Security Audit Report

Neutron Independent Chain

v1.0

March 25, 2025

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Hadron Labs to perform a security audit of changes that allow Neutron to become an independent chain with migration logic, staking module integration, custom logic for staking rewards, validator compensation, and governance.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.

2. Determine possible vulnerabilities, which could be exploited by an attacker.

3. Determine smart contract bugs, which might lead to unexpected behavior.

4. Analyze whether best practices have been applied during development.

5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following targets:

Phase 1

Repository	https://github.com/neutron-org/neutron-private
Commit	`71f29e201d59cfebc67e70fd8fbde47d62ecef4f`
Scope	Only the `app/upgrades/sovereign` directory was in the scope of the audit.
Fixes verified at commit	`aaba49ce93126f33c6168b8171fc0ed348bbd794` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/neutron-org/neutron-private
Commit	`6c74b900cf84fe859d77c82ee2d4676a19fe4b48`
Scope	Only the `x/harpoon` directory was in the scope of the audit.
Fixes verified at commit	`aaba49ce93126f33c6168b8171fc0ed348bbd794` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/neutron-org/neutron-dao-private
Commit	`822fe1c610ae5043b0defe92f113f1dc719806bd`
Scope	Only the `contracts/dao/voting/neutron-staking-tracker` and `contracts/dao/voting/neutron-staking-vault` directories were in the scope of the audit.
Fixes verified at commit	`ba758e262b4189805deb516141ee301621fd1d50` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Phase 2

Repository	https://github.com/neutron-org/neutron-dao-private
Commit	`b11c30da799ba5f2043ffc583d9fa36f86bd63d6`
Scope	Only the `contracts/dao/neutron-staking-rewards` and `contracts/dao/neutron-staking-info-proxy` directories were in the scope of the audit.
Fixes verified at commit	`ba758e262b4189805deb516141ee301621fd1d50` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/neutron-org/neutron-private
Commit	`74b6f3922bc291ec7d631f5c0c9fc9a8e872b8aa`
Scope	Only the `x/revenue` directory was in the scope of the audit.
Fixes verified at commit	`aaba49ce93126f33c6168b8171fc0ed348bbd794` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

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

Functionality Overview

The audit scope features Neutron to migrate into an independent chain from existing Interchain Security implementation. The migration process includes the following:

- **Validator Set Migration and Staking Module Integration**: transition ICS validator sets to use sovereign validator sets.

- **Governance and Staking Vault Enhancements**: supports DAO DAO–based governance by providing a historical record of user voting power in CosmWasm contracts.

- **Staking Rewards Contract with Fixed Target APR**: provides a mechanism for stakers to earn a fixed target Annual Percentage Rate (APR) of 3%, which is funded by the DAO's treasury.

- **Revenue (`x/revenue`) Module for Validator Incentivization**: motivates validators to maintain high performance by rewarding them based on block production (signing) and timely Oracle price updates (via Slinky).

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium	-
Level of documentation	High	The client provided detailed documentation in Notion pages.
Test coverage	Medium	The reported code coverage for the CosmWasm smart contracts is `63.2%`, with 316/500 lines covered: ● `contracts/dao/voting/neutron-staking-tracker/src/contract.rs`: 223/382 ● `contracts/dao/voting/neutron-staking-tracker/src/state.rs`: 15/16 ● `contracts/dao/voting/neutron-staking-vault/src/contract.rs`: 73/96 ● `contracts/dao/voting/neutron-staking-vault/src/state.rs`: 5/6 For the `app/upgrades/sovereign` upgrade, coverage stands at 22% of statements. For the `x/harpoon` module, after excluding generated `.pb.go` and `.pb.gw.go` files, coverage stands at 59.2% of statements. In addition to unit tests, the client has developed integration tests in a separate repository. Furthermore, they provided instructions on how to run a testnet for the sovereign upgrade (delCS testnet), which we successfully executed, confirming that the expected results were achieved.

Summary of Findings for Phase 1

No	Description	Severity	Status
1	Unsupported staking hooks cause a denial of service	Critical	Acknowledged
2	Failure to handle `BeforeDelegationRemoved` hook leads to stake inconsistency	Major	Resolved
3	Incorrect parameter passed during `SudoMsg::AfterValidatorRemoved`	Major	Resolved
4	Potential incorrect delegation logic in `StakeWithDrop`	Major	Resolved
5	Bypassing the embedded filesystem causes file I/O errors on non-build machines, causing a denial of service	Major	Resolved
6	Blacklist feature does not account for block height, causing incorrect vote computation	Major	Resolved
7	Unbounded validator iteration causes potential out-of-gas errors	Major	Resolved
8	Rounding discrepancy in `before_validator_slashed` leads to misestimation of slashed tokens	Minor	Resolved
9	New validator accounts are not pre-funded, potentially causing upgrade failure	Minor	Resolved
10	Failure to check for contract existence in hook subscriptions may result in a chain halt or denial of service	Minor	Resolved
11	Missing validations in the `x/harpoon` module	Minor	Resolved
12	Risk of chain takeover if staking levels are insufficient	Minor	Acknowledged
13	Unrestricted contract hooks subscription may cause out-of-gas error	Minor	Acknowledged
14	Failure to handle error from `SendCoinsFromModuleToModule` allows execution to continue despite transfer failure	Minor	Resolved
15	Suboptimal staking parameters affect IBC reliability,	Minor	Resolved

	validator limits, and storage efficiency		
16	Inconsistent type usage and redundant parameters in `findHooksToRemove`	Informational	Acknowledged
17	Inconsistent usage of `cons_address`	Informational	Acknowledged
18	Code quality improvements	Informational	Acknowledged
19	Misleading instances in the `neutron-staking-vault` contract	Informational	Acknowledged
20	Misuse of the `Addr` type in the `neutron-staking-tracker` contract	Informational	Acknowledged
21	Misleading comments in the `neutron-staking-tracker` contract	Informational	Acknowledged
22	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
23	Lack of context in comments explaining manual validator set management	Informational	Resolved
24	Redundant code within the sovereign upgrade handler and `AnteHandler` setup options	Informational	Resolved
25	Ineffective unit test for the sovereign upgrade handler	Informational	Resolved
26	Usage of magic numbers decreases maintainability	Informational	Resolved
27	Lack of guaranteed IBC client updates during the upgrade may cause IBC transaction failures	Informational	Resolved
28	Hardcoded contract addresses may cause deployment misconfigurations	Informational	Acknowledged
29	Inconsistent documentation regarding missing validator blacklist in the `AfterDelegationModified` hook	Informational	Resolved
30	Potential confusion during shares-to-tokens comparison	Informational	Resolved

Summary of Findings for Phase 2

No	Description	Severity	Status
31	Validators receive lesser rewards due to incorrect reward denom	Critical	Resolved
32	Validator rewards are lost if the payment schedule is updated	Major	Resolved
33	Failure to update user stake may cause excess reward distribution	Major	Resolved
34	Incorrect stake retrieval for LST protocols leads to zero rewards	Major	Resolved
35	Validators with significant voting power could censor others to prevent them from receiving rewards	Major	Acknowledged
36	Validators can manipulate oracle prices to maximize rewards	Major	Acknowledged
37	Potential denial-of-service due to significant state reads	Major	Acknowledged
38	Missing genesis state validation in `x/revenue`	Minor	Resolved
39	Inconsistent provider definition in the Staking Info Proxy contract and documentation	Minor	Resolved
40	Lack of validation for `TwapWindow` can lead to recent price deletion	Minor	Resolved
41	Potential overflow during block interval calculations	Minor	Resolved
42	Incorrect TWAP response returned from the `PaymentInfo` query	Minor	Resolved
43	Negative prices are not handled	Minor	Resolved
44	Validator rewards are lost due to possible fund shortages	Minor	Resolved
45	Missing upper bounds limits in the `neutron-staking-rewards` contract	Minor	Resolved
46	Unbounded iteration when processing validator payments may slow down or halt the chain	Minor	Acknowledged
47	The DAO address is not blocked from claiming	Minor	Resolved

	rewards		
48	Updating the `Config::staking_denom` field may cause incorrect rewards calculations and failure in claiming rewards	Minor	Acknowledged
49	Potential TWAP window misalignment	Minor	Acknowledged
50	Missing enforcement of validator reward eligibility based on active set status	Minor	Resolved
51	Potential chain halt due to pre-blocker retrieving deleted validator’s state	Informational	Resolved
52	Lack of documentation regarding consensus participation conditions	Informational	Acknowledged
53	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
54	Query returns incomplete information	Informational	Acknowledged
55	Lack of provider attributes in `update_providers` response hinders debugging	Informational	Acknowledged
56	Unhandled parsing errors during `query_providers`	Informational	Acknowledged
57	Code duplication when updating global index	Informational	Acknowledged
58	Inefficient state loading leads to unnecessary resource usage	Informational	Acknowledged
59	Unnecessary string copies and allocations from overuse of `Clone::clone`	Informational	Acknowledged
60	Lack of slashing event cleanups leads to inefficient state management	Informational	Acknowledged
61	Lack of pausing feature	Informational	Acknowledged
62	Misleading comment in the `query_voting_power` function	Informational	Acknowledged
63	The `query_user_stake` function can be optimized	Informational	Acknowledged
64	`to_address` is not checked to be a valid address	Informational	Acknowledged
65	Spelling errors in the codebase	Informational	Acknowledged
66	Usage of magic numbers during reward rate	Informational	Acknowledged

	conversion		
67	Optimization by storing precomputed reward rate per block for gas efficiency	**Informational**	**Acknowledged**

Detailed Findings for Phase 1

1. Unsupported staking hooks cause a denial of service

Severity: Critical

In `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:199-223`, `sudo` messages are called when staking actions are triggered to maintain consistency with the `x/staking` module.

The issue is that the `BeforeDelegationSharesModified` and `BeforeDelegationRemoved` hooks are not implemented in the `sudo` entry point. This is required because the sovereign upgrade handler registers `HOOK_TYPE_BEFORE_DELEGATION_SHARES_MODIFIED` and `HOOK_TYPE_BEFORE_DELEGATION_REMOVED` in `app/upgrades/sovereign/upgrades.go:167-168`, indicating that the `x/harpoon` module will invoke the `sudo` entry point when these hooks are triggered. The contract can also potentially be subscribed to the `AFTER_UNBONDING_INITIATED` hook.

Consequently, staking actions that call the `BeforeDelegationSharesModified`, `AfterUnbondingInitiated`, and `BeforeDelegationRemoved` hooks will always fail, leading to a denial of service. For example, users will be unable to delegate to an existing validator or perform an undelegation.

Additionally, a chain halt may occur if redelegations are slashed when a validator misbehaves, as the `Unbond` function will call the unimplemented `BeforeDelegationSharesModified` hook.

Recommendation

We recommend adding a `SudoMsg` variant for each potential hook type to which the contract could be subscribed.

Even though the `AFTER_UNBONDING_INITIATED` hook is not registered in the sovereign upgrade handler for the `neutron-staking-tracker` contract, we still recommend handling all possible hooks. Any unnecessary hook should simply return `Ok(Response::new())` to prevent unintended transaction failures if it is mistakenly registered.

Status: Acknowledged

The client states that the staking tracker contract is a crucial part of the Neutron, and returning success in case some hook is not implemented can lead to an internal inconsistency of the contract when, for example, they accidentally deployed a contract without a hook implementation, but they have a hook registered in the module, and they expect it to be called. In that case, the client wants all relevant transactions to fail (and even the chain to be halted) because there won't be any inconsistency in the contract, which can lead to even worse consequences.

2. Failure to handle BeforeDelegationRemoved hook leads to stake inconsistency

Severity: Major

In contracts/dao/voting/neutron-staking-tracker/src/contract.rs, a delegator's shares are only updated within the AfterDelegationModified handler.

However, when unbonding or redelegating, the AfterDelegationModified hook is only triggered if the amount of shares remaining with the validator is non-zero. If the delegator fully unbonds or redelegates their entire stake, the BeforeDelegationRemoved hook is triggered instead within Keeper.RemoveDelegation.

Since the contract does not implement the BeforeDelegationRemoved hook, delegations are not properly removed from the contract's state when a validator's delegation reaches zero. Consequently, the contract may show incorrect delegation records.

For example, if a delegator has 1_000 shares with ValA and redelegates all of them to ValB, the tracker will incorrectly display 1_000 shares with both ValA and ValB, leading to inconsistencies between the contract's delegation state and the actual x/staking state.

Recommendation

We recommend implementing the BeforeDelegationRemoved hook to correctly remove the user's delegation entry from the DELEGATIONS state and update the VALIDATORS state accordingly.

Status: Resolved

3. Incorrect parameter passed during SudoMsg::AfterValidatorRemoved

Severity: Major

In contracts/dao/voting/neutron-staking-tracker/src/contract.rs:203-206, the after_validator_removed function is called with cons_addr as the third parameter and val_addr as the fourth parameter.

The issue is that the after_validator_removed function's logic expects the parameters to be supplied in a different order. Specifically, the validator's operator address (val_addr parameter) should be passed as the third parameter, and the consensus address (cons_addr parameter) should be passed as the fourth parameter, as seen in contracts/dao/voting/neutron-staking-tracker/src/contract.rs:586-587.

Consequently, the SudoMsg::AfterValidatorRemoved hook will always fail because the consensus address is incorrectly utilized as the validator's operator address, leading to a denial of service issue.

Recommendation

We recommend ensuring the parameters are passed in the correct order when calling the after_validator_removed function.

Status: Resolved

4. Potential incorrect delegation logic in StakeWithDrop

Severity: Major

In app/upgrades/sovereign/drop.go:17-65, the delegation logic in StakeWithDrop contains two key issues:

1. Incomplete delegation when DropDelegate fails:

    - If DropDelegate fails (partially or fully), the fallback mechanism in app/upgrades/sovereign/drop.go:56-62 only delegates the remaining amount (daoDelegateAmount - daoDelegateAmount / 2) using native staking.

    - This means only half of the initial balance is ultimately delegated instead of the full amount.

2. Inefficient delegation even when DropDelegate succeeds:

    - In app/upgrades/sovereign/drop.go:28, the DropDelegate function attempts to delegate 50% of the daoDelegateAmount.

    - Later, the function checks whether DropDelegate has completed the entire delegation using GetAllDelegatorDelegations(ctx, dropAddress). However, since we only call DropDelegate with 50% of the amount, the total retrieved delegation can never exceed 50%.

    - Consequently, the condition at line 50 fails, and the remaining 50% is always delegated using native staking.

    - This results in an inefficient delegation strategy, as half the amount will always bypass DropDelegate, potentially impacting staking expectations.

Recommendation

We recommend applying the following recommendations:

1. Modify the fallback mechanism to delegate the entire remaining balance in `MainDAOContractAddress` when `DropDelegate` fails, ensuring that the expected amount is fully staked.

2. Ensure that a successful execution of `DropDelegate` results in full delegation via `DropDelegate`, avoiding unnecessary reliance on native delegation.

Status: Resolved

5. Bypassing the embedded filesystem causes file I/O errors on non-build machines, causing a denial of service

Severity: Major

In `app/upgrades/sovereign/deics.go:39-40`, the `validators/staking` path is embedded in the binary via the `Vals` (`embed.FS`) object.

However, the `GatherStakingMsgs` function in line `57` incorrectly calls `os.ReadFile`, which looks for the file on the local filesystem instead of using the embedded filesystem. This is problematic because, in production, where the binary is built on one machine and executed on another, the `neutrond` binary will raise file I/O errors due to the missing physical files.

Consequently, validators will be unable to start the `neutrond` binary, disrupting the migration process.

Recommendation

We recommend replacing `os.ReadFile` with `Vals.ReadFile` in line `57` to read from the embedded filesystem.

Status: Resolved

6. Blacklist feature does not account for block height, causing incorrect vote computation

Severity: Major

In `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:788-803`, the `query_voting_power_at_height` function relies on the `BLACKLISTED_ADDRESSES` map to exclude blacklisted addresses from voting power calculations. When a proposal is created, the total voting power on the current block height is recorded. A blacklisted user will have zero voting power, preventing them from voting on the proposal.

The issue is that BLACKLISTED_ADDRESSES is stored as a Map instead of a SnapshotMap, meaning that modifications to the blacklist state affect voting power calculations. If the contract owner blacklists a new user or removes a blacklist for an existing user, the total voting power will differ from the recorded power in the proposal. This leads to situations where the governance voting process becomes unreliable, potentially exceeding 100% of total voting power if an address is removed from the blacklist and votes on a past proposal.

For example, if a user is removed from the blacklist state after the proposal creation, the actual total voting power will be larger than the proposal's recorded total voting power. When the user votes on the proposal, the tally logic will be computed incorrectly, as their voting power is not included in the proposal's recorded total voting power. This may cause malicious proposals to pass incorrectly, allowing attackers to execute arbitrary messages to steal funds from the protocol.

A similar issue exists in contracts/dao/voting/neutron-staking-tracker/src/contract.rs:805-834, where query_total_power_at_height recalculates total voting power using the current state of BLACKLISTED_ADDRESSES, meaning changes to the blacklist can retroactively alter past total voting power calculations.

Recommendation

We recommend modifying BLACKLISTED_ADDRESSES to use the SnapshotMap storage design to record the block height when an address is blacklisted. This ensures that historical voting power queries always return values consistent with the blacklist status at the queried height.

Status: Resolved

7. Unbounded validator iteration causes potential out-of-gas errors

Severity: Major

In contracts/dao/voting/neutron-staking-tracker/src/contract.rs:757, the calculate_voting_power function iterates over all VALIDATORS.keys.

This is problematic because loading all validators from storage incurs a significant cost, as historical validators are stored indefinitely. Additionally, unbonding and unbonded validators are also included during the iteration, increasing the required gas costs.

Consequently, the query may fail due to an out-of-gas error, causing a denial of service issue. This affects the governance as the query is used to compute the total voting power in the current block height. An attacker can increase the iteration costs by creating many low-stake validators with different addresses to cause a denial of service in the DAO governance, preventing the protocol from working as intended.

Similarly, this issue also affects the `query_total_power_at_height` function in `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:805`.

Recommendation

We recommend applying the following recommendations:

- Separate `VALIDATORS` into `BONDED_VALIDATORS` and `UNBONDED_VALIDATORS` states and update them based on the hooks logic. The `BONDED_VALIDATORS` iteration will be capped by the `maxValidators` limit in the `x/staking` module.
- Use `SnapshotMap::remove` with height tracking to clear unbonded and unbonding validators from hot-path queries while preserving historical data.
- Modify the `calculate_voting_power` and `query_total_power_at_height` functions to only iterate over `BONDED_VALIDATORS` instead of all validators.
- For the `query_total_power_at_height` function, ensure that `BONDED_VALIDATORS` is iterated over exactly once. Within each iteration, iterate over a pre-loaded list of blacklisted addresses, calculating their voting power 'in-line' per validator and subtract this from the total voting power.

Status: Resolved

8. Rounding discrepancy in `before_validator_slashed` leads to misestimation of slashed tokens

Severity: Minor

In `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:371-395`, the `before_validator_slashed` function calculates slashed tokens using the `to_uint_ceil` function, which rounds up the value. However, this behavior is different from the SDK slashing logic as it rounds down the slash amount. This difference causes a mismatch between the contract's state and the actual chain state, leading to an overestimation of slashed tokens in the contract.

Specifically, the SDK's slashing logic determines the `slashAmount` using the `TruncateInt` function, causing the slash amount to be rounded down. After that, the `effectiveFraction` variable, which represents the percentage of slashed tokens divided by the validators tokens, is rounded up using the `QuoRoundUp` function before being passed to the `BeforeValidatorSlashed` hook.

The issue is that the `before_validator_slashed` function rounds up the slashing percentage using the `to_uint_ceil` function, causing the computed slashed amount to be slightly larger than the actual slashed amount.

Consequently, the [tokensToBurn variable]() will differ from the slashed_tokens_uint128 variable, violating an important invariant that enforces accurate tracking of the validator's tokens (recorded in the validator.total_tokens field). While the difference is usually minimal (one untrn in most cases), the amount could accumulate over time and become significant, leading to inconsistencies between recorded and actual voting power.

Additionally, since the hook only passes the effectiveFraction variable to the before_validator_slashed function, the actual slash amount cannot be reversed computed as the percentage value has already been rounded up.

Please refer to the proof of concept in the [Appendix]() to reproduce this issue.

Note that a mitigating factor to this issue is that whenever a delegator stakes additional tokens to an existing validator, the validator's total_tokens and total_shares fields are updated with the latest values from x/staking, as implemented in after_delegation_modified (contracts/dao/voting/neutron-staking-tracker/src/contract.rs:512-538).

Recommendation

We recommend modifying the SDK to update the BeforeValidatorSlashed hook to include the [tokensToBurn variable](). This would enable the before_validator_slashed function to compute the slash amount correctly.

Alternatively, if modifying the SDK is not feasible, we recommend setting the total_tokens and total_shares fields to zero in the AfterValidatorRemoved hook as an additional mitigation measure.

Status: Resolved

9. New validator accounts are not pre-funded, potentially causing upgrade failure

Severity: Minor

In app/upgrades/sovereign/deics.go:210-211, the CreateValidator function is called within the loop processing newValMsgs.

However, unlike ICS validators, these new validator accounts are not automatically funded from the MainDAO contract address. Since new validator accounts are not pre-funded, CreateValidator calls may fail due to insufficient balance.

Consequently, this could result in validator creation failures, leading to a failed network upgrade.

Recommendation

We recommend ensuring that all new validator accounts have sufficient funds before calling `CreateValidator`. This can be accomplished by pre-funding them manually before the upgrade or implementing an automated mechanism to fund these accounts from a designated source, such as the `MainDAO` contract.

Additionally, consider performing pre-upgrade checks to verify validator balances in order to prevent upgrade failures.

Status: Resolved

10. Failure to check for contract existence in hook subscriptions may result in a chain halt or denial of service

Severity: Minor

A `HookSubscription` in the `x/harpoon` module tracks contract addresses to be executed when the associated `x/staking` module hook is triggered.

However, there is no verification that the contracts exist at those addresses, neither at genesis (see `x/harpoon/types/genesis.go:18-31`) nor in the `ManageHookSubscription` message handler (see `x/harpoon/keeper/msg_server.go:28-34`).

Consequently, calling a non-existent contract causes an error to be returned, and the chain may halt or experience a denial of service for associated staking transactions, depending on whether the hook is triggered within an `EndBlocker`.

We classify this issue as minor since the DAO can only add new contract addresses, indicating that the governance would perform rigorous manual reviews on message parameters before voting on the proposal.

Recommendation

As it may be useful to add non-existent contracts at genesis (e.g., in a network setup script), we recommend verifying contract existence in the `doCallSudoForSubscriptionType` function (`x/harpoon/keeper/keeper.go:193`) before calling `wasmKeeper.Sudo` at line `200`, skipping any non-existent contracts. Additionally, consider using `wasmKeeper.HasContractInfo` to determine whether a contract exists.

Status: Resolved

11. Missing validations in the x/harpoon module

Severity: Minor

In x/harpoon, the GenesisState and SubscribedContracts query does not perform the following validations:

- **Uniqueness by HookType**: in x/harpoon/genesis.go:13, if multiple HookSubscription entries share the same HookType, the last entry overwrites earlier entries.

- **Unique contract addresses**: duplicate addresses within a single HookSubscription trigger multiple executions of the same contract per hook, potentially causing invalid voting power calculations or denial-of-service if the contract returns an error.

- **Invalid hooks in GenesisState**: any HookSubscription with a HookType equal to HOOK_TYPE_UNSPECIFIED should be rejected during the GenesisState validation.

- **Invalid hooks during SubscribedContracts query**: in x/harpoon/keeper/query_server.go:27, the SubscribedContracts query does not validate whether the HookType is valid. This is possible because the value is not guaranteed by the protocol buffers. If an invalid HookType is used, an empty list of contract addresses (see x/harpoon/keeper/keeper.go:148) will be returned instead of an informative error message.

Recommendation

We recommend applying the following recommendations to ensure consistent HookSubscription validation between the GenesisState, SubscribedContracts query, and MsgManageHookSubscription handling:

- Move the MsgManageHookSubscription.checkHooksUnique logic in x/harpoon/types/tx.go:55 to a standalone function in x/harpoon/types/hooks.go. It can then be used from both GenesisState.Validate and MsgManageHookSubscription.Validate functions.

- Add a function in x/harpoon/types/genesis.go that validates duplicate addresses and returns potential errors from GenesisState.Validate. Note that MsgManageHookSubscription handling already prevents duplicates by skipping existing addresses in x/harpoon/keeper/keeper.go:107.

- Move the MsgManageHookSubscription.checkHooksExist logic in x/harpoon/types/tx.go:55 to a standalone function in x/harpoon/types/hooks.go.

This allows the function to be used for `GenesisState.Validate` in `x/harpoon/types/genesis.go:20`, `SubscribedContracts` query handler and the `MsgManageHookSubscription.Validate` function to ensure correct and consistent `HookType` validation.

Status: Resolved

12. Risk of chain takeover if staking levels are insufficient

Severity: Minor

The upgrade process in `app/upgrades/sovereign/upgrades.go:36-94` transitions the chain from using ICS validators to a sovereign validator set. By the time of writing, `MainDAOContractAddress` currently holds approximately 372M NTRN, which is expected to be used for staking through `StakeWithDrop` in line `81`. This will eventually call `DropDelegate` in `app/upgrades/sovereign/drop.go:28` and `52`.

However, the `SetupRewards` function in `app/upgrades/sovereign/upgrades.go:58-61` is currently not implemented, so it is uncertain how much of this balance will actually be staked.

Several factors contribute to this risk:

- `SetupRewards` may allocate a portion of `MainDAOContractAddress` funds for rewards, reducing the amount available for staking.
- The `DropDelegate` contract is a black box, making it unclear how effectively it will distribute stake.
- As noted in a [separate issue](), if `DropDelegate` fails, only half of the intended delegation is currently staked.

If an insufficient amount is staked during the upgrade, a malicious actor could create a validator at *upgrade height + 1* and stake more than 33% of the total voting power, enabling censorship or denial-of-service attacks. At 67%, they could gain full control over the blockchain.

Recommendation

We recommend determining the amount of rewards allocated via `SetupRewards` carefully to ensure that a sufficient portion of the `MainDAOContractAddress` balance is staked during the upgrade to prevent a potential takeover scenario. Additionally, the `DropDelegate` contract should distribute stake only to the new sovereign validators and not to ICS validators.

Status: Acknowledged

The client states that they acknowledge the issue, and the mainnet parameters will be carefully determined.

13. Unrestricted contract hooks subscription may cause out-of-gas error

Severity: Minor

In `x/harpoon/keeper/keeper.go:193-204`, the `doCallSudoForSubscriptionType` function dispatches sudo messages to the contracts registered for the specific hook.

The issue is that when registering the contracts in `MsgManageHookSubscription`, there is no limit to how many contracts can be registered. If there are too many contracts registered, the transaction may fail due to an out-of-gas error, causing a denial of service issue.

We classify this issue as minor because the `MsgManageHookSubscription` message can only be called by the authority address, which is a privileged address. Additionally, in case the above situation occurs, the authority can recover an out-of-gas situation by unregistering the contracts.

Recommendation

We recommend enforcing a limit on how many contracts can be registered for a specific hook.

Status: Acknowledged

The client states that they will fix this in future releases, but for now they keep the current implementation, since addition of new hooks subscription is a governance gated operation.

14. Failure to handle error from `SendCoinsFromModuleToModule` allows execution to continue despite transfer failure

Severity: Minor

In `app/upgrades/sovereign/deics.go:171-172`, the result of `SendCoinsFromModuleToModule` is assigned to the `err` variable, but it is neither checked nor returned.

This means that if the function fails to transfer ICS staked funds from `NotBondedPoolName` to `BondedPoolName`, no errors will be returned, and the migration will continue as if the transfer was successful. Subsequent logic that assumes the funds were properly moved could then behave unexpectedly.

Recommendation

We recommend modifying the `MoveICSToStaking` function to return the result of `bk.SendCoinsFromModuleToModule`. This ensures that any failure in transferring funds is propagated properly and prevents execution from continuing under incorrect assumptions.

Status: Resolved

15. Suboptimal staking parameters affect IBC reliability, validator limits, and storage efficiency

Severity: Minor

In `app/upgrades/sovereign/deics.go:184-195`, the staking parameters defined may affect network performance and security due to the following reasons:

- **Short `HistoricalEntries` value**: the `HistoricalEntries` parameter is critical for IBC handshakes, as it determines how long historical entries are available. With Neutron's block time of approximately 2s, the current value of `100` results in a retention period of less than 4 minutes. This may be insufficient for high-latency IBC handshakes, potentially preventing IBC connections from being established.

- **Temporary `MaxValidators` setting**: during the upgrade, `MaxValidators` is set to `len(consumerValidators) + len(newValMsgs)` to prevent panics. However, this is likely not the intended long-term value for Neutron. If not reduced post-upgrade, Neutron may need to compensate an excessive number of validators under the upcoming `x/revenue` model.

- **Excessive `MaxEntries`**: the `MaxEntries` parameter defines the maximum number of unbonding or redelegation entries allowed. For unbondings, it sets the limit per account-validator pair, while for redelegations, it applies per account and source/destination validator trio. While the intended value of `100` provides users with greater flexibility, it is significantly higher than the Cosmos standard of `7` and increases storage requirements. A malicious actor could exploit this by spamming the system with numerous small unbonding or redelegation requests, leading to unnecessary state growth.

Recommendation

We recommend applying the following recommendations:

- Update the `HistoricalEntries` parameter to a higher value, such as `1000`, to ensure reliable IBC handshakes.
- After the upgrade, Neutron should execute a governance proposal to reduce `MaxValidators` to its intended long-term value.

- `MaxEntries` should be lowered to a reasonable range to balance flexibility for users and mitigate unnecessary state growth.

Status: Resolved

The client states that after the migration, they will create a proposal to reduce the `MaxValidators` value.

16. Inconsistent type usage and redundant parameters in `findHooksToRemove`

Severity: Informational

In `x/harpoon/keeper/keeper.go:210-228`, the `findHooksToRemove` function unnecessarily mixes `allHooks` as `[]int32` and `hooksToAdd` as `[]types.HookType`, despite `types.HookType` is an alias for `int32`.

Moreover, `allHooks` is only used within this function and does not need to be passed as a parameter. The function can instead iterate over `types.HookType_name` to determine which hooks should be removed.

Recommendation

We recommend modifying the function to iterate over `types.HookType_name` instead of accepting `allHooks` as a parameter. This simplifies the function by ensuring consistent type usage and removing unnecessary type conversions, making the code easier to understand and maintain.

Status: Acknowledged

17. Inconsistent usage of `cons_address`

Severity: Informational

In `contracts/dao/voting/neutron-staking-tracker/src/state.rs:58`, the `Validator` struct includes a `cons_address` field that is initially set to an empty string in the `after_validator_created` handler, as seen in `contracts/dao/voting/neutron-staking -tracker/src/contract.rs:647`.

The issue is that it is unused, and there are leftover functions that support it but are not implemented:

- This field remains empty unless the `after_validator_begin_unbonding` function is executed, which is set on line `450`.

- There is an unused and untested `get_consensus_address` function in lines `715-747` and a misleading comment in line `621`. This suggests the original intent was to derive a `cons_address` value within the `after_validator_created` handler from the Validator's queryable consensus public key.

- In the unit tests phase (`contracts/dao/voting/neutron-staking-tracker/src/testing/tests.rs`), mock consensus addresses are used as keys for the `VALIDATOR SnapshotMap` in lines `354`, `371`, `440`, `448`, and `1035`. This is inconsistent with the contract code under test, which uses the 'valoper' address for the key.

- There are misleading references in the documentation to an "*Operator-to-Consensus Mapping*" in `contracts/dao/voting/neutron-staking-tracker/README:18`, an incorrect statement that the `VALIDATORS` storage map is "*indexed by* *`valcons_address`*" in line `39`, and a reference to a non-existent `OPERATOR_TO_CONSENSUS` state in line `41`.

Recommendation

We recommend applying the following recommendations:

- Make the unit tests consistent by always using the mock 'valoper' addresses as keys for the `VALIDATOR` storage setup.
- Ensure that the `neutron-staking-tracker` contract documentation accurately describes how the validator consensus addresses are used and stored.
- Use the `get_consensus_address` function within the `after_validator_created` handler to derive the correct value for the `cons_address` field before the `Validator` is entered into storage. Ensure that the `get_consensus_address` function is properly tested and that the comment in line `621` is moved to the corresponding call site.

Alternatively, if the `cons_address` field is not needed anymore:

- Removing the `cons_address` field from the `Validator` definition in `contracts/dao/voting/neutron-staking-tracker/src/state.rs:58`.
- Remove the unused `get_consensus_address` function, and the misleading comment in line `621`, and ignore the `cons_address` field in all `SudoMsg` variants.

Status: Acknowledged

18. Code quality improvements

Severity: Informational

The following instances represent dead code, redundant storage, and unnecessary address cloning found in the codebase:

- In contracts/dao/voting/neutron-staking-tracker/src/contract.rs: 95-101 and 143-150, the execute_bond and execute_unbond functions are never called, and no corresponding ExecuteMsg variants exist.

- The query handlers query_dao, query_name, query_description, and query_list_bonders, defined in lines 836-857, are also never called, and there are no corresponding QueryMsg variants.

- The DAO storage Item defined in contracts/dao/voting/neutron-staking-tracker/src/state.rs:168 is written to during the contract instantiate handler in line 51, but only used by the 'dead' query_dao handler mentioned above.

- The Config definition in contracts/dao/voting/neutron-staking-tracker /src/state.rs:15-21 contains fields that are never read or are only accessed by these 'dead' query handlers, namely: denom, name, and description. Redundant fields cause additional serialization/deserialization and read/write overhead.

- In lines 59 and 111-112, the Validator and Delegation definitions include fields that also serve as storage keys. This is redundant because retrieving those entries already requires knowing the keys. This increases serialization/deserialization and read/write costs.

- Clone::clone is called in contracts/dao/voting/neutron-staking-tracker/src/contract.rs: 444, 483, and 600 when it is not required, resulting in unnecessary heap allocations and string copies.

- Lastly, the code frequently employs greedily-evaluated Option::ok_or or Option::unwrap_or calls, leading to unnecessary heap allocations and string copies in the 'happy' path.

Recommendation

We recommend applying the following recommendations:

- Remove the 'dead' code in contracts/dao/voting/neutron-staking-tracker/src/contract.rs: 95-101, 143-143, 836-857 and contracts/dao/voting/neutron-staking-tracker/src/state.rs:168.
- Remove the redundant fields defined in stored structs in lines 16-17, 19, 59, and 111-112.
- Remove the call to Clone::clone in contracts/dao/voting/neutron-staking-tracker/src/contract.rs:444, 483 and 600.

- Use the corresponding `Option::ok_or_else` or `Option::unwrap_or_else` to employ lazy evaluation that will only occur in the 'non-happy' path in lines `247`, `267`, `309`, `436`, `592`, `727`, `731`, `875` and `887`.
- The call to `Option::unwrap_or` in line `497` can be simplified to avoid cloning by replacing it with `map_or_else(Uint128::zero, |d| d.shares).`

Status: Acknowledged

19. Misleading instances in the `neutron-staking-vault` contract

Severity: Informational

The `neutron-staking-vault` contract stores the address of the `neutron-staking-tracker` contract in order to delegate voting power query requests.

However, there are several instances where incorrect binding, function, and contract names are used:

- In `contracts/dao/voting/neutron-staking-vault/src/contract.rs`:`64`, `110`, and `128`, the binding name given to this address value is incorrectly set to `vesting_contract_address`.

- Similarly, in lines `202` and `225`, where the voting power query request delegations occur, the binding names given to the voting power values are confusingly `unclaimed_amount` and `unclaimed_amount_total`.

- In the unit tests for the `neutron-staking-vault` contract, a test setup function for instantiating a mock `neutron-staking-tracker` contract is given the misleading name `instantiate_vesting_contract` (`contracts/dao/voting/neutron-staking -vault/src/tests.rs:63`).

- Finally, the `CONTRACT_NAME` constant defined in `contracts/dao/voting /neutron-staking-vault/src/contract.rs:15` and used by the `cw2::set_contract_version` function is given an incorrect value of "`crates.io:neutron-investors-vesting-vault`".

Recommendation

We recommend applying the following recommendations:

- Modify the name of the binding in lines `64`, `110`, and `128` to `staking_tracker_addresss`.
- Modify the name of the bindings in lines `202` and `225` to reflect that they are voting power values.
- Modify the name of the function defined in `contracts/dao/voting/neutron-staking-vault/src/tests.rs:63` to `instantiate_staking_tracker`.

- Modify the value of the `CONTRACT_NAME` constant defined in `contracts/dao/voting/neutron-staking-vault/src/contract.rs:15` to “`crates.io:neutron-staking-vault`”.

Status: Acknowledged

20. Misuse of the `Addr` type in the `neutron-staking-tracker` contract

Severity: Informational

The `neutron-staking-tracker` contract uses the `&Addr` type to represent ‘valoper’ addresses in both keys for the `VALIDATORS` and `DELEGATIONS` storage maps in `contracts/dao/voting/neutron-staking-vault/src/state.rs:136` and `150`.

The issue is that this is an anti-pattern as the ‘valoper’ addresses cannot be validated by the `Api::addr_validate` function due to having a different bech32 prefix to the EOA and contract addresses. The requirement to use `Addr::unchecked` when creating the key types adds complexity when reading the code.

Additionally, the `Addr` type is also used in the `QueryMsg::VotingPowerAtHeight` variant in `contracts/dao/voting/neutron-staking-vault/src/msg.rs:94`. This is an anti-pattern because deserializing an `Addr` does not perform address validation.

Recommendation

We recommend applying the following recommendations:

- Implement the `&str` type for ‘valoper’ keys in the `VALIDATORS` and `DELEGATIONS` storage definitions.
- Implement the `String` type for the `address` field of the `QueryMsg::VotingPowerAtHeight` variant in `contracts/dao/voting/neutron-staking-vault/src/msg.rs:94`.

Status: Acknowledged

21. Misleading comments in the `neutron-staking-tracker` contract

Severity: Informational

The main purpose of the `neutron-staking-tracker` contract is to act as a utility contract that tracks the historical voting power of validators and delegators.

However, the comments in `contracts/dao/voting/neutron-staking-vault/src/state.rs:9-13` refer to the contract as a "*vault*" and to a "*token denomination used for delegations and governance*" which misleads readers of the codebase.

Additionally, the comment in line `159` implies that the boolean value stored for each blacklisted address designates whether the address is blacklisted. This is misleading because it is the inclusion of the address as a key in the `BLACKLISTED_ADDRESSES`, regardless of stored value, which is used to determine the blacklisted status.

Recommendation

We recommend modifying the comments in `contracts/dao/voting/neutron-staking-vault/src/state.rs:9-13` and `159` to accurately reflect the contracts' purpose and blacklist status logic.

Status: Acknowledged

22. Contracts should implement a two-step ownership transfer

Severity: Informational

The `neutron-staking-tracker` and `neutron-staking-vault` contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer, as seen in `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:166` and `contracts/dao/voting/ neutron-staking-vault/src/contract.rs:118`.

While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

The client states that they find this mechanism complicated and non-usable in their case since a change of owner is quite a rare event. Each transaction is being carefully reviewed, and in their case, the owner should not be changed since all contracts will be owned by the Main DAO itself.

23. Lack of context in comments explaining manual validator set management

Severity: Informational

The comment in app/upgrades/sovereign/deics.go:160-162 attempts to explain the critical mechanism whereby ICS validators are included in the active validator set for the next block while also ensuring their transition to the Unbonding state.

However, the comment lacks context to help future maintainers understand this mechanism and uphold the invariants.

Recommendation

We recommend re-working the comment to include the following context:

- Each ICS Validator has an initial stake of 1untrn and a LastValidatorPower explicitly set to 1. It is then manually bonded by the call to bondValidator, which sets its ValidatorByPowerIndex to 0 due to the token amount being divided by DefaultPowerReduction, which is the integer 1_000_000.

- After the sovereign upgrade handler is executed as part of the x/upgrade module EndBlocker, the x/staking module's EndBlocker will execute. This order is enforced by the call to SetOrderEndBlockers in app/app.go:1032.

- In the x/staking module's EndBlocker, the ApplyAndReturnValidatorSetUpdates function in x/staking/keeper/val_set_change.go:130-270 is called, which transitions validators with a LastValidatorPower storage entry and a current 'Consensus Power' of 0 to the Unbonding state, while also including them in the returned ValidatorUpdate list.

- The returned ValidatorUpdate list is ultimately used to inform CometBFT of the current validator set.

Status: Resolved

24. Redundant code within the sovereign upgrade handler and AnteHandler setup options

Severity: Informational

The final statement in the DeICS function app/upgrades/sovereign/deics.go:217 calls SetLastTotalPower with a nominal value of 1 and returns any resulting error from the DeICS function. The LastTotalPower storage entry will be overwritten with the correct value in the execution of the ApplyAndReturnValidatorSetUpdates function in x/staking/keeper/val_set_change.go:259.

Additionally, the HandlerOptions struct defined in app/ante_handler.go:24-35 contains the unused field ConsumerKeeper. Redundant fields are discouraged as they may be misleading and increase maintenance costs.

Recommendation

We recommend removing the call to SetLastTotalPower in app/upgrades/sovereign/deics.go:21 and instead return nil from the DeICS function. Additionally, consider removing the unused Consumer field in app/ante_handler.go:30.

Status: Resolved

25. Ineffective unit test for the sovereign upgrade handler

Severity: Informational

In app/upgrades/sovereign/upgrades_test.go:35-67, the TestUpgrade function first creates a mock App in line 36 before executing the DeICS function in line 48. In line 52, it asserts that no errors are returned and that the number of validators registered with the app.StakingKeeper is greater than 0 in line 53.

However, the mock App returned from the suite.GetNeutronZoneApp(suite.ChainA) contains no ICS validators and already has 4 validators registered with the app.StakingKeeper. This greatly reduces coverage of the DeICS function where errors could be returned and negates the assertion in line 53.

Consequently, ineffective tests can provide a false sense of security and prevent regressions from being discovered during future development.

Recommendation

We recommend ensuring that ICS validators are added to the app.ConsumerKeeper before the call to DeICS and to instead assert the number of validators registered with app.StakingKeeper grew by the expected amount.

Status: Resolved

26. Usage of magic numbers decreases maintainability

Severity: Informational

In app/upgrades/sovereign/deics.go:98, 115, 144, 170, and 194 use magic numbers that indicate the bond denom value.

Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Recommendation

We recommend replacing the above instances with the DefaultDenom variable defined in app/params/denom.go:4.

Status: Resolved

27. Lack of guaranteed IBC client updates during the upgrade may cause IBC transaction failures

Severity: Informational

When a chain undergoes an upgrade that changes more than one-third of its validator set, IBC light clients on counterparty chains must be updated accordingly to ensure continued trust in the new validator set.

At height N+1, where N is the upgrade height, the chain still operates with the old validator set but includes a “next validator set hash” referencing the upcoming set. The IBC client update at N+2 will then invoke the VerifyAdjacent function (https://github.com/cometbft/cometbft/blob/v0.38.15/light/verifier.go#L93), allowing validation without requiring one-third of the trusted validator set to sign. This process ensures a smooth transition for IBC connections post-upgrade.

While relayer software (such as hermes) may automatically update clients at heights N+1 and N+2, this behavior is not guaranteed, as it depends on the configuration set by relaying operators. If the IBC light clients are not properly updated, IBC messages may fail to transfer or be acknowledged between chains, leading to transaction delays or failures.

Recommendation

Ahead of the upgrade, we recommend that Neutron coordinates with IBC relaying operators for the chains it connects to (such as Cosmos Hub, Osmosis, and Stride, etc.) to inform them of the required client updates. Specifically, IBC clients on counterparty chains should be updated at mandatory heights: N+1 and then N+2, where N is the upgrade height.

Status: Resolved

The client states that they have their own relayers and will notify all external relayer operators.

28. Hardcoded contract addresses may cause deployment misconfigurations

Severity: Informational

In `app/upgrades/sovereign/constants.go:18-22`, multiple contract addresses are hardcoded, including staking, DAO, and voting registry contracts. The current deployment setup is manual, requiring developers to update these addresses manually.

Manually assigning contract addresses increases the risk of human error, which can result in misconfiguration. Consequently, this can lead to deployment inconsistencies

Recommendation

We recommend applying the following recommendations:

- Implement an automated deployment script to dynamically fetch and assign contract addresses.
- Store addresses in a configuration file or environment variables to ensure consistency across deployments.
- Implement validation checks to prevent incorrect address usage.

Status: Acknowledged

29. Inconsistent documentation regarding missing validator blacklist in the `AfterDelegationModified` hook

Severity: Informational

At the time of writing, the documentation in the Notion section https://www.notion.so/hadron/Staking-vault-overview-16885d6b9b10809494e2dd247c11581b#19385d6b9b1080989806ca87854c44f4 states that the `AfterDelegationModified` hook adjusts the validator's total stake in the `VALIDATORS` state unless the validator is blacklisted. However, the logic in `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:462-581` does not implement any validator blacklist mechanism.

If a validator blacklist is intended, its absence in the code means that all validators' stakes are adjusted without exception, contradicting the documentation.

Recommendation

We recommend either updating the documentation to reflect the actual implementation or modifying the code to introduce a validator blacklist in accordance with the documented behavior.

Status: Resolved

30. Potential confusion during shares-to-tokens comparison

Severity: Informational

In `app/upgrades/sovereign/drop.go:44-50`, the `delegatedByDrop` variable accumulates delegation shares. After that, it is compared to `daoDelegateAmount.Balance.Amount` in line `50`, which represents the token amount. Since shares and tokens have a conversion rate that depends on the validator's prior slashes, this comparison appears incorrect at first glance.

However, in this specific case, validators are newly created in the same block, which means they have not been slashed yet. This ensures a 1:1 exchange rate between shares and tokens at this stage, making the direct comparison valid.

Despite this, the lack of a clarifying comment can confuse developers who review or modify the code in the future.

Recommendation

We recommend adding a comment explaining that shares and tokens are equal at this stage because the validators have been newly created and have not been slashed.

Status: Resolved

Detailed Findings for Phase 2

31. Validators receive lesser rewards due to incorrect reward denom

Severity: Critical

In x/revenue/keeper/twap.go:30-48, the UpdateRewardAssetPrice function queries the price for Neutron denom in USD value from Slinky and stores it as a RewardAssetPrice in CalcNewRewardAssetPrice. The price is used to compute the base revenue for validators in x/revenue/keeper/keeper.go:219.

The issue is that the price queried from Slinky uses the NTRN denom in x/revenue/keeper/twap.go:17. This is incorrect because the reward denom is denominated as untrn in x/revenue/types/constants.go:9, which is the base unit. This causes the recorded price to inflate by 10^6.

Consequently, the computed base revenue amount will be less than intended, resulting in validators receiving fewer rewards.

Recommendation

We recommend modifying the UpdateRewardAssetPrice function to convert the price from NTRN to untrn before storing it in the CalcNewRewardAssetPrice function. This ensures that the compensation aligns with the expected RewardDenom, providing accurate rewards to validators.

Status: Resolved

32. Validator rewards are lost if the payment schedule is updated

Severity: Major

In x/revenue/preblock.go:114-126, the PaymentScheduleCheck function resets the payment period (ps.StartNewPeriod(ctx)) if the payment schedule is updated. The issue is that validators are not compensated for periods that have elapsed since the start of the previous payment schedule (determined by ps.TotalBlocksInPeriod(ctx)).

Consequently, if the authority updates the payment schedule, validators will not receive any reward for the start period of the previous payment schedule to the latest block height, causing a shortfall for validators.

Recommendation

We recommend accruing rewards based on the elapsed time before resetting the payment schedule.

Additionally, the `BaseCompensation` should be scaled accordingly to reflect the reduced period length. For example, if the payment schedule is updated when the previous period is 10% completed, only 10% of the `BaseCompensation` should be distributed per validator.

Status: Resolved

33. Failure to update user stake may cause excess reward distribution

Severity: Major

When a staking action is executed, the `neutron-staking-tracker` contract dispatches `sudo` hooks to update the user stake amount (`UpdateStake` message) or record a slashing event (`Slashing` message).

Specifically, these messages are dispatched as `SubMsg::reply_on_error` (see `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:855` and `875`). If an error occurs, the transaction will not roll back, as seen in `contracts/dao/voting/neutron-staking-tracker/src/contract.rs:827-830`.

The issue is that not reverting on errors indicates that `UpdateStake` messages may fail to update user shares correctly, leading to incorrect reward distribution. For example, the `SudoMsg::AfterDelegationModified` message should decrease the user's shares after they have undelegated funds. If an error occurs in the `update_stake` function, the `neutron-staking-tracker` contract will ignore it, allowing users to continue earning rewards with their old stake amount and receive more rewards than intended.

Recommendation

We recommend automatically pausing the `neutron-staking-rewards` contract in case an error occurs when updating the user stake amount. Once the contract is paused, a manual investigation should be carried out to identify the root cause of the error and implement a custom migration to update the user stake amount correctly.

Status: Resolved

34. Incorrect stake retrieval for LST protocols leads to zero rewards

Severity: Major

In `contracts/dao/neutron-staking-rewards/src/contract.rs:517`, the `safe_query_user_stake` function queries the Staking Info Proxy contract to compute the user's stake amount.

According to the documentation, the Staking Vault contract is the provider for the Staking Info Proxy contract, and if Liquid Staking Tokens (LST) protocols are blacklisted from the Staking Vault (as per information provided by the client), the query will incorrectly return zero.

Consequently, liquid staking token protocols will be computed as having zero stake, preventing them from receiving rewards.

Recommendation

We recommend modifying the Staking Info Proxy contract to query the Staking Tracker contract instead of the Staking Vault contract during stake retrieval. This change ensures that stake amounts in LST protocols are accurately tracked and rewarded.

Status: Resolved

35. Validators with significant voting power could censor others to prevent them from receiving rewards

Severity: Major

In `x/revenue/types/params.go:86-99`, the default `PerformanceRequirement` for blocks and oracle votes is configured with `AllowedToMiss` at `0.005` and `RequiredAtLeast` at `0.9`. This setup, combined with the CometBFT mechanism where a block proposer can exclude other validators from the votes as long as it includes at least 2/3 of the voting power, opens up the possibility of censorship.

Under this approach, a validator with 10% of the total stake can censor other validators on blocks for which they are the proposer. This forces the targeted validators' performance to fall below the `RequiredAtLeast` threshold, preventing them from receiving compensation.

Similarly, validators with more than 0.5% of the total stake can reduce the compensation for targeted validators by ensuring their performance does not meet the full requirement.

Recommendation

We recommend implementing monitoring and analysis mechanisms to detect and punish dishonest validators. This could involve tracking patterns of exclusion and applying penalties to validators found to be censoring others, ensuring fair compensation distribution.

Status: Acknowledged

The client states that they are going to monitor validators, which will show validators' performance and potential misbehaviors. Also, considering that Main DAO will be the largest staker, it will able to punish malicious (or just poorly performed) validators directly.

36. Validators can manipulate oracle prices to maximize rewards

Severity: Major

Validators with significant voting power can manipulate the price of the reward asset, particularly when they propose blocks.

Due to the stake-weighted median method used by Slinky, validators with 1/3 of the stake can "exactly force" the price in a block they propose and strongly influence the price in other blocks. This manipulation can be exacerbated if the TWAP Window is shorter than the reward evaluation period.

Consequently, validators can submit incorrect prices for brief periods (i.e., before the end of the payment schedule period) to maximize their rewards.

Recommendation

We recommend implementing robust monitoring and validation mechanisms to detect and prevent price manipulation by validators. This could be achieved by increasing the TWAP window to reduce the impact of short-term manipulations and introducing penalties for validators who submit inaccurate prices.

Additionally, consider implementing alternative methods for determining the reward asset price that is less susceptible to manipulation by validators.

Status: Acknowledged

The client states that, for now, they will have off-chain monitoring for prices reported by validators. If misbehavior or price manipulation is detected, responsible validators will be punished.

37. Potential denial-of-service due to significant state reads

Severity: Major

In the current implementation, when a user delegates, undelegates, or redelegates, it triggers a series of actions that can lead to a significant number of state reads.

The process involves multiple calls across different contracts and state queries, resulting in a potential `O(s * p * v)` complexity, where `s` represents the number of slashing events, `p` represents the number of providers, and `v` represents the number of validators.

This complexity arises because each action triggers a chain of calls and queries, including:

1. Calling the `AfterDelegationModified` hook as implemented in `neutron-staking-tracker`.
2. Calling the `UpdateStake` function in `neutron-staking-info-proxy`.
3. Calling the `UpdateStake` function in `neutron-staking-rewards`.

4. Calling the `process_slashing_events` function to handle unprocessed slashing events for users.
5. The `safe_query_user_stake` function will be called for each `s` slashing event that has not been processed for a user. The number of iterations may be significant depending on the user's actions and the chain's operating period.
6. This queries `UserStake` in `neutron-staking-info-proxy`.
7. For each `p` provider, the `StakeAtHeight` query is called.
8. For each `v` validator, the `VALIDATORS` state is being read, and if the validator is in bonded status, the `DELEGATIONS` state is retrieved to compute the total stake amount.

With a potentially large number of `s` slashing events and `v` validators, every delegate, undelegate, or redelegate action will trigger a large number of state reads, potentially leading to out-of-gas errors.

Additionally, if the chain has been operating for some time and a new user joins, all previous slashing events are iterated to recompute the user's stake amount. This is unneeded because the user has no previous stake, unnecessarily increasing the computational power required.

This issue may also occur when a user claims their rewards (starting from step 4).

Recommendation

We recommend applying the following recommendations:

- Only iterate slashing events if the user's stake amount is larger than zero. This prevents unnecessary computations for new users with no previous stake.

- Separating the `VALIDATORS` state into `BONDED_VALIDATORS` and `UNBONDED_VALIDATORS` to reduce the number of validators that need to be iterated over, as suggested in the "Unbounded validator iteration causes potential out-of-gas errors" issue.

Status: Acknowledged

The client states that the whole event is very unlikely to happen. They have a 330M gas limit in a block, and it's almost impossible to create so many slashing events to bloat the contract. If a validator behaves so badly that it creates many slashing events, it will be kicked out of the set much sooner than the state will be bloated.

38. Missing genesis state validation in `x/revenue`

Severity: Minor

In `x/revenue/genesis.go:12-29`, the `Validators` and `Prices` fields in the `GenesisState` are stored without validation, as seen in lines `20-35`. This is problematic because msiconfigurations could occur, causing overflows or incorrect TWAP calculations.

For example, if a ValidatorInfo entry starts with a non-zero CommitedBlocksInPeriod or CommitedOracleVotesInPeriod, these fields may grow and exceed the number of blocks in the initial payment schedule period, resulting in overflows when calling the PerformanceRating function (see x/revenue/keeper/keeper.go:225-231) and negative values for the missedBlocks and missedOracleVotes parameters.

Additionally, invalid CumulativePrice entries could cause inaccurate TWAP results for GetTWAPStartingFromTime if used in x/revenue/keeper/twap.go:211, potentially distributing incorrect payments to validators.

Recommendation

We recommend applying the following recommendations:

- Validate the CommitedBlocksInPeriod and CommitedOracleVotesInPeriod fields for each ValidatorInfo entry in GenesisState.Validators to ensure they do not exceed the initial payment schedule block period.

- For GenesisState.Prices, recompute and override all CumulativePrice values in each RewardAssetPrice entry from the AbsolutePrice and Timestamp series to guarantee correctness.

Status: Resolved

39. Inconsistent provider definition in the Staking Info Proxy contract and documentation

Severity: Minor

In contracts/dao/neutron-staking-info-proxy/src/contract.rs:272, the query_voting_power function queries ProviderStakeQuery::VotingPowerAtHeight from a provider.

The documentation indicates that the Staking Vault contract is a provider for the Staking Info Proxy contract.

However, according to the code suggested in PR 18, which migrates the blacklist address logic from the Staking Tracker contract to the Staking Vault contract, the provider definition becomes the neutron-staking-tracker contract instead of the neutron-staking-vault contract.

Consequently, the query to ProviderStakeQuery::VotingPowerAtHeight will fail, as this query is implemented in the Staking Vault contract, not the Staking Tracker contract (which implements ProviderStakeQuery::StakeAtHeight).

Recommendation

We recommend ensuring that the documentation aligns with the code. Additionally, consider verifying that the correct provider is being queried for `VotingPowerAtHeight` (or `StakeAtHeight`) and update the documentation or code accordingly to reflect the accurate provider.

Status: Resolved

40. Lack of validation for `TwapWindow` can lead to recent price deletion

Severity: Minor

In `x/revenue/types/params.go:52-83`, the `Validate` function for the `Params` struct does not include a check for the `TwapWindow` parameter. If `TwapWindow` is misconfigured to a negative or zero value, it could result in the deletion of all reward asset prices in the `CleanOutdatedRewardAssetPrices` function, as seen in `x/revenue/keeper/twap.go:166`.

Additionally, if `TwapWindow` is misconfigured too high, it can delay the removal of outdated prices, causing inordinate state growth, excessive price smoothing, and reduced sensitivity to recent market changes, ultimately leading to inaccurate revenue calculations.

Recommendation

We recommend adding validation for the `TwapWindow` parameter to ensure it is within a reasonable range. This should include validations to prevent negative or zero values and limit excessively high ones. This validation will help maintain the integrity and efficiency of the reward asset price management system.

Status: Resolved

41. Potential overflow during block interval calculations

Severity: Minor

In `x/revenue/types/payment_schedule.go:137-146`, the `ValidatePaymentScheduleType` function does not verify that `BlocksPerPeriod` is configured within a reasonable range.

If the `BlocksPerPeriod` value is configured too high, an overflow error may occur when it is added to the `CurrentPeriodStartBlock` variable in line `69`. This causes the `BlockBasedPaymentSchedule.PeriodEnded` function to incorrectly return `true`.

Consequently, the PreBlockHandler.PaymentScheduleCheck function will always trigger validator payouts in x/revenue/preblock.go:98, potentially depleting the treasury.

Recommendation

We recommend adding an upper bound limit for BlocksPerPeriod (e.g., a constant max value) and validating that BlocksPerPeriod never exceeds this limit. This ensures that the sum of CurrentPeriodStartBlock and BlocksPerPeriod cannot overflow.

Status: Resolved

42. Incorrect TWAP response returned from the PaymentInfo query

Severity: Minor

In x/revenue/keeper/grpc_query_server.go:52, the GetTWAPStartingFromTime function is called with the startAt parameter configured to the current block time. In x/revenue/keeper/twap.go:198-212, the firstPrice and lastPrice variables will refer to the same RewardAssetPrice, and the function will return early with the lastPrice.AbsolutePrice value in line 209.

However, this contradicts the documentation comments for QueryPaymentInfoResponse in proto/neutron/revenue/query.proto:57-63, which implies that reward_denom_twap should reflect an average price over time. The RewardDenomTwap will also be different from the TWAP used to compute the value of the BaseRevenueAmount field in the CalcBaseRevenueAmount function (see x/revenue/keeper/grpc_query_server.go:61).

Recommendation

We recommend using GetTwap to compute a genuine TWAP for RewardDenomTwap in the query response. This ensures consistency with the TWAP logic used to calculate BaseRevenueAmount and aligns with the intended behavior described in the documentation.

Status: Resolved

43. Negative prices are not handled

Severity: Minor

In x/revenue/keeper/keeper.go:305-307, the CalcBaseRevenueAmount function checks if assetPrice is zero but does not handle cases where it is a negative value.

Consequently, negative prices reported from Slinky could go undetected, leading to failed revenue payments due to negative base revenue amounts.

Recommendation

We recommend validating that `assetPrice` is greater than zero by using a “less than or equal to zero” check instead of an “equal to zero” check.

Status: Resolved

44. Validator rewards are lost due to possible fund shortages

Severity: Minor

In `x/revenue/keeper/msg_server.go:43`, the `FundTreasury` function is not called during the `sovereign` upgrade. Thus, to ensure that validators receive their rewards, the treasury must be funded manually, potentially through a DAO or governance proposal.

Without adequate funding, validators may not be compensated for their contributions.

Specifically, in `x/revenue/keeper/keeper.go:250-261`, the `x/revenue` module attempts to distribute rewards to validators using `k.bankKeeper.SendCoinsFromModuleToAccount`. If this transaction fails due to insufficient funds in the treasury (`RevenueTreasuryPoolName`), the function does not return an error but instead logs the failure and emits an event.

While execution continues, the issue is that a failed bank message results in `validatorInfo` being reset. This means that when the treasury is later topped up, the previously lost rewards are not retried, and affected validators permanently miss their compensation. Since some validators may receive their rewards while others do not, there is an inconsistency in distribution.

Recommendation

To ensure continuous and reliable funding for validator rewards, we recommend calling the `FundTreasury` function as part of the `sovereign` upgrade process. This should include careful allocation of sufficient `NTRN` tokens to cover `StakingRewards`, `Delegation` (see the “Risk of chain takeover if staking levels are insufficient” issue), and `RevenueTreasury` needs, ensuring that the system remains operational and validators are appropriately rewarded.

We also recommend implementing the `EventAttributeRevenueAmount` attribute, which should be set to `revenueAmt` when payment fails in `x/revenue/keeper/keeper.go:257-260`. This will facilitate tracking and processing unpaid validator revenue claims more easily.

Additionally, consider modifying the reward logic to queue failed payments and retry them once the treasury is replenished, ensuring that validator rewards are not permanently lost due to temporary funding shortages.

Status: Resolved

45. Missing upper bounds limits in the `neutron-staking-rewards` contract

Severity: Minor

In `contracts/dao/neutron-staking-rewards/src/state.rs:18-28`, the `validate` function does not enforce upper bounds for `annual_reward_rate_bps`. This omission allows extreme values to be set, which may lead to excessively high or negligible `rate_per_block` value in line `453`.

Recommendation

We recommend introducing maximum limits for the `annual_reward_rate_bps` field (e.g., 10,000 bps for 100%).

Status: Resolved

46. Unbounded iteration when processing validator payments may slow down or halt the chain

Severity: Minor

In `x/revenue/keeper/keeper.go:215-248`, the `ProcessRevenue` function retrieves all validator information using `k.GetAllValidatorInfo(ctx)` and iterates through it to calculate and distribute compensation.

However, this loop is not constrained by the `x/staking` module's `MaxValidators` parameter, meaning it processes all validators that have existed within the payment period. Many validators may enter and exit the network if the payment period is configured to be lengthy, causing the `revenuetypes.ValidatorInfo` store to grow significantly.

This results in increased loop processing, which slows down the chain. In the worst-case scenario, ABCI methods exceeding their allowed execution time could lead to chain halts, as they are expected to complete within a specified period.

We classify this issue as minor severity because only the authority can configure the payment period, which is a privileged address. The validator info state is also reset whenever the ongoing period ends or the payment schedule is updated.

Recommendation

We recommend introducing a configurable parameter to limit the number of validators processed per block and batching the `ProcessRevenue` operation over multiple blocks if necessary.

This can be achieved by modifying the `ValidatorInfo` storage key to include the payment period's start height, enabling the previous period's `ValidatorInfo` entries to be processed in batches and then removed. At the same time, the new `ValidatorInfo` is simultaneously recorded for the current period.

Status: Acknowledged

The client states that they may fix this in the future releases, but for now, they do not think it is a problem since they are not planning to have a huge payment period, and it is impossible (or really expensive) to cause problems by this issue with their planned payment period (1 month).

47. The DAO address is not blocked from claiming rewards

Severity: Minor

In `contracts/dao/neutron-staking-rewards/src/contract.rs:173-175`, the `ExecuteMsg::UpdateStake` function returns a `ContractError::DaoStakeChangeNotTracked` error when the user address matches the `Config::dao_address`.

However, this validation is not implemented in the `claim_rewards` function (see lines `236-273`). This is problematic because both functions call `process_slashing_events` before computing `pending_rewards`, yet `claim_rewards` allows the DAO to successfully claim rewards based on its current stake. This bypasses the intended rule to disallow the DAO to earn rewards on its natively staked `untrn`.

We classify this as minor severity as the DAO itself must issue this message, which could only happen if a governance proposal is passed.

Recommendation

We recommend updating the `claim_rewards` function to be consistent with `update_stake.` If the caller is the DAO address, the function should return an appropriate error.

Status: Resolved

48. Updating the `Config::staking_denom` field may cause incorrect rewards calculations and failure in claiming rewards

Severity: Minor

Both the `neutron-staking-rewards` and `neutron-staking-info-proxy` contracts allow the contract owner to update `Config::staking_denom` independently, as seen in `contracts/dao/neutron-staking-rewards/src/contract.rs:137-139` and `contracts/dao/neutron-staking-info-proxy/src/contract.rs:94-96`.

If a contract's `staking_denom` is changed without also updating the other in the same transaction, calls relying on consistent denominations (e.g., `ExecuteMsg::UpdateStake` and `ExecuteMsg::ClaimRewards`) will fail with `ContractError::InvalidStakeDenom` error returned in `contracts/dao/neutron-staking-rewards/src/contract.rs:532`.

Since errors returned from the `update_stake` function will be ignored, the `UserInfo::pending_stake` computation will be incorrect because the `UserInfo::user_reward_index` is not updated when the user's stake amount changes. This is further explained in the "Failure in updating user stake may cause excess reward distribution" issue.

Additionally, errors returned from the `claim_rewards` handler will revert the transaction, preventing users from claiming their rewards until the `staking_denom` mismatch is resolved. Once resolved, however, the claimable amounts will be incorrect if the user's stake has changed during the time the error was being returned due to `update_stake` failing without chain state reversion.

Furthermore, modifying the `staking_denom` in both contracts implicitly converts pending rewards, accrued under the old denom, into the new denom at a 1:1 rate. This is currently undocumented behaviour that may confuse users and future maintainers.

We classify this issue as minor severity because only the contract owner can update the staking denom, which are privileged addresses.

Recommendation

We recommend applying the following recommendations:

- Option A: Maintain a single source of truth for `staking_denom`, such as storing it only in `neutron-staking-info-proxy` contract and having `neutron-staking-rewards` contract query that value as needed. Additionally, consider documenting or revising the pending rewards conversion to ensure users clearly understand that previously accrued rewards will be transitioned to the new asset at a 1:1 rate.

- Option B: If modifying the `staking_denom` is not deemed necessary, configure `Config::staking_denom` to be "static" in both the `neutron-staking-rewards` and `neutron-staking-info-proxy` contracts by setting it in the `instantiate` handler and not allowing the field to be updated. Additionally, ensure that both contracts are configured with the same value in automated deployment scripts.

Status: Acknowledged

The client states that they assume that the staking denom is static and that no additional logic should be implemented for changing it.

49. Potential TWAP window misalignment

Severity: Minor

In `x/revenue/keeper/twap.go:51-54`, outdated reward asset prices are removed during `k.CleanOutdatedRewardAssetPrices`. However, suppose the `TwapWindow` is configured to a longer duration (e.g., weekly) while the reward distribution occurs at a shorter interval (e.g., daily). In that case, the TWAP calculation during the first period will not fully respect the intended window.

This occurs because the TWAP requires a complete window of past data, which is unavailable at the start.

Recommendation

We recommend introducing a validation check to ensure that the initial payment schedule period equals the `TwapWindow`. This ensures that the TWAP mechanism has sufficient historical data before the first reward distribution, preventing inconsistencies in early calculations.

Status: Acknowledged

The client states that it does not make sense to have a validation check to ensure that the initial payment schedule period equals the `TwapWindow`. This would allow them to use a long TWAP window for short reward distribution periods (not in plans, but in theory, it should be possible).

50. Missing enforcement of validator reward eligibility based on active set status

Severity: Minor

The client has specified two general situations for validator reward eligibility:

- A validator in the active set at the start of the period is entitled to a payout for that period. Missed blocks and votes during the period count against their performance rating, affecting their final payout.

- A validator not in the active set at the start of the period is not entitled to a payout for that period. Such a validator is not considered committed to signing blocks for the period and does not receive a payout.

However, the current implementation does not enforce the second point. If a validator joins the active set shortly after the period starts, the current logic incorrectly distributes rewards for this period.

This is because the only criteria for calculating the `PerformanceRating` (which affects the compensation amount) is the number of missed blocks/oracles during the payment schedule period, as seen in `x/revenue/keeper/keeper.go:225-231`.

Recommendation

We recommend updating the `ProcessRevenue` function to enforce the rule that only validators in the active set are eligible for rewards at the start of the period. This ensures that validators joining the active set after the period starts do not receive payouts for that period, maintaining consistency with the guidelines.

Status: Resolved

51. Potential chain halt due to pre-blocker retrieving deleted validator's state

Severity: Informational

In `x/revenue/preblock.go:142-151`, the pre-block handler iterates over `extendedCommitInfo.Votes` and retrieves the validator using `h.stakingKeeper.GetValidatorByConsAddr` in line `148`.

According to the documentation, "**the agreed-upon vote extensions at height H are provided to the proposing validator at height H+1**." This means that vote extensions from the previous block (*H*) are processed in the current block (*H+1*). This is problematic because if a validator is removed in the previous block (via the `RemoveValidator` function), the `GetValidatorByConsAddr` state will be deleted, causing the pre-block handler to error.

Although we did not find an exploitation path for this issue (as `RemoveValidator` is only called for unbonded validators), returning an error in the pre-block handler would halt the chain, causing a denial of service.

Recommendation

We recommend skipping the entry if the `GetValidatorByConsAddr` function returns an error, similar to Skip Connect's implementation.

Status: Resolved

This issue only occurs during a one-block delay between the validator set update and the Comet set, which could only happen in a chain that uses ICS. While this issue does not affect Neutron (as there is no delay in set updates for the usual PoS Cosmos SDK chains), this issue is retained to alert projects that fork Neutron's codebase during an ICS chain setup.

52. Lack of documentation regarding consensus participation conditions

Severity: Informational

In `x/revenue/keeper/keeper.go:166-173`, the `CommitedBlocksInPeriod` counter is set to be incremented for all `BlockIDFlag` values except `BlockIDFlagAbsent`.

This is problematic because `BlockIDFlagUnknown` and `BlockIDFlagNil` are treated as committed votes, although CometBFT's documentation mentions that these flags "*indicate an error condition*" and "*voted for nil*" (against the majority).

As the rationale behind interpreting the `BlockIDFlagUnknown` and `BlockIDFlagNil` flags as valid votes is not explicitly documented, this may confuse readers by appearing to increment `CommitedBlocksInPeriod` incorrectly.

Recommendation

We recommend revising the implementation and documenting comments on why these flags are not excluded if the behavior is intentional. Otherwise, consider restricting the logic to `BlockIDFlagCommit` only.

Status: Acknowledged

53. Contracts should implement a two-step ownership transfer

Severity: Informational

The `neutron-staking-rewards` and `neutron-staking-info-proxy` contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer, as seen in contracts/dao/neutron-staking-rewards/src/contract.rs:125-127 and contracts/dao/neutron-staking-info-proxy/src/contract.rs:88-90.

While this is common practice, it presents a risk for the contract ownership to become lost if the owner transfers ownership to an incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

The client states that they find this mechanism complicated and non-usable in their case since a change of owner is quite a rare event. Each transaction is being carefully reviewed, and in their case, the owner should not be changed since all contracts will be owned by the Main DAO itself.

54. Query returns incomplete information

Severity: Informational

In contracts/dao/neutron-staking-info-proxy/src/contract.rs:211, the query_config function returns the contract configuration to the caller. However, the Config.staking_denom field is not included in the ConfigResponse.

Similarly, the query_state function in contracts/dao/neutron-staking-rewards/src/contract.rs:301 does not include the State.slashing_events field in the StateResponse.

Consequently, queries will return incomplete information to the caller, reducing developer experience as they must perform a raw query to retrieve the underlying values.

Recommendation

We recommend adding the missing fields to the ConfigResponse and StateResponse.

Status: Acknowledged

55. Lack of provider attributes in `update_providers` response hinders debugging

Severity: Informational

In `contracts/dao/neutron-staking-info-proxy/src/contract.rs:137-139`, the `update_providers` function updates the list of providers but does not include attributes for the new providers in the response. The function also does not log the old providers before updating them.

Consequently, this lack of logging makes it difficult to track changes and verify updates, hindering debugging and auditing processes for external smart contracts callers and indexers.

Recommendation

As a standard practice when updating critical configuration, we recommend modifying the `update_providers` function to include attributes for both the old and new providers in the response. This will enhance transparency and facilitate more effective debugging and auditing processes.

Status: Acknowledged

56. Unhandled parsing errors during `query_providers`

Severity: Informational

In `contracts/dao/neutron-staking-info-proxy/src/contract.rs:220-226`, using `Iterator::flat_map` can silently discard parsing errors when converting key byte slices to the `Addr` type. If any key is malformed, it will be skipped, resulting in fewer providers returned than actually exist.

Additionally, the code uses `ToString::to_string` on each address, causing unnecessary string allocations in a frequently used query.

We classify this issue as informational severity as currently all addresses are validated in line `133` before being written to storage.

Recommendation

We recommend using `Result::unwrap` or `Result::expect` if key parsing is never expected to fail. Otherwise, consider replacing `Iterator::flat_map` with `Iterator::map` and collect the entries into `StdResult<Vec<String>>`. This allows the `query_providers` function to be more resilient to future changes, such as the removal of validation in other regions of the codebase.

Additionally, consider using `Addr::into_string` to convert from `Addr` to `String` without the usage of allocation or copy.

Status: Acknowledged

57. Code duplication when updating global index

Severity: Informational

In `contracts/dao/neutron-staking-rewards/src/contract.rs:410-424`, the `update_global_index` function contains duplicated logic to update the global index and state. This is because the `get_updated_state` function already contains similar logic and can be reused to avoid duplication.

This duplication may increase code maintainability challenges, as any updates to this logic would need to be applied in multiple places, which is error-prone.

Recommendation

We recommend refactoring the `update_global_index` function to utilize the response from the `get_updated_state` function. This approach ensures that any updates to the logic are automatically reflected in both locations, reducing the risk of inconsistencies and errors.

Status: Acknowledged

58. Inefficient state loading leads to unnecessary resource usage

Severity: Informational

In `contracts/dao/neutron-staking-info-proxy/src/contract.rs:150` and `176`, the `update_stake` and `slashing` functions load the `CONFIG` state before validating whether the caller is authorized. This results in unnecessary state loading when the caller is not authorized, leading to inefficient resource usage.

Recommendation

We recommend moving the `CONFIG` state read logic after the authorization check. This ensures that the state is only loaded for authorized requests, improving efficiency.

Status: Acknowledged

59. Unnecessary string copies and allocations from overuse of `Clone::clone`

Severity: Informational

Multiple functions in `contracts/dao/neutron-staking-rewards/src/contract.rs` unnecessarily pass large, heap allocated, parameters by value, necessitating many calls to `Clone::clone` where those functions are used. This results in needless extra allocations and string copies, increasing the computational cost of frequently executed functions such as `update_stake`.

These instances are illustrated below:

- The `config` parameter in lines `379`, `410`, and `429`.
- The `user_info` parameter in line `483`.
- The `user_addr` and `staking_info_proxy` parameters in lines `380`, `465`, `510`, and `511`.
- The `user` and `staking_denom` parameters in lines `163`, `311`, `364`, `466`, `485`, and `512`.

In each location, parameters can be reference types rather than “owned” types, eliminating many unnecessary `Clone::clone` calls.

Recommendation

We recommend applying the following recommendations:

- Update function signatures to accept references (e.g., `&Config`, `&UserInfo`, `&Addr`, `&str`) instead of owned values where ownership is not required.
- If a function accepts an old value and returns an updated value, such as `get_updated_state` and `get_updated_user_info`, consider modifying the parameter to be mutable to avoid cloning at the beginning of the function (e.g., `mut state: State`).
- Remove `Clone::clone` calls that become redundant after switching to parameters to reference types.
- Only use `Clone::clone` or `ToOwned::to_owned` if an external API or internal logic explicitly requires an owned copy and convert to the owned copy only at the point of use.

Status: Acknowledged

60. Lack of slashing event cleanups leads to inefficient state management

Severity: Informational

In contracts/dao/neutron-staking-rewards/src/state.rs:39, the State.slashing_events vector accumulates slashing events from various hooks, including after_validator_bonded, after_validator_begin_unbonding, and before_validator_slashed.

The issue is that there is no mechanism to clean up or remove old slashing events. This may lead to an ever-growing list of slashing events, potentially causing inefficiencies and increased storage usage.

Recommendation

We recommend removing slashing_events from the State struct and storing them in their own storage map using the height as the key. This approach will increase the efficiency of the load_unprocessed_slashing_events function by skipping the requirement to iterate over irrelevant events with Iterator::skip_while.

While this does not directly address the removal of old events, it ensures that the cost of reading and writing the State struct remains constant. Also, processing slashing events can start iterating from the user's last update height.

Status: Acknowledged

61. Lack of pausing feature

Severity: Informational

The neutron-staking-rewards contract lacks a pause/unpause feature. This feature helps as a precautionary measure to prevent unexpected behaviors, such as draining all funds from the contract.

Recommendation

We recommend implementing a pausing mechanism that is authorized for the contract owner and the neutron-staking-info-proxy contract as per the "Failure in updating user stake may cause excess reward distribution" issue. This feature should pause the claim_rewards function to prevent reward claims during suspicious activities while allowing updates for the update_stake and slashing functions. This ensures that the contract remains functional for critical operations while mitigating risks.

Status: Acknowledged

62. Misleading comment in the `query_voting_power` function

Severity: Informational

In `contracts/dao/neutron-staking-info-proxy/src/contract.rs:262-263`, the comment indicates that the user's voting power query will validate the returned denom against `Config::staking_denom`. However, no such validation occurs or could occur due to the `ProviderStakeQuery::VotingPowerAtHeight` query result being a standalone `Uint128`.

Consequently, this inconsistency could potentially mislead or confuse future maintainers.

Recommendation

We recommend updating the comment to align with the function's actual behavior.

Status: Acknowledged

63. The `query_user_stake` function can be optimized

Severity: Informational

In `contracts/dao/neutron-staking-info-proxy/src/contract.rs:230-246`, the provider keys are iterated over multiple times: first to parse them into a `Vec<Addr>`, then again to collect the voting power values into another vector, and finally to sum these values.

This pattern introduces redundant iterations and allocations, particularly in the typical "happy path" scenario for a query that will be frequently used.

Recommendation

We recommend performing a single pass over the keys and aggregating each provider's voting power directly, returning early on error. We also recommend using a mutable accumulator or leveraging `Iterator::try_fold` to streamline error handling and summation, thereby reducing overhead and improving clarity.

Status: Acknowledged

64. `to_address` is not checked to be a valid address

Severity: Informational

In `contracts/dao/neutron-staking-rewards/src/contract.rs:258-264`, the `to_address` parameter provided by the caller is not checked to be a valid bech32 address before sending the rewards to the address.

Consequently, if the issued `BankMsg::Send` fails due to a malformed nominated `recipient` address, this could lead to a confusing error message.

Recommendation

We recommend using `Api::addr_validate` to ensure the `to_address` is a valid bech32 address for the chain before using it as the `recipient` and returning an informative error if not.

Status: Acknowledged

65. Spelling errors in the codebase

Severity: Informational

In `proto/neutron/revenue/genesis.proto:46-50`, the `commited_blocks_in_period` and `commited_oracle_votes_in_period` fields are misspelled (the correct spelling is "*committed*"). As these fields are auto-generated throughout the codebase, the misspelling appears in 70 locations in total.

Additionally, the command description in `x/revenue/client/cli/query.go:61` contains a typo: it mentions "*shows the payment into*" instead of "*shows the payment info*".

Recommendation

We recommend correcting the spelling in the Protobuf file, regenerating the code, and updating all references to the incorrectly spelled fields. Additionally, consider correcting the typo in the CLI command description to accurately reflect its functionality.

Status: Acknowledged

66. Usage of magic numbers during reward rate conversion

Severity: Informational

In `contracts/dao/neutron-staking-rewards/src/contract.rs:449`, the conversion of the annual reward rate from basis points (bps) to a decimal is performed using a hardcoded divisor of `10_000`. This value represents the total basis points in 100%, but using a raw number directly reduces code readability and maintainability.

Recommendation

We recommend defining a constant, such as `ONE_HUNDRED_PERCENT_BPS`, set to `10_000`, and implementing it in the conversion. This improves clarity and ensures that if the basis point system changes, updates can be made in a single place.

Status: Acknowledged

67. Optimization by storing precomputed reward rate per block for gas efficiency

Severity: Informational

In `contracts/dao/neutron-staking-rewards/src/contract.rs:450-453`, the contract currently derives `rate_per_block` dynamically by dividing `annual_reward_rate_bps` by `blocks_per_year` whenever it is needed. Since this calculation involves fixed-point arithmetic and division operations, it incurs unnecessary gas costs, especially in frequent reward distribution calls.

Since `rate_per_block` remains constant between configuration updates, recalculating it repeatedly is inefficient and increases gas consumption. By storing `rate_per_block` directly in the configuration, the contract can avoid redundant computations, leading to a reduction in gas costs per transaction.

Recommendation

We recommend modifying the contract to store `rate_per_block` directly instead of storing `annual_reward_rate_bps` and `blocks_per_year`. This change optimizes gas usage by eliminating repetitive calculations.

Status: Acknowledged

Appendix

1. Test case for "Rounding discrepancy in before_validator_slashed leads to misestimation of slashed tokens"

```
#[test]
fn slashed_tokens() {
    // SDK-level slashing math (Go):
    // tokens := sdkmath.NewIntFromUint64(999_999_999_999_999)
    // fraction := sdkmath.LegacyNewDecWithPrec(3, 2)

    // slashAmountDec := sdkmath.LegacyNewDecFromInt(tokens).Mul(fraction)
    // slashAmount := slashAmountDec.TruncateInt()
    // fmt.Printf("slash amount: %s\n", slashAmount)

    // effectiveFraction :=
sdkmath.LegacyNewDecFromInt(slashAmount).QuoRoundUp(sdkmath.LegacyNewDecFromInt(
tokens))
    // fmt.Printf("effective fraction: %s\n", effectiveFraction)
    //
    // Results in:
    // slash amount: 29999999999999
    // effective fraction: 0.029999999999999030

    let slashing_fraction: Decimal256 = "0.029999999999999030".parse().unwrap();

    let tokens = Decimal256::from_ratio(999_999_999_999_999u128, 1u128);

    assert_eq!(
        // using ceil results in inconsistency with x/staking module
        (slashing_fraction * tokens).to_uint_ceil(),
        Uint256::from_u128(30_000_000_000_000)
    );

    assert_eq!(
        // using floor remains consistent with x/staking module
        (slashing_fraction * tokens).to_uint_floor(),
        Uint256::from_u128(29_999_999_999_999)
    );
}
```