Security Audit Report

Babylon

v1.0

June 12, 2025

Table of Contents

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.

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 Babylon Labs Ltd. to perform a security audit of several updates for Babylon Genesis.

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 target:

Repository	https://github.com/babylonlabs-io/babylon
Label	Paths referencing this target are prefixed below with `babylon:`
Scope	The scope is restricted to: • The changes applied between `v1.1.x` and `v2.x` in https://github.com/babylonlabs-io/babylon/compare/release/v1.1.x...release/v2.x, reviewed at commit `00763782f728b8a5d4c96d50d3c49be76e33b13b`, base branch at `f0a29d60f206268b56992fa50f38a48077eb4f59`.

	• The `x/incentive` module reviewed at commit `d95f863e44cd9c0f8279e04e204da60b1112b070`
Fixes verified at commit	`3da7d458efddd4f03013cc082f1b4c6cd979ad3c` 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/babylonlabs-io/finality-provider
Label	Paths referencing this target are prefixed below with `finality-provider:`
Scope	The scope is restricted to the changes applied in the following pull requests: • https://github.com/babylonlabs-io/finality-provider/pull/462 reviewed at commit `32eea898b7627706af954db33003de9419d626c9`, base branch at `6bc3ef37e07b6a1ffe468ce6f9f5c66bd79a9c61`.
Fixes verified at commit	`0de7af7d65b5a7e9201482031622522deade10ce` 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/babylonlabs-io/vigilante
Label	Paths referencing this target are prefixed below with `vigilante:`
Scope	The scope is restricted to the changes applied in the following pull requests: • https://github.com/babylonlabs-io/vigilante/pull/345 reviewed at commit `cb09aef7f3bf7e11b5dbf5dbf15e9c7925b69cde`, base branch at `61880560cba31e80cfe3aad7e895418c340c3598`.
Fixes verified at commit	`240b199dbc17142a641025b703b80139ed247702` 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

Babylon Genesis is a Cosmos SDK-based blockchain that provides two core security-sharing protocols between Bitcoin and Proof-of-Stake networks:

- The Bitcoin timestamping protocol periodically checkpoints Babylon Genesis' state on the Bitcoin blockchain.

- The Bitcoin staking protocol enables Bitcoin holders to provide economic security to decentralized systems through trustless staking via specialized Bitcoin scripts with time-locked transactions and slashing conditions enforced through Extractable One-Time Signatures (EOTS).

Babylon Genesis maintains a Bitcoin light client for transaction verification, uses Vigilante to relay data between Bitcoin and Babylon Genesis, and supports IBC protocols to extend Bitcoin's security guarantees to connected blockchain networks.

The scope of the audit is restricted to the codebase submitted for the audit.

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	The codebase comprises several interconnected components, including the Cosmos SDK appchain, Bitcoin Scripts, integration of third-party Cosmos modules, and the Vigilante relayer.
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	The client provided detailed documentation and diagrams.
Test coverage	Medium	`go test` reports the following test coverages for the repositories in scope: • `babylon`: 55.60% • `finality-provider`: 31.60% • `vigilante`: 37.60%

Summary of Findings

No	Description	Severity	Status
1	Missing TLS credentials and HMAC key in gRPC client enables credential compromise and MITM attacks	Major	Resolved
2	Retrieving staking transactions using incorrect page increments results in skipped transactions	Major	Resolved
3	Missing message size enforcement in `DeliverTx` enables oversized IBC messages payload injection	Major	Resolved
4	Unbounded growth in historical finality provider rewards leads to state bloat	Minor	Acknowledged
5	Insufficient validation of genesis' `RefundableMsgHashes` allows malformed entries	Minor	Resolved
6	Unsynchronized EOTS private key retrieval causes potential panic and memory corruption	Minor	Resolved
7	Passphrase handling in command line arguments	Minor	Resolved
8	Incomplete BLS key validation	Minor	Resolved
9	Usage of deprecated `x/crisis` module allows attackers to DoS the chain	Minor	Resolved
10	Plaintext password storage for BLS key decryption undermines key confidentiality	Minor	Acknowledged
11	Unbounded state growth due to unpruned BTC staking gauges	Minor	Acknowledged
12	Missing length validation of `TransactionKey.Hash` in the `x/btccheckpointing` module	Minor	Resolved
13	Incomplete validation of `RawCheckpointWithMeta` permits inconsistent genesis configuration	Minor	Resolved
14	Unvalidated genesis state in x/mint module leads to chain halt risks	Minor	Resolved
15	The `x/btcstaking` module `GenesisState` lacks complete validation	Minor	Resolved

16	Incorrect active finality provider validation in the `x/finality` module `GenesisState`	Minor	Resolved
17	Incomplete validation of `FinalityProviderDistInfo` allows for loss of commission and rewards	Minor	Resolved
18	Lack of validation for consecutive `EpochNumber` assignments at genesis	Minor	Resolved
19	Missing validation of the `Power` field of a `Validator` may result in potential overflows	Minor	Resolved
20	Partial validation of `Evidence` allows invalid public keys and signatures	Minor	Resolved
21	Invalid `LargesBtcReOrg` set at genesis may result in an immediate chain halt	Minor	Resolved
22	Missing validation of `FinalityProviderSigningInfo` permits unexpected behavior	Minor	Resolved
23	Incomplete passphrase validation for keyring backends creates a security risk	Minor	Resolved
24	Missing `Period` validation allows invalid rewards tracking state	Minor	Resolved
25	Insufficient `EventsPowerUpdateAtHeight` validation allows negative amounts to corrupt power calculations	Minor	Resolved
26	Insufficient genesis validation allows invalid event tracker height configuration	Minor	Resolved
27	Missing validation allows gaps in historical rewards and invalid tracker start periods	Minor	Resolved
28	Fee grant allowance not restored during refunds causes silent grant depletion	Minor	Acknowledged
29	Upgrade handler channel rate limiting enables denial of service via channel spam	Minor	Resolved
30	Unused `FpSlashed` function causes unnecessary state growth for slashed finality providers	Informational	Acknowledged
31	Overlapping block scanning leads to performance degradation	Informational	Acknowledged
32	Inefficient handling of failing events	Informational	Resolved

33	Inconsistent error message for passphrase flag	Informational	Resolved
34	Misleading error message for keyring backend validation	Informational	Resolved
35	Conflicting tokenfactory conditions	Informational	Resolved
36	Remove redundant address length validation	Informational	Resolved
37	Inefficient query implementation with mixed state functions	Informational	Acknowledged
38	Prevent setting blocked addresses as withdrawal addresses for incentive rewards	Informational	Resolved
39	Unvalidated `DelegationStateUpdate` entries permit arbitrary validator address values	Informational	Resolved
40	Potential `nil` pointer dereference when validating `BlsMultiSig`	Informational	Resolved
41	The `MaxAddressSize` constant is excessively large	Informational	Resolved
42	Lack of sanity checks before subtraction allows negative `TotalActiveSat` value	Informational	Resolved
43	Misleading comment for `SetRewardTrackerEvent`	Informational	Resolved
44	Unresolved TODOs in the codebase	Informational	Acknowledged

Detailed Findings

1. Missing TLS credentials and HMAC key in gRPC client enables credential compromise and MITM attacks

Severity: Major

In `finality-provider:eotsmanager/cmd/eotsd/daemon/unlock.go:36-71`, the `unlockKeyring` function initializes a gRPC client connection using the `eotsclient.NewEOTSManagerGRpcClient` method while explicitly passing an empty string as the HMAC key.

This disables HMAC-based client authentication. Although the function `ProcessHMACKey` emits a warning, it permits the connection to proceed without enforcing any authentication.

Furthermore, the gRPC connection is configured with `insecure.NewCredentials`, resulting in unencrypted and unauthenticated communication.

Consequently, the `Unlock` method transmits sensitive data, including the EOTS public key and user passphrase, without confidentiality or integrity protections.

This introduces the risk of credential compromise or man-in-the-middle (MITM) attacks.

Recommendation

We recommend enforcing strict authentication and secure transport in gRPC connections by:

- Replacing `insecure.NewCredentials` with credentials configured with proper TLS certificates.
- Ensuring a valid and securely stored HMAC key is passed to `NewEOTSManagerGRpcClient`, thereby enabling the `HMACUnaryClientInterceptor`.

Status: Resolved

The client fixed the issue in #487.

2. Retrieving staking transactions using incorrect page increments results in skipped transactions

Severity: Major

In `vigilante:btcstaking-tracker/stakingeventwatcher/stakingeventwatcher.go:995`, the `fetchStakingTxsByEvent` function retrieves staking transaction hashes

via the paginated query function `StakingTxHashesByEvent`, providing `i` and `batchSize` as the page and page-entry-limit parameters, respectively.

However, after each call, `i` is incorrectly increased by `batchSize` instead of being incremented by `1`. If more than `batchSize` staking transactions exist between `startHeight` and `endHeight`, up to `batchSize` pages of transactions are skipped. As a result, the watcher silently omits delegations and never revisits those heights.

Recommendation

We recommend incrementing `i` by `1` after each fetch operation.

Status: Resolved

The client fixed the issue in #359.

3. Missing message size enforcement in `DeliverTx` enables oversized IBC messages payload injection

Severity: Major

In `babylon:app/ante/ibc_msg_size.go:25-44`, the `AnteHandle` function of the `IBCMsgSizeDecorator` validates IBC message size constraints only during the `CheckTx` phase.

This validation includes limits on message, memo, and address sizes for `MsgTransfer` and `MsgSendTx` messages.

However, no equivalent enforcement occurs during the `DeliverTx` phase, which is responsible for block execution.

As a result, malicious proposers can bypass mempool checks and inject oversized IBC or ICA messages directly into blocks without the risk of being slashed.

This forces other validators to process potentially resource-exhausting payloads, introducing a denial of service vector within the consensus mechanism.

Recommendation

We recommend extending size validations to the `DeliverTx` phase.

Status: Resolved

The client fixed the issue in #1081.

4. Unbounded growth in historical finality provider rewards leads to state bloat

Severity: Minor

The `IncrementFinalityProviderPeriod` function in `babylon:x/incentive/keeper/reward_tracker.go:214-256` persistently stores historical reward records in the `finalityProviderHistoricalRewards` map.

The stored data, represented by `newFpHistoricalRwd`, includes cumulative reward values that grow monotonically due to repeated additions of current reward per satoshi. This storage pattern is triggered by common operations such as delegation updates, staking changes, and reward withdrawals

However, no mechanism exists to remove old records that are no longer needed to process rewards withdrawal or delegation modifications.

As a result, the blockchain state could expand unboundedly and increase storage costs and requirements for validators.

Recommendation

We recommend implementing a pruning strategy that retains only necessary historical data used in current reward calculations.

Status: Acknowledged

The client acknowledges this finding without a need for immediate resolution as it involves an optimization in storage. The inefficiency is not directly exploitable.

The client plans to address it in an upcoming release and has created #812 to track this.

5. Insufficient validation of genesis' `RefundableMsgHashes` allows malformed entries

Severity: Minor

The `validateMsgHashes` function in `babylon:x/incentive/types/genesis.go:226-238` verifies that each hash in the `RefundableMsgHashes` list is non-empty and unique.

However, it lacks enforcement of structural or length constraints on the hash values. This permits malformed or arbitrarily long strings to be included in the genesis file.

Recommendation

We recommend strengthening the validation logic by enforcing at least a maximum allowed length for each hash.

Status: Resolved

The client fixed the issue in #1066.

6. Unsynchronized EOTS private key retrieval causes potential panic and memory corruption

Severity: Minor

In the `getKeyFromMap` function in `finality-provider:eotsmanager/localmanager.go:402-414`, the code accesses the `privateKeys` map without holding the required mutex lock.

The function assumes callers always hold the `LocalEOTSManager.mm` mutex, but the callstack `SignSchnorrSigFromKeyname`, `eotsPrivKeyFromKeyName`, `getKeyFromMap` violates this assumption.

Concurrent access to Go maps triggers runtime panics, creating a denial-of-service vector for the `eotsd` daemon, and in Go runtime bug scenarios, could cause memory corruption, leading to private key exposure.

We classify this as Minor because currently the `SignSchnorrSigFromKeyname` function is only called from a CLI command handler, and the risk of concurrent access is negligible. It requires immediate attention, however, as an unaware future maintainer may use the function in a concurrent context.

Recommendation

We recommend acquiring the `LocalEOTSManager.mm` mutex within `getKeyFromMap` or document and enforcing the locking requirement in all calling functions to prevent concurrent map access.

Status: Resolved

The client fixed the issue in #480.

7. Passphrase handling in command line arguments

Severity: Minor

In the `unlockKeyring` function, defined in `finality-provider:eotsmanager/cmd/eotsd/daemon/unlock.go:36`, the passphrase is passed as a command-line argument.

However, this could expose the passphrase in shell history.

While the daemon is likely used in controlled environments, it is best practice to avoid passing sensitive information via command-line arguments.

Recommendation

We recommend using environment variables or secure input methods for passphrase handling.

Status: Resolved

The client fixed the issue in #487.

8. Incomplete BLS key validation

Severity: Minor

In `babylon:x/checkpointing/types/bls_key.go:29`, the `ValidatorWithBlsKeySet.Validate` function checks if the BLS key can be unmarshaled, but does not verify if it is a valid point on the BLS12-381 curve.

This could potentially allow invalid public keys to be accepted and unexpected errors or results when those keys are used.

Recommendation

We recommend adding explicit validation of the BLS public key point.

Status: Resolved

The client fixed the issue in #1129.

9. Usage of deprecated `x/crisis` module allows attackers to DoS the chain

Severity: Minor

The Babylon Genesis chain currently employs the `x/crisis` module to allow any participant to halt the chain in the event of an invariant violation by sending a `MsgVerifyInvariant`

message. This mechanism is intended to increase the robustness of the network by enabling the detection of critical inconsistencies.

However, the module is deprecated as indicated in GHSA-qfc5-6r3j-jj22 and GHSA-w5w5-2882-47pc because it fails to induce a panic within transaction processing, thus treating broken invariants as reverted transactions.

Additionally, as reported in the CVEs, processing `MsgVerifyInvariant` messages incurs significant computational overhead, while the fee does not align with the computational demand, making these transactions cheaper relative to their processing cost.

Recommendation

We recommend removing the `x/crisis` module. Instead, simulation tests should be enhanced, and the implementation of the `x/circuit` module could be evaluated.

Status: Resolved

The client fixed the issue in #1156.

10. Plaintext password storage for BLS key decryption undermines key confidentiality

Severity: Minor

In `babylon:cmd/babylond/cmd/verify_val_bls_key.go:20`, the `VerifyValidatorBlsKey` command accepts a `--bls-password-file` flag, allowing users to supply a password for decrypting the BLS key from a file.

However, this file is expected to store the password in plaintext, negating the security benefits of key encryption. Any attacker with access to the local filesystem can trivially read the password and decrypt the associated BLS key.

This creates a false sense of security while effectively negating encrypted key storage with plaintext password exposure and introducing significant risks in validator setups.

Recommendation

We recommend eliminating reliance on plaintext password files.

Status: Acknowledged

The client acknowledges this finding with the note that plaintext storage is only one of the options for specifying the BLS password, with more secure options available.

The client plans to deprecate plaintext file support and help users migrate in future releases. Issue to track for the resolution: #1165

11. Unbounded state growth due to unpruned BTC staking gauges

Severity: Minor

The `RewardBTCStaking` function in `babylon:x/incentive/keeper/btc_staking_gauge.go:19` maintains a new BTC staking gauge for each height but never removes old gauges after they are used.

However, this leads to unbounded state growth as the chain progresses, since each height's gauge data remains in the state store indefinitely.

Recommendation

We recommend pruning the staking gauges after they are used.

However, the removal of previous gauges may impact the `QueryBTCStakingGaugeRequest` query, so it may also be required to update the query to note that it will not return inactive gauges.

Status: Acknowledged

The client acknowledges this finding as an optimization involving the pruning of historical states without a direct exploitability.

The client plans to address it in upcoming releases. Issue to track for the resolution: #812.

12. Missing length validation of `TransactionKey.Hash` in the `x/btccheckpointing` module

Severity: Minor

In `babylon:x/btccheckpoint/types/types.go:207-212`, the `TransactionKey.Validate` method verifies that the `Hash` field is not `nil`.

However, it does not verify the length of the `Hash`, which should match the expected size of the Bitcoin block hash.

This omission potentially leads to state corruption if an invalid genesis file is used.

Recommendation

We recommend validating that the length of the `Hash` field matches the expected size of a Bitcoin block hash within the `Validate` method.

Status: Resolved

The client fixed the issue in #1099.

13. Incomplete validation of `RawCheckpointWithMeta` permits inconsistent genesis configuration

Severity: Minor

In `babylon:x/checkpointing/types/checkpoint.go:5-19`, the `Validate` method validates the `Ckpt` and `BlsMultiSig` fields.

However, validation for other fields like `Status` and `BlsAggrPk` is missing. This allows the genesis configuration to bypass constraints enforced during runtime operations, potentially leading to inconsistencies.

Consequently, an improperly configured genesis file could result in a checkpoint with an invalid state, such as a sealed checkpoint without a valid BLS aggregation public key.

Additionally, the call to `BlsMultiSig.ValidateBasic` is made twice if the field is not `nil`, first when calling `Ckpt.ValidateBasic` and then again straight afterwards.

Recommendation

We recommend expanding the `Validate` method to encompass all fields within `RawCheckpointWithMeta`, ensuring all invariants are enforced during genesis initialization.

Additionally, we recommend removing the second call to `BlsMultiSig.ValidateBasic`.

Status: Resolved

The client fixed the issue in #1120.

14. Unvalidated genesis state in `x/mint` module leads to chain halt risks

Severity: Minor

In `babylon:x/mint/keeper/genesis.go:9-14`, the `InitGenesis` function does not validate the `Minter` field of the `GenesisState` before committing it to storage. If the `Minter` configuration is invalid, it may lead to zero rewards being produced per block.

This would break the invariant in the `x/incentive` module's `RewardBTCStaking` function, called within the `x/finality` module `EndBlocker`, which will panic and halt the chain if there is no reward gauge set for height being processed.

A reward gauge is only set if a non-zero amount of minted rewards is intercepted by the `HandleCoinsInFeeCollector` function in `babylon:x/incentive/keeper/intercept_fee_collector.go:13-35`.

Recommendation

We recommend calling the `GenesisState.Validate` method during the `InitGenesis` function to ensure the `Minter` is properly configured, as well as validating other genesis state fields.

Additionally, to defend against chain halts, set an empty rewards gauge in the `HandleCoinsInFeeCollector` in the event of zero rewards being intercepted.

Status: Resolved

The client fixed the issue in #1018.

15. The `x/btcstaking` module `GenesisState` lacks complete validation

Severity: Minor

In `babylon:x/btcstaking/types/genesis.go:24-53`, the `GenesisState.Validate` function checks the `Params`, `LargestBtcReorg` and `AllowedStakingTxHashes` fields.

However, it lacks validation for the `FinalityProviders`, `BtcDelegations`, `BlockHeightChains`, `BtcDelegators`, and `Events` fields.

This allows a genesis state to be committed that may be inconsistent with the application logic, potentially leading to unexpected behavior.

Recommendation

We recommend validating all `GenesisState` fields where it is possible to do so.

Status: Resolved

The client fixed the issue in #1123.

16. Incorrect active finality provider validation in the `x/finality` module `GenesisState`

Severity: Minor

In `babylon:x/finality/types/power_table.go:145-153`, the `VotingPowerDistCache.Validate` function validates that `NumActiveFps` is not greater than the total number of finality providers.

However, this does not accurately reflect the number of active finality providers, which should only include those that are not jailed, slashed, or have a current delegation of zero sats.

Consequently, rewards may be distributed to undeserving finality providers if the invalid `VotingPowerDistCache` is used in the execution of the `RewardBTCStaking` function.

Recommendation

We recommend updating the validation logic to accurately reflect the number of active finality providers that meet the criteria for reward distribution.

Status: Resolved

The client fixed the issue in #1072.

17. Incomplete validation of `FinalityProviderDistInfo` allows for loss of commission and rewards

Severity: Minor

In `babylon:x/finality/types/power_table.go:213-222`, the `FinalityProviderDistInfo.Validate` method only validates the `BtcPk` field.

However, the `Addr` field, representing the finality provider's address, and the `Commission` field are not validated.

This omission allows for the potential setting of an invalid address at genesis, which could lead to a loss of commission payments, and a `Commission` value exceeding `1.0`, which would result in a reduction in delegator rewards by passing negative `Coins` amounts to `AddFinalityProviderRewardsForBtcDelegations` in `babylon:x/incentive/keeper/btc_staking_gauge.go:66`

Recommendation

We recommend validating that the `Addr` field is a valid Bech32 address and that the `Commission` field value is between `0` and `1.0` in the `FinalityProviderDistInfo.Validate` function.

Status: Resolved

The client fixed the issue in #1109.

18. Lack of validation for consecutive `EpochNumber` assignments at genesis

Severity: Minor

In `babylon:x/epoching/types/genesis.go:83-112`, the `validateEpochs` function verifies the uniqueness of `EpochNumber`, `FirstBlockHeight`, and `SealerBlockHash`.

However, it does not validate that `EpochNumber` values are assigned in a consecutive sequence.

Consequently, the invariant that a previous epoch always exists, which is assumed by the `RecordSealerAppHashForPrevEpoch` function in `babylon:x/epoching/keeper/epochs.go:119-123`, may be broken and could result in a chain halt.

Recommendation

We recommend incorporating a check within the `validateEpochs` function to ensure that `EpochNumber` values increment sequentially, thereby guaranteeing a complete epoch history upon chain initialization.

Status: Resolved

The client fixed the issue in #1086.

19. Missing validation of the `Power` field of a `Validator` may result in potential overflows

Severity: Minor

In `babylon:x/epoching/types/epoching.go:211-215`, the `Validator.Validate` function does not check if the `Power` field is non-negative.

A negative `Power` would result in a series of silent overflows when the value is cast to an unsigned integer in the `Accumulate` function in `babylon:x/checkpointing/types/types.go108-111`, resulting in an incorrect checkpoint status.

Recommendation

We recommend implementing a validation check to ensure the `Validator.Power` field is non-negative.

Status: Resolved

The client fixed the issue in #1153.

20. Partial validation of `Evidence` allows invalid public keys and signatures

Severity: Minor

In `babylon:x/finality/types/finality.go:87-104`, the `Evidence` entries are not comprehensively validated for expected field lengths.

Specifically, the code does not verify that the `FpBtcPk` and `ForkFinalitySig` fields within the `Evidence` struct are valid public keys and signature lengths, respectively.

As the `FpBtcPk` field is also used in the storage key for the `Evidence` entry, a corrupted value in a genesis file may result in the a finality provider not being slashed if they submit a second finality signature for the same block height, as `HasEvidence` will return false in `x/finality/keeper/msg_server.go:194` when called with the non-corrupted `FpBtcPk` value.

Recommendation

We recommend implementing length checks for `FpBtcPk` and `ForkFinalitySig` during `Evidence` validation.

Status: Resolved

The client fixed the issue in #1103.

21. Invalid `LargesBtcReOrg` set at genesis may result in an immediate chain halt

Severity: Minor

In `babylon:x/btcstaking/types/btcstaking.go:125-147`, the `LargestBtcReOrg.Validate` function does not check the `BlockDiff` value against the `RollbackFrom` and `RollbackTo` heights.

Consequently, if an invalid `LargestBtcReOrg` entry is set with a `BlockDiff` that is larger than the configured `Params.BtcConfirmationDepth` value, the chain will halt the first time the `HaltIfBtcReorgLargerThanConfirmationDepth` function is run in the `x/btcstaking` module `EndBlocker`.

Recommendation

We recommend validating the `BlockDiff` field by checking if it equals the difference between `RollbackFrom.Height` and `RollbackTo.Height` as a form of stateless validation that matches runtime creation logic in `babylon:x/btcstaking/types/btcstaking.go:119`.

Additionally, we recommend verifying that the value set at genesis is less than the `Params.BtcConfirmationDepth` value, ensuring that the chain will not immediately halt.

Status: Resolved

The client fixed the issue in #1115.

22. Missing validation of `FinalityProviderSigningInfo` permits unexpected behavior

Severity: Minor

In `babylon:x/finality/types/signing_info.go:43-51`, the `FinalityProviderSigningInfo.Validate` method validates the `FpBtcPk` field.

However, the `StartHeight` and `MissedBlockCounter` fields, which are of type `int64`, are not verified to be non-negative.

Consequently, if negative values are committed from a malformed genesis state, the finality provider liveness logic, which operates on those fields, may produce unexpected results.

Recommendation

We recommend validating that `StartHeight` and `MissedBlockCounter` are greater than or equal to zero in the `FinalityProviderSigningInfo.Validate` function.

Status: Resolved

The client fixed the issue in #1075.

23. Incomplete passphrase validation for keyring backends creates a security risk

Severity: Minor

In `finality-provider:eotsmanager/localmanager.go:110`, the passphrase length check only validates for `keyring.BackendFile`, missing validation for `keyring.BackendOS`, which could lead to weak passphrases being accepted when using `keyring.BackendOS`.

The restriction of keyring backends to `"test"` or `"file"` is only enforced in `finality-provider:eotsmanager/cmd/eotsd/daemon/start.go:41-43`, as the config validation in `finality-provider:eotsmanager/config/config.go:90-92` only checks that the `KeyringBackend` field is not an empty string.

Recommendation

We recommend moving both the backend validation and passphrase requirements to the config validation function in `finality-provider:eotsmanager/config/config.go`.

Status: Resolved

The client fixed the issue in #489.

24. Missing `Period` validation allows invalid rewards tracking state

Severity: Minor

In `babylon:x/incentive/types/rewards.go:82-101`, the `Validate` function does not verify that the `Period` field is greater than zero.

This omission allows `FinalityProviderCurrentRewards` objects with zero or negative periods to be stored, causing unsigned integer underflows in `babylon:x/incentive/keeper/reward_tracker.go:238,288` when calculating the previous `Period` in reward tracking operations.

Recommendation

We recommend adding a validation check in the `Validate` method to ensure the `Period` field is greater than zero.

Status: Resolved

The client fixed the issue in #1137.

25. Insufficient `EventsPowerUpdateAtHeight` validation allows negative amounts to corrupt power calculations

Severity: Minor

In `babylon:x/incentive/types/rewards.go:135-158`, the `EventsPowerUpdateAtHeight` validation checks finality provider and delegator addresses but fails to verify that `TotalSat` values are positive.

This oversight could allow events with negative `TotalSat` values from the genesis state to be processed, causing `BtcActivated` events to decrease total staked amounts and `BtcUnbonded` events to increase them, inverting their intended behavior and corrupting the staking power calculations.

Recommendation

We recommend validating that the `TotalSat` field is greater than zero for each event during the validation process.

Status: Resolved

The client fixed the issue in #1112.

26. Insufficient genesis validation allows invalid event tracker height configuration

Severity: Minor

In `babylon:x/incentive/keeper/genesis.go:115-117`, the `InitGenesis` function sets `LastProcessedHeightEventRewardTracker` without validating it against the current chain height.

This allows genesis configurations where the tracker height equals or exceeds the current height, causing reward events to remain unprocessed until the blockchain surpasses the misconfigured value.

Recommendation

We recommend validating that `LastProcessedHeightEventRewardTracker` is less than the current block height before setting it.

Status: Resolved

The client fixed the issue in #1140.

27. Missing validation allows gaps in historical rewards and invalid tracker start periods

Severity: Minor

In the `validateFPHistoricalRewards` and `validateBTCDelegationsRewardsTrackers` functions in `babylon:x/incentive/types/genesis.go:240-286`, the validation logic fails to verify that finality providers have historical rewards entries for every period from `0` to `FinalityProviderCurrentRewards.Period - 1`, and that delegation tracker `StartPeriodCumulativeReward` values are less than the current rewards period.

This incomplete validation allows genesis states with missing historical periods or tracker start periods exceeding the current period, causing reward calculation failures and incorrect reward distributions.

Recommendation

We recommend adding validation to ensure each finality provider has historical rewards entries for all periods from `0` through `FinalityProviderCurrentRewards.Period - 1`, and that all tracker entries have `StartPeriodCumulativeReward` values less than the current rewards period.

Status: Resolved

The client fixed the issue in #1132.

28. Fee grant allowance not restored during refunds causes silent grant depletion

Severity: Minor

In `babylon:x/incentive/keeper/refundable_msg_index.go:10-29`, the `RefundTx` function processes fee refunds by transferring tokens from the fee collector module back to the transaction's fee payer.

When a fee grant is used, the Cosmos SDK correctly deducts the fee from the granter's allowance and transfers the tokens.

However, during the refund, while the tokens are returned to the granter, the allowance consumed from the fee grant is not reinstated.

This results in silent depletion of the fee grant, effectively reducing the granter's usable allowance without actual transaction fee expenditure.

Over time, this inconsistency can restrict the granter's ability to fund future transactions.

Recommendation

We recommend restoring the consumed fee grant allowance during the refund process.

Status: Acknowledged

The client acknowledges this issue and plans to resolve it in an upcoming release.

Issue to track for the resolution: #1167.

29. Upgrade handler channel rate limiting enables denial of service via channel spam

Severity: Minor

In `babylon:app/upgrades/v2/upgrades.go:98-109`, the upgrade handler enforces rate limiting across all channels.

However, this behavior allows a malicious actor to preemptively create a large volume of fake channels prior to the upgrade. As these channels are automatically included in the rate-limiting set, the system attempts to process them during the next epoch hour reset.

Consequently, this mass processing may exceed the permissible block execution time, triggering a denial of service (DoS) condition and halting the chain.

Although gas costs and counterparty chain involvement pose natural barriers to channel creation, the threat remains viable under specific conditions.

Recommendation

We recommend using a whitelist or governance process to enable rate limiting only for specific channels.

Status: Resolved

The client fixed the issue in #1162.

30. Unused `FpSlashed` function causes unnecessary state growth for slashed finality providers

Severity: Informational

In `babylon:x/incentive/keeper/reward_tracker.go:52-77`, the `FpSlashed` function handles reward distribution and state cleanup for slashed finality providers and their delegators.

However, this function is never invoked. When a finality provider is slashed, the system maintains incorrect `TotalActiveSat` values equal to pre-slash amounts in `BTCDelegationRewardsTracker`, `FinalityProviderHistoricalRewards`, and `FinalityProviderCurrentRewards` storage.

In addition to producing misleading `TotalActiveSat` values in query results, subsequent calls to `sendAllBtcDelegationTypeToRewardsGauge` during reward withdrawals cause ongoing state growth by redundantly creating new storage entries for slashed finality providers.

Recommendation

We recommend adding an `EventPowerUpdate_Slashed` event handler that creates a final `FinalityProviderHistoricalRewards` entry with the pre-slash `TotalStakedSat` amount, removes the `FinalityProviderCurrentRewards` entry, and flags the finality provider as slashed to prevent further state growth during reward withdrawals.

Status: Acknowledged

The client acknowledges this as a valid informational issue as it affects state growth.

They plan to address this in future releases and will track it in #1166.

31. Overlapping block scanning leads to performance degradation

Severity: Informational

In `vigilante:btcstaking-tracker/stakingeventwatcher/stakingeventwatcher.go:156-177`, the `Start` function launches concurrent goroutines to fetch delegations and blocks from CometBFT.

However, while `fetchDelegations` traverses the blockchain from genesis up to a dynamic, event-driven boundary (based on batch size), `fetchCometBftBlockForever` begins from a static height retrieved once via `CometBFTTipHeight` and stored in `currentCometTipHeight`. This results in overlapping block range traversals between the two Go routines.

Consequently, the same block heights may be redundantly processed, triggering repeated event queries and unnecessary handler invocations, which would degrade performance.

Recommendation

We recommend processing each height only once between multiple goroutines.

Status: Acknowledged

The client acknowledges this as an informational issue that does not require an immediate resolution as the traversal overlap is negligible.

They have a partial fix in #369 and plan to fully address this in an upcoming release.

32. Inefficient handling of failing events

Severity: Informational

In `vigilante:btcstaking-tracker/stakingeventwatcher/stakingeventwatcher.go:945`, the `fetchDelegationsByEvents` function processes delegation events in a loop but exits immediately if any single event processing fails.

Similarly, in `vigilante:btcstaking-tracker/stakingeventwatcher/stakingeventwatcher.go:963-996`, the `fetchStakingTxsByEvent` function processes staking transactions in batches but has no error recovery mechanism.

However, in a production environment, this could lead to a single problematic event (e.g., RPC timeout, invalid data) blocking the processing of all subsequent events.

This could lead to increased load on the Babylon node during retries and to an inefficient processing.

Recommendation

We recommend implementing error handling that allows processing to continue despite individual event failures, tracking failed events for separate retry mechanisms, adding monitoring for failed event rates, and considering a maximum retry limit for failed events.

Status: Resolved

The client fixed the issue in #370.

33. Inconsistent error message for passphrase flag

Severity: Informational

In the `unlockKeyring` function, defined in `finality-provider:eotsmanager/cmd/eotsd/daemon/unlock.go:45-47`, the error message incorrectly references `"chain-id flag"` when it should be `"passphrase flag"`.

Developers might be confused when debugging issues related to passphrase handling.

Recommendation

We recommend updating the error message to correctly reference the correct flag.

Status: Resolved

The client fixed the issue in #487.

34. Misleading error message for keyring backend validation

Severity: Informational

In the `startFn` function, defined in `finality-provider:eotsmanager/cmd/eotsd/daemon/start.go:41-43`, the error message states that the keyring backend must be `"test"` for automatic signing, but the code actually accepts both `"test"` and `"file"` backends.

This creates confusion for users and developers. The keyring backend options have been updated, but the error message has not been updated to reflect this.

Recommendation

We recommend updating the error message to accurately reflect that both `"test"` and `"file"` backends are supported.

Status: Resolved

The client fixed the issue in #482.

35. Conflicting tokenfactory conditions

Severity: Informational

In `babylon:app/keepers/keepers.go:598`, the `DefaultIsSudoAdminFunc` effectively disables sudo functionality for all addresses.

However, in `babylon:app/keepers/keepers.go:111`, token factory capabilities are set.

While this is not altering the expected behavior, those conditions are conflicting.

Recommendation

We recommend resolving this conflict based on the intended configuration.

Status: Resolved

The client fixed the issue in #1036.

36. Remove redundant address length validation

Severity: Informational

In `babylon:app/params/config.go:80`, the `SetAddressVerifier` function performs two consecutive address length validations:

- A check against `address.MaxAddrLen` (255 bytes)
- A stricter check requiring either 20 or 32 bytes

The first check is redundant since any address that passes the second check (20 or 32 bytes) will automatically satisfy the first check (≤ 255 bytes).

Recommendation

We recommend removing the redundant validation.

Status: Resolved

The client fixed the issue in #1170.

37. Inefficient query implementation with mixed state functions

Severity: Informational

In `babylon:x/incentive/keeper/grpc_query.go:28,78`, the `Rewards` query handler is calling `sendAllBtcDelegationTypeToRewardsGauge`, which attempts state modifications.

While these modifications won't be committed (due to Cosmos SDK's query context being read-only), this is still problematic because it's inefficient and misleading.

The same function is used to make and commit state changes during other message types, which violates the principle of separation between state-modifying and read-only operations.

This mixing of concerns makes the code harder to maintain and could lead to confusion about the function's intended behavior in different contexts.

Recommendation

We recommend refactoring the code to separate the state-modifying logic from the read-only query logic, creating distinct functions for each use case. This would make the code's intent clearer and prevent any potential misuse of the functions in different contexts.

Status: Acknowledged

The client acknowledges this finding and plans to resolve it in an upcoming release.

Issue to track for resolution: #1169.

38. Prevent setting blocked addresses as withdrawal addresses for incentive rewards

Severity: Informational

In `babylon:x/incentive/keeper/msg_server.go:73-90`, the `SetWithdrawAddress` handler function for the `MsgSetWithdrawAddress` message stores the user-provided `withdrawAddress` as the recipient address for rewards. By default, the delegator address is used as the recipient address.

However, `withdrawAddress` might be a blocked address that is returned by the `BlockedAddresses` function in `babylon:app/app.go:860-870`. For instance, a module address would prevent withdrawing rewards, as sending the bank coins would fail in this case.

Since this only affects the individual delegator, can be reversed, and has no other impacts, we classify this issue as Informational.

Recommendation

We recommend returning an error if `withdrawAddress` is a blocked address.

Status: Resolved

The client fixed the issue in #1106.

39. Unvalidated `DelegationStateUpdate` entries permit arbitrary validator address values

Severity: Informational

In `babylon:x/epoching/types/epoching.go:225-231`, the `DelegationLifecycle.Validate` function checks for the presence of lifecycle entries.

However, the implementation fails to validate the individual `DelegationStateUpdate` entries, specifically the `ValAddr` field, allowing arbitrary values to be supplied at genesis.

While it does not appear that this data is used in any application logic, any invalid data will be returned by the `DelegationLifecycle` query handler in `babylon:x/epoching/keeper/grpc_query.go:198-208`.

Recommendation

We recommend validating each `DelegationStateUpdate` entry to ensure the `ValAddr` field represents a valid address.

Status: Resolved

The client fixed the issue in #1178.

40. Potential `nil` pointer dereference when validating `BlsMultiSig`

Severity: Informational

In `babylon:x/checkpointing/types/types.go:186-200` the `RawCheckpoint.ValidateBasic` function calls the `ValidateBasic` method on the `BlsMultiSig` field.

However, since it is a pointer, it could lead to a runtime panic if the pointer is `nil`.

Currently, the `RawCheckpoint.ValidateBasic` is only called at genesis, so the runtime panic will not result in failed transactions or chain halts, but it has the potential for both if used in other contexts by future maintainers.

Recommendation

We recommend adding a `nil` check before calling `ValidateBasic` on the `BlsMultiSig` field to prevent a potential runtime panic.

Status: Resolved

The client fixed the issue in #1118.

41. The `MaxAddressSize` constant is excessively large

Severity: Informational

In `babylon:app/ante/ibc_msg_size.go:15`, the `MaxAddressSize` constant is set to `65000` bytes.

This value is significantly larger than the maximum size of a Bech32 address, which is `90` characters, and allows for excessive amounts of data to be included in the address fields of IBC transfer and Interchain Account messages.

Recommendation

We recommend reducing `MaxAddressSize` to a more conservative value consistent with the maximum length of a Bech32 address.

Status: Resolved

The client fixed the issue in #1175.

42. Lack of sanity checks before subtraction allows negative `TotalActiveSat` value

Severity: Informational

In `babylon:x/incentive/keeper/reward_tracker_store.go:173-183`, the `subFinalityProviderStaked` function lacks sanity checks on the `amt` parameter before subtracting it from the finality provider's `TotalActiveSat` amount. The function trusts callers to provide an amount less than or equal to the current `TotalActiveSat`. This allows the invariant that `TotalActiveSat` must always be non-negative to be broken, which potentially causes incorrect rewards calculations in the rewards distribution logic.

We classify this as Informational because currently the only caller first checks that subtracting the amount parameter does not result in a negative `TotalActiveSats` value on a `BTCDelegationRewardsTracker` instance associated with the finality provider. As long as the invariant that the sum of all `TotalActiveSats` in associated rewards trackers equals the finality provider's `TotalActiveSats`, the call to `subFinalityProviderStaked` is safe.

Recommendation

We recommend adding a validation check to ensure the subtraction result remains non-negative and returning an error if the amount exceeds the current `TotalActiveSats` value, similar to the `subDelegationSat` implementation.

Status: Resolved

The client fixed the issue in #1094.

43. Misleading comment for `SetRewardTrackerEvent`

Severity: Informational

In `babylon:x/incentive/keeper/reward_tracker_events.go:137`, the comment for the `SetRewardTrackerEvent` function states that it returns a reward tracker, but only an `error` or `nil` is returned.

This may cause confusion for future maintainers and reviewers.

Recommendation

We recommend rewording the misleading comment.

Status: Resolved

The client fixed the issue in #1090.

44. Unresolved TODOs in the codebase

Severity: Informational

The codebase contains multiple TODO comments that highlight incomplete features, deferred improvements, or areas requiring further attention.

It is best practice to resolve todos before releasing the code into production.

Recommendation

We recommend reviewing and resolving outstanding TODO comments.

Status: Acknowledged

The client acknowledges the existence of TODOs in the codebase. The Babylon Genesis chain is an evolving chain with a roadmap of planned improvements.