Security Audit Report

Hub 2024 Q2: Partial Set Security

Authors: Josef Widder, Ivan Golubovic, Aleksandar Ljahovic

Last revised 5 June, 2024

Table of Contents

Audit Overview

In May 2024, Informal Systems conducted a security audit for Hub team. The audit aimed at inspecting the correctness and security properties of Partial Set Security. The relevant resources for the audit were following:

- EPIC to track work — https://github.com/cosmos/interchain-security/issues/853
- ADR — https://cosmos.github.io/interchain-security/adrs/adr-015-partial-set-security
- WIP feature branch — https://github.com/cosmos/interchain-security/tree/feat/partial-set-security
- PSS PR is out in draft. https://github.com/cosmos/interchain-security/pull/1809
- the diff: https://github.com/cosmos/interchain-security/compare/release/v4.1.x...release/v4.2.x
- InterchainSecurity repo with frozen tag: https://github.com/cosmos/interchain-security/tree/v4.2.0-rc0

The audit was performed from May 6, 2024 to May 29, 2024 by the following personnel:

- Josef Widder
- Ivan Golubovic
- Aleksandar Ljahovic

Relevant Code Commits

The audited code was from the interchain-security repository at the following commit:

- hash `71666b171267c4c48a0526ccee022b49d1295979`

Conclusion

We performed a thorough review of the project. We found some subtle problems - more on them in the section Findings .

We are glad to report that the dev team has acknowledged our findings and is working towards fixing them.

Audit Dashboard

Target Summary

- **Type**: Protocol and Implementation
- **Platform**: Go
- **Artifacts**: https://github.com/cosmos/interchain-security

Engagement Summary

- **Dates**: 06.05.2024 - 29.05.2024
- **Method**: Manual code review, protocol analysis

Severity Summary

Finding Severity	**#**
Critical	0
High	0
Medium	0
Low	4
Informational	6
Total	**10**

System Overview

Partial Set Security (PSS) enhances Interchain Security (ICS) by allowing consumer chains to be validated by subsets of the provider chain's validators, specifically within the Cosmos Hub ecosystem.

Under Interchain Security, the provider chain (Cosmos Hub) used its entire validator set to secure consumer chains (e.g., Stride and Neutron). PSS introduces two categories of consumer chains validated by subsets of the provider's validators:

1. **Opt In**: Validators can choose to opt in and out at will.
2. **Top N**: The top N% of validators on the Hub are required to validate the consumer chain, with other validators having the option to opt in.

Consumer chains can also use validator-shaping features:

- **Validator-set cap**: Limits the maximum number of validators for a consumer chain.
- **Validator-power cap**: Caps the maximum power a validator can hold on a consumer chain.
- **Allowlist/Denylist**: Specifies which Hub validators can or cannot validate the consumer chain.

Consumer Chain Onboarding

New consumer chains can join as Opt In or Top N and set validator-shaping features via the *ConsumerAdditionProposal*. Existing consumer chains, Stride and Neutron, are automatically migrated to Top 95% chains due to their existing soft opt-out feature.

Key Components

Validator-Set Updates

PSS constructs validator-set updates using epochs (600 block sequences). For each consumer chain, the current validators are filtered and updated based on their power and opt-in status. This process involves:

1. **GetConsumerValSet**: Retrieves the current validator set.
2. **Applies various filters** to the validator set: power capping, power shaping, allow/deny listing.
3. **DiffValidators**: Generates the differences to create ValidatorUpdates for VSCPackets.

Reward Distribution

PSS modifies reward distribution so that only validators who validate a consumer chain receive its rewards. An IBC Middleware tracks rewards from consumer chains, which are then allocated through the AllocateTokens method during BeginBlock. Validators can set different commission rates for different consumer chains.

Threat Model

Threats

1. Unbounded Validator Set (Opt-In Chains)

- **Explanation:** An Opt-In chain might end up with more validators than intended due to a lack of enforcement mechanisms for the `validator-set_cap`. A large validator set can negatively impact performance.
- **Property Checked:** Effectiveness of `validator-set_cap` implementation for Opt-In chains, ensuring it restricts the validator set size.
- **Consequence:** An excessive number of validators in an Opt-In chain can slow down the chain and increase resource consumption.
- **Conclusion:** The following findings were identified during the inspection of this threat:
  Inconsistent Validator Set Capping Behavior in PSS

2. Limitations on Validator Selection

- **Explanation:** A consumer chain's validator set might include unauthorized validators (from denylist) or exclude authorized ones (from allowlist) due to flaws in the filtering logic.
- **Property Checked:** Logic for utilizing allowlist and denylist to ensure only authorized validators are included in the consumer chain's validator set.
- **Consequence:** Including unauthorized validators can compromise security by allowing malicious actors to participate in consensus or disrupt the chain's operation. Conversely, excluding authorized validators can prevent legitimate validators from fulfilling their roles.
- **Conclusion:** The following findings were identified during the inspection of this threat:
  Risk of Unsecure or Non-Operational Validator Sets Due to Validator Selection Parameters
  Redundant Denylist Check with Allowlist Present

3. Power Takeover

- **Explanation:** In a consumer chain, a validator or a colluding group of validators could accumulate excessive voting power exceeding the intended `validator_power_cap`. This power could be used to manipulate consensus or reward distribution.
- **Property Checked:** Power capping mechanism and its ability to enforce `validator-power_cap`, preventing any validator from exceeding the defined power limit on the consumer chain.
- **Consequence:** A validator with excessive power in a chain can disrupt consensus, unfairly influence reward distribution, or manipulate penalties for misbehavior, jeopardizing the overall security and fairness of the system.
- **Conclusion:** The following findings were identified during the inspection of this threat:
  No validation for TopN property
  Potential Optimization in Max Power Capping Logic

4. Reward Misallocation

- **Explanation:** Validators might receive incorrect rewards (more/less than deserved) due to flaws in the reward allocation logic based on voting power or issues with handling consumer chain rewards through the IBC Middleware.
- **Property Checked:** Reward allocation algorithm for consumer chains, ensuring it accurately reflects the voting power of participating validators and correctly handles rewards received via IBC transfers.
- **Consequence:** Validators receiving incorrect rewards can lead to unfairness and potentially incentivize malicious behavior.
- **Conclusion:** The following findings were identified during the inspection of this threat:
  Lack of validator set history leads to race conditions and unexpected behaviour

5. Commission Rate Manipulation:

- **Explanation:** A validator might exploit a vulnerability in the `MsgSetConsumerCommissionRate` message to manipulate their commission rate for a specific consumer chain, potentially gaining an unfair advantage in reward distribution.
- **Property Checked:** Security of the `MsgSetConsumerCommissionRate` message handling and validation to prevent manipulation of commission rates.
- **Consequence:** A validator manipulating commission rates can gain an unfair advantage in rewards, impacting the overall fairness of the system.
- **Conclusion:** The implementation does not suffer from this threat. The commission rate set by the validator for a specific consumer chain is subject to validations that will prevent any attempt at manipulation.

6. Inactive Validator Consensus Disruption

- **Explanation:** The consumer chain's validator set might include inactive validators during epoch boundaries. These validators cannot participate in consensus, potentially halting the chain's operation or causing crashes.
- **Property Checked:** Process for selecting validators for the consumer chain, ensuring only active validators on the hub chain are included during epoch transitions.
- **Consequence:** Including inactive validators can disrupt consensus, potentially halting the chain's progress or even causing crashes, jeopardizing the overall functionality of the consumer chain.
- **Conclusion:** The implementation does not suffer from this threat. The process of selecting only active validators is ensured by the implementation in a way that only validators with *bonded* status could end up validating the consumer chain.

7. Voting Power Discrepancy

- **Explanation:** The relative voting power relationships between validators on the provider and consumer chains might not be preserved during validator set updates. This inconsistency can lead to unexpected power distribution within the consumer chain.
- **Property Checked:** Mechanism for updating the validator set while maintaining the relative voting power relationships between validators as established on the provider chain.
- **Consequence:** Discrepancies in voting power can lead to situations where validators have more or less power than intended on the consumer chain, potentially impacting fairness and security.
- **Conclusion:** The implementation does not suffer from this threat. There has not been identified that a validator power could be recalculated in such a way that it would harm the relative relationship of two validators on provider and consumer chain.

8. Incomplete Consumer Chain State Removal

- **Explanation:** Consumer chain information might not be completely removed from the Hub's state after the chain is stopped or times out. This residual data could lead to unforeseen consequences, such as unintended resource consumption or potential security vulnerabilities.
- **Property Checked:** Completeness and effectiveness of the state cleaning procedure for stopped or timed-out consumer chains, ensuring all relevant data is removed from the Hub.
- **Consequence:** Incomplete removal of consumer chain state can cause wasted storage space, potentially impacting performance. Additionally, residual data might introduce vulnerabilities if exploited by malicious actors, compromising the overall security of the system.
- **Conclusion:** The implementation does not suffer from this threat. It has been identified that all the relevant Partial Set Security data is cleaned up on every consumer chain stoppage/timeout. This data includes allowlist, denylist, TopN, validator set cap and validator power cap as well as stored validator set and opted in validators (some of them might be out of validator set).

Invariants

- **Validator Set Size:**
  - For Opt-In chains, the validator set size is capped at a maximum of `validator_set_cap` validators.

    - For Top N chains, the validator set size is less than or equal to the number of validators in the top N% of the Hub's validator set.
- **Validator Liveness:** Only active validators (bonded) on the Hub chain can be included in the consumer chain's validator set during epoch boundaries.
- **Validator Selection:**
    - The validator set for a consumer chain consists only of validators from the allowlist (if defined) and excludes those from the denylist (if defined).
    - In Top N chains, all validators within the top N% of the Hub's validator set are included in the consumer chain's validator set, unless they are on the denylist.
- **Voting Power Cap:** The voting power of any individual validator on a consumer chain is limited to a maximum of `validator_power_cap` percent of the total voting power on that chain.
- **Reward Allocation:**
    - The total rewards distributed to validators for a consumer chain match the total rewards received from that chain via IBC transfers.
    - The share of rewards a validator receives on a consumer chain is proportional to its relative voting power on that chain.
- **Opt-In Consistency:** A validator's opt-in status for a consumer chain should be consistent across the Hub and the consumer chain itself.
- **Commission Rate Validity:** The commission rate set by a validator for a consumer chain should be within a valid range defined by the protocol.

Findings

Title	Type	Severity	Impact	Exploitability
Potential Optimization in Max Power Capping Logic	IMPLEMENTATION	1 LOW	1 LOW	1 LOW
No validation for TopN property	IMPLEMENTATION	1 LOW	1 LOW	1 LOW
Redundant Denylist Check with Allowlist Present	IMPLEMENTATION	1 LOW	1 LOW	1 LOW
Inaccurate Rounding of maxPower	IMPLEMENTATION	1 LOW	1 LOW	1 LOW
Validator avoids jailing by opting out	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE
Inconsistent Validator Set Capping Behavior in PSS	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE
Lack of validator set history leads to race conditions and unexpected behaviour	IMPLEMENTATION DOCUMENTATION	0 INFORMATIONAL	0 NONE	0 NONE
Risk of Unsecure or Non-Operational Validator Sets Due to Validator Selection Parameters	IMPLEMENTATION DOCUMENTATION	0 INFORMATIONAL	0 NONE	0 NONE
Code Efficiency Improvements and Optimization Opportunities	IMPLEMENTATION	0 INFORMATIONAL	0 NONE	0 NONE

Minor Code Improvements	**IMPLEMENTATION** **DOCUMENTATION**	**0 INFORMATIONAL**	**0 NONE**	**0 NONE**

Potential Optimization in Max Power Capping Logic

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The current implementation of the max power capping functionality in consumer chains utilizes a two-step approach for distributing voting power among validators. The first step involves calculating the total power from validators exceeding the cap and the number of validators with power below the cap. Subsequently, the main loop iterates through validators, adjusting their power and recalculating the remaining power to be distributed in each iteration.

Problem Scenarios

While the current approach in `NoMoreThanPercentOfTheSum` functions correctly, the initial loop dedicated to calculating `remainingPower` and `validatorsWithPowerLessThanMaxPower` introduces unnecessary redundancy. This redundancy can potentially impact code readability and contribute to minor performance overhead.

Recommendation

To optimize the max power capping logic, consider refactoring the code to eliminate the separate loop for calculating `remainingPower` and `validatorsWithPowerLessThanMaxPower` variables. Also, logic within the first `if` branch (`if v.Power >= maxPower`) could be merged into the main processing loop.

The validators that are being processed are already sorted by power in decreasing order. The main for loop in its redesigned form with first if branch merged into the second one, will first process the validators with power greater than the maximum power which makes it possible to calculate the remaining power before the validators with power lower than the maximum power are encountered in the loop.

This optimization would:

- Reduce code complexity by eliminating redundant calculations.
- Potentially improve code readability by condensing the logic.
- Lead to minor performance improvements by reducing loop iterations.

No validation for TopN property

Project	Hub 2024 Q2: Partial Set Security
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The Partial Set Security (PSS) specification defines "Top N" consumer chains where the validator set comprises the top N% of validators from the provider chain, with N ranging from 50 to 100. However, the code review identified places where the implementation lacks validation to ensure N falls within this valid range.

Problem Scenarios

Top N consumer chains are intended to have a validator set size proportional to the chosen N value (between 50% and 100% of the provider's validators). The code contains instances where the provided N value is used in calculations without explicit validation to confirm it adheres to the specified range (HandleOptOut, ComputeMinPowerToOptIn (misleading comment also stating it should be 0-100)).

While the ConsumerAdditionProposal message properly validates the N value to be within the 50-100 range, additional validation in the identified code sections would strengthen the overall security posture. This redundancy acts as an extra layer of state verification, mitigating potential inconsistencies that might arise due to unexpected changes.

Recommendation

To address this incomplete validation, it's recommended to incorporate checks within the following functions to ensure the provided N value for Top N consumer chains falls within the allowed range (50-100):

- `HandleOptOut` function
- `ComputeMinPowerToOptIn` function

These additional checks would fortify the system's integrity by preventing the creation of Top N consumer chains with invalid N values, thereby maintaining the intended validator set sizes and security properties of the PSS design.

Redundant Denylist Check with Allowlist Present

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION
Severity	1 LOW
Impact	1 LOW
Exploitability	1 LOW
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The current implementation of consumer chain validator selection uses both allowlists and denylists. However, the presence of an allowlist effectively makes the denylist redundant.

The code also prioritizes the denylist, so any validator on the denylist will be excluded regardless of being on the allowlist.

Problem Scenarios

- **Redundant Denylist Check:** The code performs an unnecessary check for the denylist if an allowlist is defined. Allowlist is used to exactly point to the validators that are wanted on the chain. The denylist has no real purpose here.
- **Empty Validator Set:** If both allowlist and denylist are non-empty and contain identical entries (allowlist = denylist), the current behavior using denylist precedence results in an empty validator set. This leaves the consumer chain with no validators.

Recommendation

To optimize code execution and address the potential for an empty validator set, consider the following approaches:

Enforce Single List: Modify the consumer chain configuration to allow only one list (allowlist or denylist) to be set at a time. This ensures clarity and eliminates the redundant denylist check.

Allowlist Precedence with Denylist Merge: If maintaining both lists is necessary, prioritize the allowlist and incorporate denylist functionality by removing denylisted validators from the allowlist before selection. This approach streamlines the validation process and avoids the empty validator set scenario.

Inaccurate Rounding of maxPower

Project	**Hub 2024 Q2: Partial Set Security**
Type	**IMPLEMENTATION**
Severity	**1 LOW**
Impact	**1 LOW**
Exploitability	**1 LOW**
Status	**ACKNOWLEDGED**
Issue	

Involved artifacts

- x/ccv/provider/keeper/partial_set_security.go

Description

An analysis of the calculation for the `maxPower` reveals that the calculation may not yield the correct results in certain scenarios. Specifically, when the expression `sdk.NewDec(sum(validators)).Mul(sdk.NewDec(int64(percent))).QuoInt64(100)` evaluates to a whole number, the use of the `Ceil()` function followed by a subtraction of 1 (`- 1`) results in an incorrect value.

Problem Scenarios

The expression `sdk.NewDec(sum(validators)).Mul(sdk.NewDec(int64(percent))).QuoInt64(100)` is intended to calculate a percentage of the sum of validators, and `Ceil()` is used to round up to the nearest whole number. However, if the result is already a whole number, `Ceil()` should ideally leave the number unchanged, akin to what a `Floor()` function would do if it existed in this context. Subtracting 1 from the result in such cases leads to an incorrect value, as demonstrated in the test:

```
func TestMaxPowerCalc(t *testing.T) {
    sumValidators := int64(200)
    percent := int64(2)
    maxPower := sdk.NewDec(sumValidators).Mul(sdk.NewDec(percent)).QuoInt64(100).Ceil
().RoundInt64() - 1

    require.Equal(t, int64(4), maxPower)  // Error: expected: 4, actual: 3
}
```

Recommendation

To address this issue, review the logic used for calculating `maxPower`. Consider the following changes:

1. Evaluate whether the use of `Ceil()` is appropriate when the result is already a whole number.
2. Implement a conditional check to determine if the result of `sdk.NewDec(sum(validators)).Mul(sdk.NewDec(int64(percent))).QuoInt64(100)` is a whole number and handle it accordingly, without subtracting 1.
3. Update the test case to validate the new logic and ensure accurate results.

Validator avoids jailing by opting out

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

Partial Set Security features adds a new type of chains in terms of their validator set called Opted In chains. In such chains, a validator set is made of validators that are willingly taking part in the consensus. In addition, these validators can also opt out of a validator set when they don't want to be part of the consumer chain's consensus anymore.

One of the consequences of such validator set creation process is the jailing process being changed in a way that only opted in validators on consumer chain can be jailed for downtime. Otherwise, all the validators that did not opt in for consumer chain could end up jailed. So, opted out validators are not eligible for jailing by consumer chain.

Problem Scenarios

In PSS, only opted-in validators on a consumer chain can be jailed for downtime or misbehavior. A malicious validator on an Opt-In chain can exploit this by opting-out before the jailing process, evading punishment.

Recommendation

Firstly, the aforementioned fact should be properly documented for the team to be aware of this possibility.

To address this otherwise, consider:

- **Grace Period Jailing:** Allow jailing opted-out validators for a grace period after opting-out to complete the jailing process if misbehavior evidence exists.
- **Reputation System:** Implement a reputation system that tracks a validator's opt-out history across chains. Frequent opt-outs near potential jailing events could trigger further investigation.
- **Alternative Penalties:** Explore alternative penalties beyond jailing for opted-out validators, such as slashing voting power or reducing rewards for a set period.

Inconsistent Validator Set Capping Behavior in PSS

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The current implementation of validator set capping in PSS exhibits inconsistency between Top N and Opt-In consumer chains. In Opt-In chains, the `validator_set_cap` parameter effectively limits the total number of validators participating in the chain. However, for Top N chains, the validator set cap does not restrict the total number of validators.

Problem Scenarios

Consider a scenario where a consumer chain is configured as Top 50, intending to have the top 50 validators from the provider chain as its validator set. Additionally, a `validator_set_cap` of 100 is set, aiming to limit the total number of validators to 100 (including the top 50).

In this scenario, the current implementation would only enforce the cap on validators outside the top 50. This could result in a validator set exceeding 100 if more than 50 validators from outside the top 50 opt-in to participate. This behavior deviates from the expectation of a capped validator set and could lead to unintended consequences such as resource exhaustion or performance degradation.

Recommendation

To ensure consistent behavior for validator set capping across both Opt-In and Top N chains, it's recommended to consider the following approaches:

- **Apply Cap to Entire Set (Top N):** Modify the logic for Top N chains to consider the `validator_set_cap` when selecting validators. This would ensure the total validator set size remains within the specified cap, even for Top N chains.

- **Introduce Separate Cap for Top N:** Implement a separate capping mechanism specifically for Top N chains. This separate cap would define the maximum number of validators allowed beyond the top N, providing more granular control over the validator set size for Top N chains.

Lack of validator set history leads to race conditions and unexpected behaviour

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The current implementation of consumer chains in PSS only stores the current validator set. This lack of historical data can lead to race conditions and unintended consequences, potentially affecting reward distribution and validator jailing.

Problem Scenarios

1. **Reward Misallocation:** Rewards are distributed to the current validator set based on block production, even if prior validators performed the work. This can result in:
   - Opting-out validators missing out on deserved rewards.
   - Opting-in validators receiving rewards they haven't earned.
2. **Jailing Ineffectiveness:** Validators responsible for downtime on a consumer chain can evade penalties by opting out just before the corresponding information reaches the provider chain. The absence of historical validator set data hinders the identification and punishment of such misbehavior.

Recommendation

The PSS specification should be updated to explicitly address the implications of missing historical validator set data. This should include clear guidelines outlining the potential consequences on reward distribution and validator jailing mechanisms.

Risk of Unsecure or Non-Operational Validator Sets Due to Validator Selection Parameters

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- interchain-security/partial-set-security

Description

The validator selection process in PSS, which includes allowlists, denylists, and power shaping features, introduces potential risks of creating insecure or non-operational consumer chain validator sets. Improper configuration of these parameters can lead to critical vulnerabilities and operational disruptions.

Problem Scenarios

An overly restrictive configuration of allowlists or denylists in a consumer chain can result in a very small validator set, potentially even just a single validator. This creates a single point of failure, making the consumer chain highly vulnerable. If this sole validator opts out, the entire consumer chain can halt operations.

Recommendation

To mitigate risks associated with validator selection parameters, the following recommendations are essential:

- **Comprehensive Documentation:** Thorough documentation outlining the functionality and associated risks of allowlists, denylists, and power shaping features should be developed.
- **Consumer Chain Awareness:** Consumer chains must be clearly informed about the potential consequences of setting overly restrictive validator selection parameters. This will empower them to make informed decisions that balance security with operational needs.

Code Efficiency Improvements and Optimization Opportunities

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- /x/ccv/provider/keeper

Description

- In multiple locations, `Int` objects are created unnecessarily. Instead of using `NewDecFromInt`, `NewDec` can be directly used. code1, code2, code3:

  ```
  totalPower = totalPower.Add(sdk.NewDecFromInt(sdk.NewInt(power)))
  ```

  Refactored code:

  ```
  totalPower = totalPower.Add(sdk.NewDec(power))
  ```

- In several mathematical calculations, immutable operations are used where mutable operations could enhance efficiency. code1, code3:

  ```
  totalPower = totalPower.Add(sdk.NewDecFromInt(sdk.NewInt(power)))
  ```

  Refactored code:

  ```
  totalPower.AddMut(sdk.NewDec(power))
  ```

- The variable `topN` is already defined as `chain.Top_N`, making it unnecessary to retrieve its value again in relay.go#L226.

- A new function `IsConsumerTotalVotingPowerZero()` should be created and utilized here. It is much more efficient.
- `math.LegacyNewDec(totalPower)` should be initialized once before entering the loop (code).
- The code can be optimized to avoid sorting validators when `len(validators) < int(validatorSetCap)`, as it's unnecessary to sort:

```
if validatorSetCap, found := k.GetValidatorSetCap(ctx, chainID); found &&
validatorSetCap != 0 && len(validators) >= int(validatorSetCap) {
    sort.Slice(validators, func(i, j int) bool {
        return validators[i].Power > validators[j].Power
    })

    return validators[:int(validatorSetCap)]
}

return validators
```

- Maps isCurrentValidator and isNextValidator should be initialize with size.

Problem Scenarios

It was concluded that the code changes in places above were not critical for gaining significant difference in speed, but suggested optimizations could slightly improve performance.

Recommendation

As explained in the Description section.

Minor Code Improvements

Project	**Hub 2024 Q2: Partial Set Security**
Type	IMPLEMENTATION DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Status	ACKNOWLEDGED
Issue	

Involved artifacts

- /x/ccv/provider/keeper

Description

- The `validatorSetUpdateId` retrieval in /keeper.go#L729-L736 is redundant:

```
if bz == nil {
    return 0
}
return binary.BigEndian.Uint64(bz)
```

- The `consumerKey` parameter in the `HandleOptIn` function does not need to be passed by reference (code). It could simply be a string variable. The check for whether `consumerKey` has been passed could be handled similarly to how it is done in the `OptIn` function within `msgServer` (code). Furthermore, this check would become redundant because it would also be verified within the `HandleOptIn` function.
- The new `allocation.Rewards` value (code) should match the `changeCoins` value from `TruncateDecimal()` (code):

```
rewardsToSend, _ := allocation.Rewards.TruncateDecimal()
allocation.Rewards =
allocation.Rewards.Sub(sdk.NewDecCoinsFromCoins(rewardsToSend...))
```

- This code could be simplified:

```
return !iterator.Valid()
```

- And this one too.
- Combine code into a single function: `StopConsumerChain`
  The code regarding cleaning the Partial Set Security data could be moved into single function for better readability.
- In the function `CapValidatorSet`, the variable `minNumberOfValidators` is misleading as it represents the precise number of validators, not just a minimum number.
- The function `FilterOptedInAndAllowAndDenylistedPredicate` serves as a predicate to check whether a given validator meets certain criteria. However, naming it as a "Filter" may be misleading.
- The comment in the function `ComputeConsumerTotalVotingPower` incorrectly states that it sums the voting powers of opted-in validators. Instead, it sums the voting powers of the actual validator set.
- DeleteValidatorsPowerCap function should be called only to clean the state, and this is something that should be commented in the code to prevent any misuse.
- Update the comment to say that PSS clean-up was added to the code (the spec mentioned in the comment doesn't capture that).

Problem Scenarios

Findings listed above could not introduce any issues, they are suggestions for code improvements as well for some additional logging and inline code improvements, for easier understanding and readability of the code.

Recommendation

As described above.

Vulnerability Classification

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

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

Impact Score

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

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

Exploitability Score

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

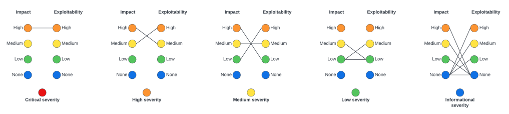
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

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

Exploitability Score	**Examples**
🟠 **High**	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
🟡 **Medium**	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
🟢 **Low**	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
🔵 **None**	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

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



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

Severity Score	**Examples**
🔴 **Critical**	Halting of chain via a submission of a specially crafted transaction
🟠 **High**	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
🟡 **Medium**	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users

Severity Score	**Examples**
🟢 **Low**	2x increase in node computational requirements via coordinated withdrawal of all user tokens
🔵 **Informational**	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/ incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer