlocker*. It iterates over all *OpenDisputes* and performs certain operations on them, including computationally expensive *TallyVote*. In case a large amount of disputes is open, it could lead to having the *BeginBlocker* take more time to complete the execution, potentially slowing down the chain or halt it in the worst case if the *cometBFT* round timeout is triggered. Additionally, malicious actors could try to leverage this to bring down the chain. The cost of opening a *Dispute* is 0.01 TRB, which is about 0.5$ at the current exchange rate. That means that it is relatively cheap to store a large amount of disputes.

Code location: https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/abci.go#L12-L57

Recommendation

Implement a batching mechanism so that a fixed maximum number of disputes can be processed during every *BeginBlocker* execution.

Remediation

The vulnerability was fixed in the *94cd03e8ac1f724edaf5901111b5fe770be4dfbe* commit.

TS-M7 - SetAggregatedReport Function Can Cause Chain Halt

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *SetAggregatedReport* is executed during the *EndBlocker* of the oracle module. This function iterates through all the queries with at least one report, and then for each of them, in case it is expired, it iterates all the submitted reports and aggregates data, performing a sorting operation. Both the number of queries and the number of reporters are unbounded, and the asymptothic complexity is *O(qrrlog(r))* where *q* is the number of queries and *r* is the number of reporters. This can lead to having nodes spend more time computing this function, potentially slowing down the chain or leading to a chain halt in case the *cometBFT* round timer is triggered before the conclusion of the *finalizeBlock*.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/aggregate.go#L27-L103

Recommendation

Implement a report aggregation mechanism on the go when storing them to offload the work on the message handler. Additionally, implement a batching mechanism for queries. Furthermore, review the design and make sure that loops are not unnecessarily repeated, i.e., if it is possible to perform certain operations using only one loop, then that is the way it should be implemented.

Remediation

The vulnerability was fixed in the *451* Merge Request.

TS-M8 - Lack of Minimum Tip Requirement Leads to Disproportionally Low Storage Costs

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *Tip* function enables users to either tip an existing query or create a new one if it does not already exist. The function requires the user to provide funds for the tip which also serves to discourage spam when creating a new query. However, while the implementation ensures that the amount is non-negative, it does not enforce a meaningful minimum threshold.

As a result, a user can pay as little as 1 *loya* (approximately $0.00005 USD with the current exchange rate) to store query data on-chain. This amount is disproportionately low compared to the cost of consuming valuable blockchain storage and could allow malicious actors to create numerous queries at negligible cost.

Code location:

https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/msg_server_tip.go#L34-L65

Recommendation

Introduce a governance-controlled parameter that would dictate a minimum required amount of tokens for the tip and assure that the funds provided in the *Tip* function meet this requirement.

Remediation

The vulnerability was fixed in the *f4841070ca9a5ec50b04a8bd99f2c4a8dbd1fa46* commit.

TS-M9 - Proposer Can Provide Arbitrary Number of Votes Due To Missing Check

Classification

Severity: Medium

Status: Resolved

Description

It was observed that there is no check on the actual number of Votes in the *PrepareProposalHandler* function. The number of Votes should be <= cardinality of bonded validators. This is not enforced and the proposer can provide an arbitrary number of votes.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/app/proposal_handler.go#L65

Recommendation

Implement a verification mechanism that will query the number of bonded validators and then ensure that *len(votes) <= len(bondedValidators)*.

Remediation

The vulnerability was fixed in the *23e66dff5808ef18de5c5b8cf4a1f53cbd0be734* commit.

TS-M10 - Bonded Tokens Snapshot Does Not Take Into Account Slashing Events

Classification

Severity: Medium

Status: Acknowledged

Description

In the reporter module's *AnteHandler*, the threshold is validated by comparing *TotalBondedTokens* with the *lastUpdated* snapshot. However, no adjustments are made to account for slashing events. As a result, when slashing occurs, *TotalBondedTokens* is reduced relative to the *lastUpdated* value. This discrepancy causes computations and decisions regarding transaction blocking or allowance to become inaccurate until the next snapshot is registered.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/ante/ante.go#L54-L74

Recommendation

Implement slashing hooks that would modify the saved snapshot each time the slashing occurred so that the *AnteHandler* function operates on the up-to-date data.

Remediation

This finding was accepted.

TS-M11 - The Reporter Module Does Not Implement Bonding-related Hooks

Classification

Severity: **Medium**

Status: **Acknowledged**

Description

The reporter module does not implement *AfterValidatorBonded* and *AfterValidatorBeginUnbonding* hooks. However, since the module keeps track of the number of delegations to bonded validators for selectors, it should also adjust the *DelegationsCount* in case a validator unbonds (removing the relevant delegations) and if a validator bonds (increasing the relevant delegations).

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/hooks.go#L28-L34

Recommendation

Implement *AfterValidatorBonded* and *AfterValidatorBeginUnbonding* hooks to keep track of *DelegationsCount* value.

Remediation

This finding was accepted.

TS-M12 - MsgWithdrawTip Bypasses 5% Stake Change Restriction

Classification

Severity: **Medium**

Status: **Resolved**

Description

The MsgWithdrawTip function allows selectors to delegate their accumulated tips to a bonded validator. However, the *TrackStakeChangesDecorator* Ante Handler, which enforces a restriction on stake changes exceeding 5% of the total bonded funds, is not applied to *MsgWithdrawTip*.

This oversight means that *MsgWithdrawTip* can be exploited to bypass the 5% restriction, allowing selectors to make stake changes that exceed the limit. Such an exploitation could destabilize the staking pool and affect the governance and security of the network.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L294

Recommendation

Integrate the *TrackStakeChangesDecorator* Ante Handler or equivalent validation logic into the *MsgWithdrawTip* process to enforce the 5% restriction on stake changes.

Remediation

The vulnerability was fixed in the commit *659b0ed9213befb3afb44319cf43f2f6845ccd18*.

TS-M13 - Potentially Expensive FeefromReporterStake execution

Classification

Severity: **Medium**

Status: **Acknowledged**

Description

The *FeefromReporterStake* allows a reporter to use part of its stake to pay the fee for a dispute. Its selectors and delegations are iterated to compute the actual stake, and then the fee is equally distributed between all selectors by performing partial unbondings.

This function involves two key iterative processes that dominate its complexity:

- The first loop collects selectors and calculates the total tokens delegated for a reporter. Worst-case assumptions:
  - Maximum selectors per reporter: 200.
  - Delegations per selector: 100 (usually, on Cosmos chains, there are 100 bonded validators).
  - Complexity: Each selector iterates over its delegations, resulting in a total of: $O(Selectors \times Delegations) = O(200 \times 100) = O(20,000)$

- The second loop unbonds shares to pay the fee Worst-case assumptions:
  - Same as the collection loop: 200 selectors × 100 delegations per selector. It is also worth noting that every one of these iterations performs a call to *Unbound*. This equals another $O(Selectors \times Delegations) = O(200 \times 100) = O(20,000)$

Overall computation complexity is equal to: $O(2 \times Selectors \times Delegations)$, which in the worst case will result in 40 000 iterations.

Such a complexity is manageable for off-chain computation. However, it will be quite costly for on-chain execution, which can also result in an execution running out of gas.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/withdraw.go#L33

Recommendation

We recommend revisiting the design of this mechanism and considering a different approach. Namely, constructing the aggregated data incrementally when processing smaller, user-prompted, messages instead of iterating over all delegations or selectors every time. This incremental approach could be implemented using the staking module's hooks.

Remediation

This finding was accepted.

TS-M14 - Possible Out-of-Gas Error in MsgVote Due to Retrieving All Voters

Classification

Severity: **Medium**

Status: **Resolved**

Description

The *GetVoters* function retrieves all voters who participated in a dispute using the dispute ID. When there are a large number of voters for a particular dispute, the process of retrieving all voters can cause the transaction to fail due to an out-of-gas error.

This affects the functionality of *MsgVote*, as it relies on *GetVoters* to check if there are any voters for the dispute. The current implementation unnecessarily retrieves all voters to perform a simple check (*if len(allVoters) == 0*), which is computationally inefficient and prone to failure.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/tally.go#L223

Recommendation

Optimize the logic to only iterate through the voter entries once and check if the first entry exists, rather than retrieving all voters.

Remediation

The vulnerability was fixed in the commit dc3e4b570be805a5f43c8b7d2d16b84be066f2bf.

TS-M15 - Inaccurate Voting Power Due to Dynamic Total Supply in the Current Block

Classification

Severity: **Medium**

Status: **Resolved**

Description

When computing the percentage of voting power for token holders, the denominator is determined by the total supply (*GetTotalSupply*) in the current block. This is problematic if the total supply changes after the dispute was initiated but within the same block, for example:

- Tokens are minted, increasing the total supply.
- Tokens are burned, decreasing the total supply.

This dynamic denominator creates an inconsistency between the voting power calculation and the token supply at the time of the dispute creation, leading to inaccurate results.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/tally.go#L186

Recommendation

The total supply used in voting power calculations should be based on the past block height when the dispute was created, not the current block height.

Remediation

The vulnerability was fixed in the a2a64b4ee013eda4f02b6046c828110893ae5c09.

TS-M16 - Vulnerable cometbft package

Classification

Severity: **Medium**

Status: **Resolved**

Description

Two vulnerabilities were identified due to outdated cometbft (github.com/cometbft/cometbft)

1. GO-2024-3259. Specifically, the system panics upon receiving a pre-commit message containing invalid data. This issue could be exploited to disrupt consensus operations, leading to a denial of service (DoS) in applications utilizing CometBFT for consensus.

2. GO-2024-3112. In the state sync process, where a syncing validator could accept invalid state data from a malicious node. Exploiting this flaw can result in a chain split, causing network instability and undermining the integrity of the blockchain.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/go.mod

Remediation

We recommend upgrading to github.com/cometbft/cometbft@v0.38.15, which resolves this issue.

Remediation

The vulnerability was fixed in the *58139d68de2f7db99c884b0ede5e3466a16b2c55* commit.

Resources

https://pkg.go.dev/vuln/GO-2024-3259 https://github.com/advisories/GHSA-p7mv-53f2-4cwj

https://pkg.go.dev/vuln/GO-2024-3112
https://github.com/cometbft/cometbft/security/advisories/GHSA-g5xx-c4hv-9ccc

TS-L1 - Incomplete ValidateBasic Function Implementation for MsgSubmitValue

Classification

Severity: Low

Status: Resolved

Description

The *MsgSubmitValue* implements an incomplete *ValidateBasic* function. It only verifies the validity of the *Creator* field while not ensuring that neither *QueryData* nor *Value* are non-empty.

It was observed, however, that the complete validation is implemented for the *GetSignerAndValidateMsg* function.

Implementing complete validation in *ValidateBasic* has additional benefits. Namely, it allows invalid transactions to be dropped even before reaching mempool. This, in turn, keeps the mempool smaller and prevents the node from allocating block space to processing transactions related to data known to fail.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/types/message_submit_value.go#L50-L70

Recommendation

Implement a complete validation in *ValidateBasic* function.

Remediation

The vulnerability was fixed in the *ee7e132dfc32a9e3e7ebc5c098d88e2db80bfddc* commit.

TS-L2 - Missing Sanitization And Validation In UpdateDataSpec

Classification

Severity: Low

Status: Resolved

Description

The same sanitization and validation mechanisms for updating data spec should be implemented as they are for registering a new data spec.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/keeper/msg_update_spec.go#L16-L33

Recommendation

Implement the necessary sanitization and validation, ideally those should be:

- *msg.Spec.ResponseValueType = strings.ToLower(msg.Spec.ResponseValueType)* and *msg.Spec.AggregationMethod = strings.ToLower(msg.Spec.AggregationMethod)*
- *!types.SupportedType(msg.Spec.ResponseValueType)*
- *!types.SupportedAggregationMethod[msg.Spec.AggregationMethod]*

Remediation

The vulnerability was fixed in the *866f1a9dc4f731947b0b793d1734c0534d1d5849* commit.

TS-L3 - Incomplete ValidateBasic Implementation for MsgCreateReporter

Classification

Severity: Low

Status: Resolved

Description

Incomplete validation in the *ValidateBasic* of the *MsgCreateReporter*. *MinTokensRequired* should be ensured to be a positive number.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/types/message_create_reporter.go#L13-L26

Recommendation

Implement a validation ensuring that *MinTokensRequired* > *0*.

Remediation

The vulnerability was fixed in the 7badfe4b8fbeb3df5d47e4c11c15abd791de3a8b commit.

TS-L4 - Chain Halt Due To Empty Cyclelist

Classification

Severity: Low

Status: Resolved

Description

The *RotateQueries* function is executed in the *EndBlocker* of the oracle module. This function gets the *CycleList* and accesses the latest query, which is at position *n*. However, if the *cycleList* is empty (the admin can empty it with the *MsgUpdateCycleList*), it would cause the chain to halt due to a panic. Specifically:

max == 0:

1. If the cycle list is empty (*max = 0*), then *max-1* will underflow to a large positive integer (*uint64* wraps around from 0 - 1 to 2^64 - 1).
2. This would result in the case condition *n >= uint64(max-1)* always evaluating to *false*, and the function proceeds to *n += 1*.
3. Later in the function, *n* is used to access an element from the cycle list.
4. If the cycle list (q) is empty (max = 0), this will result in a panic due to an index being out of range.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/cycle_list.go#L57-L69

Recommendation

Implement a mechanism that will ensure the described scenario cannot take place. It can be done by handling this case explicitly in the *RotateQueries* function or by ensuring that the Cyclelist cannot be empty.

Remediation

The vulnerability was fixed in the *166e20e04f2e02606a7630632e492d14c2007507* commit.

TS-L5 - Missing Cyclelist Validation in UpdateCyclelist Function

Classification

Severity: Low

Status: Resolved

Description

The *UpdateCycleList* function enables an authorized entity to replace the current *cycleList* with a new list provided as input. However, the function performs no validation on the input data. It simply iterates through the *cycleList*, hashes each element to generate an ID, and stores the data.

Code location:

https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/msg_update_cyclelist.go#L19-L41

Recommendation

Implement a validation on the newly provided *cycleList* to ensure the provided data is meaningful and will not disrupt the chain's operation.

Remediation

The vulnerability was fixed in the 64927ed1af4ed7b9423540c6d44cf2650374a60c and a058cb6d6c55341982b64ed34d87b10189c42a6e.

Remediation

The vulnerability was fixed in the *79c9d7c22a99353c14debd534d0edf278c350e0d* commit.

TS-L6 - Block Stuffing Due To Unlimited queryData in PreventBridgeWithdrawalReport function

Classification

Severity: Low

Status: Resolved

Description

The *PreventBridgeWithdrawalReport* function is executed during the handling of the *MsgSubmitValue* message. In this process, a user can supply an arbitrary *queryData* byte slice, which is subsequently parsed based on an *abi.Arguments* definition. However, the function does not impose a maximum size limit on the provided byte slice.

Although the *anteHandler* charges gas proportional to the size of the input, an attacker could willingly pay the gas fees to submit excessively large *queryData* slices. This could result in block stuffing, where maliciously large inputs fill up block space, thereby disrupting normal network activity.

Code location: https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/keeper/token_bridge_withdrawal_blocker.go#L14-L18

Recommendation

Implement a size limit check on the *queryData*.

Remediation

The vulnerability was fixed in the *83933c5582d6bad9623160bf7d18abb16010c6a4* commit.

TS-L7 - WithdrawTip Function Bypasses the Staking Module

Classification

Severity: Low

Status: Resolved

Description

The *WithdrawTip* function takes the rewards accumulated by the selectors and delegates those to a bonded validator. To do so, it first executes the *Delegate* function, passing *false* for the *subtractAccount* parameter (used for redelegation) and then directly transfers funds from *TipsEscrowPool* to *BondedPoolName*. However, this bypasses the execution of the *DelegateCoinsFromAccountToModule*, which is the staking module's function used for delegation. Consequently, some validations and the tracking hook for vested accounts are not applied.

Code location:

https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L318-L341

Recommendation

Utilize the *DelegateCoinsFromAccountToModule* to leverage the staking module's validations and hooks.

Remediation

The vulnerability was fixed in the *8c8db213c952001375984f13108f70e966b5ea2a* commit.

TS-L8 - Permissionless RegisterSpec Function Leads to Malicious Behaviour

Classification

Severity: Low

Status: Acknowledged

Description

This function is permissionless, allowing anyone to register data spec for *QueryType*. This could enable attackers to front-run users by registering valid *QueryTypes* with bogus Specs, such as “BTC,” “ETH,” etc. Attackers could scale this approach and store Specs for all QueryTypes with seemingly legitimate names.

It is worth noting that if a malicious user creates such a Spec, the authority can still update the changes via *MsgUpdateDataSpec*. However, the authority is set to the governance, which usually takes a relatively long time to propose, vote, and execute the proposal.

Code location:

https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/keeper/msg_server_register_spec.go#L15-L24

Recommendation

Implement an allowlist of parties that are allowed to create new Specs. Alternatively, if it is required for the functionality to remain permissionless, introduce a fee associated with registering a new Spec to act as a deterrent.

Remediation

This finding was accepted.

TS-L9 - Possible Invalid Handling of Voting Power

Classification

Severity: Low

Status: Resolved

Description

If selectors vote after the reporter has vote, the reporter's power (*reporterVote.ReporterPower*) will be decreased.

However, the *reporterVote.VoterPower* needs to be decreased too because it included the reporter's voting power before in https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/msg_server_vote.go#L66-L72.

Please note that this finding is Low as no direct impact was identified during the engagement. However, it still should be resolved for correctness' sake and to avoid potential bugs in the future.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/vote.go#L155-L160

Recommendation

Decrease *reporterVote.VoterPower* by *selectorTokens*.

Remediation

The vulnerability was fixed in the *e6a7974434fedec104aab4b9bf4ad9b4d52a4c48* commit.

TS-L10 - Incomplete Export Genesis

Classification

Severity: Low

Status: Resolved

Description

The incomplete Export Genesis was observed in several modules.

Oracle Module The ExportGenesis function does not export the Cyclelist. Consequently, if the chain needs to be halted and restarted for an update or a hard fork, restoring the previous state of the Cyclelist would not be possible.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/genesis.go#L24

Registry Module The ExportGenesis function in the registry module is designed to export the GenesisState, enabling re-importation during hard forks or specific updates. However, the current implementation fails to include all information related to the Dataspec. Specifically, only the genQueryTypeSpotPrice specification is exported, while all other Dataspec entries are discarded. This results in incomplete state preservation and loss of critical data during re-importation.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/module/genesis.go#L53

Mint Module The ExportGenesis function in the *x/mint/keeper/genesis.go:22-30* is responsible for exporting the genesis state of the blockchain. However, the function currently exports only a partial representation of the *Minter* object, omitting key fields such as Initialized and *PreviousBlockTime*. This limitation poses a functional issue during chain upgrades, specifically hard forks, where the re-importation of the incomplete genesis state may lead to inconsistent chain behavior or outright failure.

Similar case can be observed in the *InitGenesis* function.

Bridge Module The SnapshotLimit state is configured but is not exported correctly to the genesis state. Currently, it is exported as *DefaultSnapshotLimit* value, which is 1000 instead of *k.SnapshotLimit.Get* value.

Recommendation

Change the implementation of the aforementioned *ExportGenesis* functions so that it exports the complete Dataspec.

Remediation

The vulnerability was fixed in the *b62f9b383586fa43dafb1136f6c91de32ee4f2f4* commit.

TS-L11 - Potentially Incorrect Implementation of MsgRemoveSelector handler

Classification

Severity: Low

Status: Resolved

Description

The *MsgRemoveSelector* allows anyone to remove a selector from a reporter if the reporter has reached max selectors and the selector does not satisfy the *reporter.MinTokensRequired* requirement, as per the comment in the code.

However, the current condition *if len(selectors) <= int(params.MaxSelectors)* returns an error if the reporter has reached max selectors. This contradicts the abovementioned documentation because the selector should be allowed to be removed when the reporter has reached the max selector.

Code location: https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L252

Recommendation

Analyze the code comment to ensure it is correctly describing the intended design. If so, change the *if* statement to:

```
if len(selectors) < int(params.MaxSelectors)
```

Remediation

The vulnerability was fixed in the 41d22af0828af1ef06021487a923b4a1f1c91c53.

TS-L12 - Incomplete Validation of the MsgProposeDispute

Classification

Severity: Low

Status: Resolved

Description

The *ValidateBasic* method of the *MsgProposeDispute* is responsible for performing input validations on dispute messages. However, its current implementation only verifies the validity of the *Creator* address. The method does not validate other critical fields such as *Report*, *DisputeCategory*, and *Fee*, potentially leading to incorrect or inconsistent disputes being proposed.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/types/message_propose_dispute.go#L47

Recommendation

Implement a complete validation that assures every field has a reasonable value.

Remediation

The vulnerability was fixed in the 36cfb0ee0423ee554e8541566a6d7c87400f8ec3.

TS-L13 - Incomplete Implementation of ValidateBasic of MsgRegisterSpec

Classification

Severity: Low

Status: Resolved

Description

The *ValidateBasic* method of the *MsgRegisterSpec* does not perform a comprehensive validation of the Spec structure. Currently, it only verifies that the Registrar is a valid account, but it does not ensure the correctness of the data within the Spec. Additionally, the *QueryType* field is not checked to confirm that it is a non-empty string. This lack of validation could allow the submission of improperly constructed *MsgRegisterSpec* messages.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/types/message_register_spec.go#L38

Recommendation

Implement a complete validation logic to ensure the correctness of data within the Spec, along with making sure that the *QueryType* field is a non-empty string.

Remediation

The vulnerability was fixed in the 9e0a210cb1b4ac5fa7e4218d8cd49fa518e54fad.

TS-L14 - Missing Validation on Params in Registry Module

Classification

Severity: Low

Status: Resolved

Description

The *Validate* function for *Params* in the registry module is currently unimplemented. This omission results in the absence of critical checks, such as verifying that the *MaxReportBufferWindow* parameter is greater than zero. Without this validation, invalid configurations may be accepted, potentially leading to runtime errors or misbehavior.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/types/params.go#L37

Recommendation

Implement a meaningful validation logic for *Params* in the registry module, such as assertion that the *MaxReportBufferWindow* is greater than zero.

Remediation

The vulnerability was fixed in the c9fe5c04fb05f4a8c393307830f1baa31079ea71.

TS-L15 - Incomplete Validation for GenesisState in Registry Module

Classification

Severity: Low

Status: Resolved

Description

The *Validate* method in the *GenesisState* of the registry module is designed to ensure the integrity of the genesis data. However, this validation is incomplete. Currently, it validates only the *Params* field, leaving the *Dataspec* field unchecked. This oversight allows the inclusion of invalid or inconsistent *Dataspec* values during initialization, potentially leading to undefined behaviors or errors later in the application's lifecycle.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/registry/types/genesis.go#L19

Recommendation

Implement a validation on the *Dataspec* field in the *Validate* method of *GenesisState* of registry module.

Remediation

The vulnerability was fixed in the 3e812061a08c5b676c0657598902a13b31d943dc commit.

TS-L16 - Incomplete validation for MsgTip in Oracle module

Classification

Severity: Low

Status: Resolved

Description

The *ValidateBasic* function of the *MsgTip* message does not completely validate the message. It should ensure that *msg.Amount.Denom* is equal to *layer.BondDenom* and that the amount is positive. Additionally, *QueryData* should be non-empty. Some of those validations are performed in the execution logic, however, they should be moved here to drop incorrect messages before adding them in the mempool to discourage DoS.

Code Location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/types/message_tip.go#L50

Recommendation

Implement a verification mechanism to ensure that the *msg.Amount.Denom* matches the *layer.BondDenom* and the actual amount is a positive number. Additionally, make sure that the *QueryData* is non-empty. All validation performed on those values that are part of the execution logic should be moved to *ValidateBasic* function.

Remediation

The vulnerability was fixed in the ccfefc5782d196ca179da380093900f9ab1403fa.

TS-L17 - UnjailReporter Function Fails to Reset JailedUntil Field

Classification

Severity: Low

Status: Resolved

Description

The UnjailReporter function is responsible for unjailing a reporter if their JailedUntil timeout has expired. While it correctly resets the Jailed field to false, it fails to reset the JailedUntil field. This results in inconsistent state data, as the JailedUntil field still reflects the expired timeout even after the reporter has been unjailed.

This oversight can cause confusion in the system and lead to misinterpretations of the reporter's status during future operations, as the outdated JailedUntil value may suggest that the reporter is still under a jail penalty or subject to restrictions.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/jail.go#L40

Recommendation

Update the UnjailReporter function to reset the JailedUntil field to a default value (e.g., *nil* or the zero value for its type) when unjailing a reporter.

Remediation

The vulnerability was fixed in the b7f3433d4a56c623526906bb63d7d97f9347f71e.

TS-L18 - Reporter Should Not Be Allowed to Pay From Bond in Self-Created Disputes

Classification

Severity: Low

Status: Resolved

Description

The validation logic in *AddDisputeRound* correctly prevents a disputed reporter from adding fees from their bond if they are the subject of the dispute. However, in the *ProposeDispute* function, no similar validation exists to prevent a reporter from opening a dispute on themselves and paying from their bond.

Although the likelihood of a reporter initiating a self-dispute is low, the absence of this validation creates a potential inconsistency in logic and could lead to incorrect fee calculations during dispute resolution if it occurs.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/msg_server_add_fee_to_dispute.go#L37

Recommendation

In the *ProposeDispute* function, return an error if the proposer attempts to create a dispute on themselves while paying from their bond.

Remediation

The vulnerability was fixed in the 3631d75fc3d3759adad3c5316eabf5b2d8943ab0.

TS-L19 - Missing Cyclelist validation in GenesisState of the Oracle module

Classification

Severity: Low

Status: Resolved

Description

The *Validate* function of the *GenesisState* in the Oracle module does not validate the *Cyclelist*. If the Genesis Cyclelist is invalid or otherwise unparsable, it will damage the chain operation from the very start.

Code Location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/types/genesis.go#L19

Recommendation

Implement a validation in *Validate* function of the *GenesisState* in the Oracle module. At the very least, it should be checked to contain parsable and meaningful data.

Remediation

The vulnerability was fixed in the a058cb6d6c55341982b64ed34d87b10189c42a6e and 64927ed1af4ed7b9423540c6d44cf2650374a60c.

TS-L20 - Missing validation for Params in Oracle module

Classification

Severity: Low

Status: Resolved

Description

The *Validate* function implemented for *Params* in the Oracle module has a no-op implementation - it just returns *nil*. However, there is a crucial parameter, *MinStakeAmount*, that should be provided within those *Params*.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/oracle/types/params.go#L46

Recommendation

Implement a validation for *Params* that will ensure the *MinStakeAmount* has been provided.

Remediation

The vulnerability was fixed in commits *8c32f7b5a4a576f60ce6012dd03d3c177da04453* and *21ec8c48bd56610b034a9ff53cf4f7169b5f3452*.

TS-L21 - params.MinCommissionRate Not Enforced in CreateReporter Function

Classification

Severity: Low

Status: Resolved

Description

The *CreateReporter* function does not validate the *msg.CommissionRate* against the configured *params.MinCommissionRate*. While the function ensures that the commission rate does not exceed 100%, it does not enforce a minimum rate, allowing reporters to set arbitrarily low commission rates. This oversight renders the *params.MinCommissionRate* configuration ineffective.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/reporter/keeper/msg_server.go#L61

Recommendation

Add validation in the *CreateReporter* function to ensure *msg.CommissionRate* is greater than or equal to *params.MinCommissionRate*.

Remediation

The vulnerability was fixed in the 688c305a7d74c49d40e09b8dfbeca369bf6da8cd.

TS-L22 - Chain Halt Risk When Minted Coins Are Not layer.BondDenom

Classification

Severity: Low

Status: Resolved

Description

If minted coins are not of the expected *layer.BondDenom* (e.g., when configured to mint tokens from *Minter.BondDenom*), *coins.AmountOf(layer.BondDenom)* becomes zero. This leads to a zero-value transaction being constructed and passed to the *InputOutputCoins* function, which fails during validation in the bank keeper. Specifically, *ValidateBasic* rejects transactions with zero coin amounts, causing an error. This error eventually propagates to the ABCI, disrupting block processing and potentially halting the chain.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/mint/keeper/keeper.go#L86

Recommendation

1. Return *nil* early if *coins.AmountOf(layer.BondDenom)* is zero to avoid constructing invalid transactions.
2. Declare *coins.AmountOf(layer.BondDenom)* as a variable to avoid repetitive function calls and optimize gas usage.
3. Replace manual input/output handling with the *SendCoinsFromModuleToModule* function to directly transfer funds, improving readability and maintainability.

Remediation

The vulnerability was fixed in the 22e4d40dd19945d292c67be03b2a466f865318ca.

TS-L23 - CalculateBlockProvision Mints DefaultBondDenom Instead of Minter.BondDenom

Classification

Severity: Low

Status: Resolved

Description

The *CalculateBlockProvision* function always mints coins with the hardcoded *DefaultBondDenom* ("loya") instead of using the configurable *Minter.BondDenom*. While this is not critical due to consistent use of *DefaultBondDenom*, it violates good practices by ignoring dynamic configuration and reducing flexibility.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/mint/types/minter.go#L43

Recommendation

Update *CalculateBlockProvision* to mint coins using *Minter.BondDenom* instead of *DefaultBondDenom*.

Remediation

The vulnerability was fixed in the 511ab617ea5cf21bd8a7b3e9ea2cdeaa50bb0769.

TS-L24 - Vulnerable cosmossdk.io/math package

Classification

Severity: Low

Status: Resolved

Description

The cosmossdk.io/math module contains a vulnerability in the bit-length validation of *sdk.Int* and *sdk.Dec*, specifically *GO-2024-3279*. Inputs with mismatched bit lengths can bypass validation, potentially triggering a panic when processed. This issue could be exploited to cause a denial of service (DoS), affecting the stability and availability of the application.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/go.mod

Remediaiton

We recommend upgrading to cosmossdk.io/math@v1.4.0, which addresses this vulnerability.

Remediation

The vulnerability was fixed in the bda62fa07e497fed019136b2eb382af825910806 commit.

Resources

https://github.com/advisories/GHSA-7225-m954-23v7

TS-I1 - Miscellaneous Comments

Classification

Severity: **Informational**

Status: **Partially Resolved**

Description

Several code improvements were observed:

1. selector.Reporter can be replaced with prevReporter variable for readability (code)

2. line 104 uses TokensFromSharesTruncated while line 125 uses TokensFromShares. Ideally, only one of them should be used for consistency.

3. The InitialCycleList function handles hardcoded encoded values instead of using a structured approach with encoding/decoding, which could improve maintainability and readability.

4. Consider including QueryType: queryType in the QueryMeta initialization to provide additional context for the query.

5. Dead code - the following if statement will never happen - as previous cases protect against that.

6. The *EVMAddressFromSignatures* in x/bridge/keeper/keeper.go is storing hard-coded messages as strings and hashing them each time the function is executed. It would be more efficient to keep the messages hardcoded as already hashed.

7. The *WeightedMedian* function in x/oracle/keeper/weighted_median.go is designed to compute the weighted median value of a query. However, The current implementation selects a value based on the stake distribution of the reporters:

- it identifies the reporter where the cumulative stake power meets or exceeds half of the total stake power.
- the selected reporter's value is returned as the median.

  This logic ties the selection of the "median value" to the distribution of stake power, making the result more about who holds the median stake rather than the numerical median of the reported values. This leads also to a strange alignment of interests where reporters who have a large stake are incentivized to report "large" values and reporters with a small stake are incentivized to report "small" values in order to have their report selected. Using a weighted mean would probably work better.

Recommendation

Address issues mentioned above.

Remediation

Some of the points were addressed in *642056b7ff277310f21917c5de002fc651d7b8e0* commit.

TS-I2 - Redundant Team Vote Tally Computation in Voting Logic

Classification

Severity: **Informational**

Status: **Resolved**

Description

The function responsible for computing the tally of team votes does so by querying the team address and updating *tallies.*.Team based* on their preferences. However, this is redundant, as team votes are already recorded in *voteCounts.Team* during the voting phase.

The team votes can be directly accessed as *voteCounts.Team.Support*, *voteCounts.Team.Against*, and *voteCounts.Team.Invalid*, making the additional computation unnecessary.

Code location:
https://github.com/tellor-io/layer/blob/0da020d31681cd8c43545b44666fbec19c95dacd/x/dispute/keeper/tally.go#L104

Recommendation

Refactor the function to compute team votes directly from *voteCounts.Team* instead of querying and re-computing them.

Remediation

The vulnerability was fixed in the 54a9344fae9b4a2c4b685d004b96dfadeff69b2f, a1d3a4894cdd068d13708297d8c24d9a125d18d6.

Disclaimer

This report has been prepared in accordance with the current best practices and standards applicable at the time of its preparation. It is intended to provide an analysis and evaluation of the subject matter based on the information available, including any potential vulnerabilities, issues, or risks identified during the assessment process. The scope of this analysis is limited to the data, documents, and materials provided for review, and the conclusions drawn are based on the status of the information at the time of the report.

No representation or warranty, express or implied, is made as to the absolute completeness or accuracy of the information contained in this report. It is important to acknowledge that the findings, conclusions, and recommendations presented are subject to change should there be any modifications to the subject matter. This report should not be viewed as aconclusive or exhaustive evaluation of the subject matter's safety, functionality, or reliability. Stakeholders are encouraged to conduct their own independent reviews, assessments, and validations to ensure the integrity and security of the evaluated subject.

The authors and auditors of this report disclaim any liability for any direct, indirect, incidental, or consequential damages or losses that may result from reliance on this report or its contents. It is the responsibility of the stakeholders to ensure the ongoing monitoring and evaluation of the subject matter to mitigate potential risks or vulnerabilities.

The nature of technology and digital systems means that they are inherently subject to risks, including but not limited to vulnerabilities, bugs, and security threats that may not be foreseeable at the time of this report. The dynamic and evolving nature of technological standards, security practices, and threat landscapes means that absolute security cannot be guaranteed. Despite thorough analysis and evaluation, unforeseen vulnerabilities may exist, and new threats may emerge subsequent to the issuance of this report.

The authors and auditors make no guarantee regarding the impenetrability or infallibility of the subject matter under evaluation. Stakeholders are advised to implement comprehensive security measures, including but not limited to regular updates, patches, and monitoring, to safeguard against potential threats and vulnerabilities.