Quantstamp Security Assessment Certificate

May 10th 2021 — Quantstamp Verified

Prysm - ETH 2.0 Client

This security audit was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Ethereum 2.0 client
Auditors	Kacper Bąk, Senior Research Engineer Alex Murashkin, Senior Software Engineer Ed Zulkoski, Senior Security Engineer Jan Gorzny, Blockchain Researcher Leonardo Passos, Senior Research Engineer Poming Lee, Research Engineer Martin Derka, Senior Research Engineer Sung-Shine Lee, Research Engineer Sebastian Banescu, Senior Research Engineer Kevin Feng, Software Engineer
Timeline	2020-04-29 through 2020-10-13
Languages	Go
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Prysm Codebase Overview GoDoc Phase 0 for Humans Phase 0 accompanying resource Ethereum 2.0 Terms Demystified
Documentation Quality	High
Test Quality	Medium

Source Code

Repository	Commit
prysm	7bea237

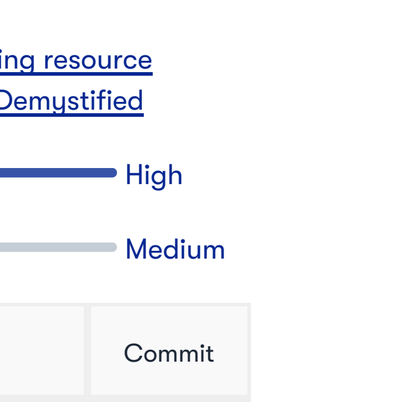
Goals

• Is the implementation vulnerable to DoS attacks?

• Does the implementation deviate from the specification?

• Does the implementation leak any sensitive data?

Total Issues	**58** (43 Resolved)
High Risk Issues	**4** (3 Resolved)
Medium Risk Issues	**12** (9 Resolved)
Low Risk Issues	**20** (14 Resolved)
Informational Risk Issues	**17** (13 Resolved)
Undetermined Risk Issues	**5** (4 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has reviewed the whole codebase of the Prysm ETH 2.0 client implementation. We have found a number of issues spanning all severity levels. Some of the high severity issues were resolved before completion of the review. Overall the code is well-written. There are many ways, however, in which it can be improved to follow best practices. For example, code clones are relatively common. Furthermore, despite being mostly self-documenting, inline code documentation is lacking. We have no doubt that it would be useful for future contributors. Despite being accompanied by the official ETH 2.0 documentation, the implementation is very nuanced and complex. The code does not always follow the specification (or it is not clear that it does). We found a number of issues that span both the specification and the implementation. Although we aggregated some of them in the "Adherence to Specification" section, they are of utmost importance and we highly recommend addressing each and every of them as if they were actual vulnerabilities.
Finally, many pieces of the code lack unit tests, and hence, relatively low coverage. We highly recommend adding meaningful unit tests and improving the coverage to maximize code quality.
Update: the team addressed almost all of the findings. Mapping between issues and solutions is present in PR#6327.

ID	Description	Severity	Status
QSP-1	The functions `IntersectionUint64()`, `IntersectionInt64()` and `IntersectionByteSlices()` may return union instead of intersection	High	Fixed
QSP-2	Potential issues due to granularity of timestamps	High	Fixed
QSP-3	Insecure gRPC connection by default	High	Acknowledged
QSP-4	`root` may exist as a subsequence	High	Fixed
QSP-5	Use of pseudo random number generator where true random number generator would be needed	Medium	Fixed
QSP-6	Second pre-image attack on Merkle Trees	Medium	Acknowledged
QSP-7	Function `ActivationEligibilityEpoch()` doesn't check `v == nil || v.validator == nil`	Medium	Fixed
QSP-8	Cast from `uint64` to `int`	Medium	Fixed
QSP-9	Possible loss of data integrity	Medium	Fixed
QSP-10	Checks before the lock	Medium	Fixed
QSP-11	DDoS attack vector through creating a mapping between public keys and validators' IPs	Medium	Acknowledged
QSP-12	Message encoding is changeable: that could lead to network partitions	Medium	Fixed
QSP-13	Opening BoltDB and backup file varies on the permission	Medium	Fixed
QSP-14	Lack of unit tests	Medium	Acknowledged
QSP-15	No support for IPv6	Medium	Fixed
QSP-16	Presence of the test-only code	Medium	Fixed
QSP-17	Possible parent block with same slot	Low	Fixed
QSP-18	No header length validation	Low	Fixed
QSP-19	Memory resources are never freed in `newBlocksFetcher()`	Low	Fixed
QSP-20	No meaningful code in `removeDisconnectedPeerStatus()`	Low	Mitigated
QSP-21	Disk space exhaustion attack	Low	Acknowledged
QSP-22	Disconnected, stale, or bad peer status records are not cleaned up	Low	Fixed
QSP-23	Connection manager options are not always initialized correctly	Low	Fixed
QSP-24	Boot nodes availability and centralization risks	Low	Acknowledged
QSP-25	Relay option is enabled in libp2p even when unused in Prysm	Low	Fixed
QSP-26	Potential discrepancy between `NextForkVersion` and `ForkVersionSchedule`	Low	Fixed
QSP-27	Subnet-based whitelisting not working	Low	Fixed
QSP-28	Rate limiting not implemented for some node communication	Low	Fixed
QSP-29	Peer descoring and disconnect on failed validation, while being optional in the spec, should be considered	Low	Acknowledged
QSP-30	Incorrect deadline for responses	Low	Fixed
QSP-31	Maximum response chunk size not checked for all topics	Low	Fixed
QSP-32	Number of bad responses is not incremented in the `Goodbye` topic	Low	Unresolved
QSP-33	`finalized_root` not checked in `Status` topic	Low	Fixed
QSP-34	Weak passwords allowed for validator accounts	Low	Fixed
QSP-35	Minimal system requirements	Low	Unresolved
QSP-36	Running out of disk space	Low	Unresolved
QSP-37	`areEth1DataEqual()` returns `false` when both `a` and `b` are `nil`	Informational	Fixed
QSP-38	Double unsubscribe	Informational	Fixed
QSP-39	Undefined behaviour of `BestFinalized()` when no peers are connected or some peers have no finalized epochs	Informational	Fixed
QSP-40	Eclipse attacks are still possible	Informational	Acknowledged
QSP-41	Potentially meaningless check	Informational	Fixed
QSP-42	Connection manager does not work properly	Informational	Fixed
QSP-43	One-time calibration of `Roughtime`	Informational	Fixed
QSP-44	Possibly unintentional fallback to TCP encryption	Informational	Fixed
QSP-45	NOISE support has no fallback	Informational	Fixed
QSP-46	Signature validation for block attestations is conditional on feature flag	Informational	Mitigated
QSP-47	`DefaultDataDir()` may return empty string	Informational	Fixed
QSP-48	Support for extra pubsub protocols	Informational	Fixed
QSP-49	Deprecated `jsonpb` dependency	Informational	Acknowledged
QSP-50	Undocumented Kafka topics	Informational	Unresolved
QSP-51	Inconsistent spans	Informational	Fixed
QSP-52	Wrong `StartSpan`	Informational	Fixed
QSP-53	Functions do not add summary to the cache	Informational	Acknowledged
QSP-54	Function `processPendingAtts()` finishes prematurely	Undetermined	Fixed
QSP-55	Corrupted clock	Undetermined	Acknowledged
QSP-56	The variable `voteCount` gets incremented when it is non-zero	Undetermined	Fixed

ID	Description	Severity	Status
QSP-57	Inconsistent `rebuildTrie` updates	? Undetermined	Fixed
QSP-58	Using `UnshuffleList()` instead of `ShuffleList()` contradicts the specification	? Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Integer overflow / underflow
- Number rounding errors
- Denial of service / logical oversights
- Access control
- Business logic contradicting the specification
- Code clones, functionality duplication
- Second pre-image attacks on Merkle Trees
- Client synchronization
- Remote code execution
- Data integrity loss
- Outdated data in cache
- Consensus splits
- Injection type attacks
- Invalid incoming messages
- Falsified messages
- Replay attacks

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
3. Best practices review, which is a review of the code to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your code.

Findings

QSP-1 The functions `IntersectionUint64()`, `IntersectionInt64()` and `IntersectionByteSlices()` may return union instead of intersection

Severity: *High Risk*

Status: Fixed

File(s) affected: `shared/sliceutil/slice.go`

Description: The functions `IntersectionUint64()`, `IntersectionInt64()` and `IntersectionByteSlices()` work correctly only for 2 arguments, i.e., they return intersections. In case more arguments are provided, the functions may return their union.
The intersection functions are used indirectly to find blocks or block roots with a filter. They are then used by `SaveFinalizedCheckpoint()` to save finalized checkpoints. Consequently, it might be possible to consider some blocks as part of the checkpoint (although they shouldn't be) which could lead to consensus splits.

Exploit Scenario: The call `IntersectionUint64([]uint64{2, 3},[]uint64{2, 3, 5},[]uint64{5})` returns `{2,3,5}` instead of `{}`.

Recommendation: The issue is fixed in PR#6067. We do not have any other recommendations.

QSP-2 Potential issues due to granularity of timestamps

Severity: *High Risk*

Status: Fixed

Description: According to specification:

```
The block is not from a future slot (with a `MAXIMUM_GOSSIP_CLOCK_DISPARITY` allowance) -- i.e. validate that signed_beacon_block.message.slot <= current_slot
```

There exists an edge case scenario, however, where this condition would be satisfied for a time that is actually in the future.

Exploit Scenario: Consider the following:

```
slotTime := 1000 * (genesisTime + slot*params.BeaconConfig().SecondsPerSlot)
currentTime := 1000 * uint64(roughtime.Now().Unix())
tolerance := uint64(timeTolerance.Milliseconds()) // 500ms
if slotTime > currentTime+tolerance { … }
```

Let's say that `slotTime` is 5000s, and actual time represented by `roughtime.Now()` is 5900 ms. However, since `roughtime.Now().Unix()` returns time in seconds - that is, it returns 5s, and `currentTime` ends up being 5000 (same as `slotTime`). However, an actual current time is 5900s, which is 900ms higher than `slotTime`, but 900ms is bigger than 500ms. Therefore, the time difference is actually higher than `MAXIMUM_GOSSIP_CLOCK_DISPARITY`.

Recommendation: If tolerance is less than a second, it makes more sense to do time computations in milliseconds directly, rather than using seconds and then multiplying by 1000. Use https://golang.org/pkg/time/#Time.UnixNano instead of just `.Unix()`. Same applies to validating other things, e.g., attestations.
Update: the team fixed the Issue#6103.

QSP-3 Insecure gRPC connection by default

Severity: *High Risk*

Status: Acknowledged

Description: When running the beacon-chain the default setting is not using a secure gRPC connection. Users might not see this (or ignore) this warning:

```
WARN node: Removing database
[2020-05-06 16:09:22]  INFO node: Checking DB database-path=/home/sebi/prysm/beaconchaindata
[2020-05-06 16:09:22]  INFO node: Starting beacon node version=Prysm/Git commit: 7bea2373975e8c3714a8e943ff0dba83ca3b0a07. Built at: 2020-05-06T16:08:52+02:00
[2020-05-06 16:09:22]  INFO blockchain: Waiting to reach the validator deposit threshold to start the beacon chain...
[2020-05-06 16:09:22]  INFO rpc: RPC-API listening on port address=0.0.0.0:4000
[2020-05-06 16:09:22]  WARN rpc: You are using an insecure gRPC connection! Provide a certificate and key to connect securely
```

This could lead to Man-in-the-Middle (MITM) attacks.

Recommendation: Generate a key and certificate via the start scripts by default.

QSP-4 `root` may exist as a subsequence

Severity: *High Risk*

Status: Fixed

File(s) affected: `beacon-chain/db/kv/utils.go`

Description: In `deleteValueForIndices()`, it could be the case that `root` exists as a subsequence between two values, instead of matching an entire value.

Recommendation: We recommend changing the condition:

```
if start == -1 {
    continue
}
```

to

```
if (start == -1) || (start % 32 != 0) {
    continue
}
```

Update: The issue was fixed in PR#6034.

QSP-5 Use of pseudo random number generator where true random number generator would be needed

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/rpc/validator/proposer.go`, `beacon-chain/sync/initial-sync/blocks_fetcher.go`, `beacon-chain/sync/initial-sync-old/round_robin.go`, `shared/testutil/block.go`

Description: PRNGs should not be used for security relevant reasons such as:

- setting random roots and block hashes to prevent a majority from being built if the eth1 node is offline, see
  . L222-223 in `beacon-chain/rpc/validator/proposer.go`
- randomly selecting failover peer from the list of available peers, see
  . L417 in `beacon-chain/sync/initial-sync/blocks_fetcher.go`
- shuffling peers to prevent a bad peer from stalling sync with invalid blocks, see
  . L455 in `beacon-chain/sync/initial-sync/blocks_fetcher.go`
  . L72 in `beacon-chain/sync/initial-sync-old/round_robin.go`
- randomly picking the validator generating attester slashing, see
  . L301, 306 in `shared/testutil/block.go`
- randomly selecting the peer from which to check slots after the given one in an attempt to find non-empty future slot, see
  . L479 in `beacon-chain/sync/initial-sync/blocks_fetcher.go`

Recommendation: Use `crypto/rand` package instead of `math/rand`.
Update: the issue is fixed in PR#6401.

QSP-6 Second pre-image attack on Merkle Trees

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `beacon-chain/state/stateutil/arrays.go`

Description: Functions from `beacon-chain/state/stateutil/arrays.go` are used across the codebase. They are vulnerable to second pre-image attack since there is no distinction made between leaf and intermediate nodes, nor the information about the number of elements is hashed with the root. A similar comment applies to the (related) functions `HashTreeRoot()` from `beacon-chain/state/state_trie.go`, `recomputeRoot()` from `beacon-chain/state/setters.go`, and `MerkleRoot()` in `shared/hashutil/merkleRoot.go`. The latter, however, appears to be unused.
The second pre-image attack could allow an attacker to propagate a block that verifies correctly but contains information that differs from another block with the same set of Merkle tree hashes. Although unlikely, it could result in a DoS attack.

Recommendation: Similarly to other parts of the code, we recommend adding the information about the number of elements into the root. We also recommend removing the unused function `MerkleRoot()`.
Update: the issue is partially fixed in PR#7115.

QSP-7 Function `ActivationEligibilityEpoch()` doesn't check `v == nil || v.validator == nil`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/state/getters.go`

Description: The function `ActivationEligibilityEpoch()` is used in a few places in the codebase. Unlike other functions, it doesn't check the condition `v == nil || v.validator == nil`. Consequently, it may fail and crash the client.

Recommendation: Add the check `if v == nil || v.validator == nil { return 0 }` before returning `v.validator.ActivationEligibilityEpoch`.
Update: the issue is fixed in PR#6347.

QSP-8 Cast from `uint64` to `int`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/sync/pending_blocks_queue.go`, `beacon-chain/rpc/validator/proposer.go`, `beacon-chain/core/state/state.go`, `beacon-chain/powchain/service.go`, `beacon-chain/rpc/beacon/validators.go`

Description: In L141 the function `sortedPendingSlots()` casts slots to `int` from `uint64`. This may result in incorrect data for large values of `slotToPendingBlocks`.
Similar issues occur in:

- `beacon-chain/rpc/validator/proposer.go#324`
- `beacon-chain/rpc/validator/proposer.go#296`
- `beacon-chain/core/state/state.go`, on L81, L86
- `beacon-chain/powchain/service.go#175`
- `beacon-chain/rpc/beacon/validators.go`, on L84, L98, L228, L242

Recommendation: Review all unsafe casts from a type T0 to a type T1 such that T1 is a proper subset of T0 (e.g., converting `uint64` to `int`), either asserting that (1) the cast is safe (an exception can never occur during runtime) or (2) change types when the latter property is not guaranteed.
Update: the issue is fixed in PR#6349 and PR#6039.

QSP-9 Possible loss of data integrity

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/forkchoice/protoarray/store.go`, `beacon-chain/state/getters.go`, `beacon-chain/state/state_trie.go`

Description: The function `Nodes()` on L103 does not acquire an `f.store.nodeIndicesLock.RLock()`. Although it is not directly using `nodeIndices`, if `nodeIndices` is locked by another process, it implicitly means that `f.store.nodes` is being updated.
Similarly, the function `ParentRoot()` does not acquire an `RLock()` unlike the other block header-related getters. Is there a reason that the `Eth1*` getters do not require locking?
The function `NumValidators()` does not acquire an `RLock()`, which may lead to an incorrect value if `validators` is being modified.
The finalizer for a `BeaconState`, as defined on L156, does not acquire the `FieldTrie` lock before updating refs on L160. This is in contrast to L138-140. Note: the Go garbage collector runs concurrently to mutator threads.

Recommendation: Unless it is safe not to, we recommend using the locks.
Update: the issue is fixed in PR#6350.

QSP-10 Checks before the lock

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/state/getters.go`, `beacon-chain/state/setters.go`

Description: Certain `idx` bound checks occur before acquiring a lock. It is not clear why certain checks occur before acquiring a lock.
For example, `ValidatorAtIndex()` incorrectly does the check on L408 before the lock, whereas `ValidatorAtIndexReadOnly()` correctly does the check on L445 after the lock.
Another example; consider `UpdateStateRootAtIndex()`, and note the check on L159, ensuring that the provided `idx` will not be out-of-bounds when writing to an index in `StateRoots`,

which is performed before acquiring the lock on L181. The following type of scenario could occur:

1. The check on L159 passes, and the thread waits for the lock on L181.
2. Elsewhere, the `StateRoots` are updated, possibly decreasing the length below `idx`.
3. The thread from step 1 acquires the lock, causing an error on L184.

Recommendation: We recommend performing index checks after acquiring the lock.
Update: the issue is fixed in PR#6350.

QSP-11 DDoS attack vector through creating a mapping between public keys and validators' IPs

Severity: *Medium Risk*

Status: Acknowledged

Description: It is possible to DDoS the system by attacking block proposers and/or committee members. Such an attack would rely on a mapping between validators' public keys and their IP addresses. Furthermore, one can detect the roles of a validator with a certain public key based on the tables of validator indices and the functions such as `precomputeProposerIndices()`. Once an attacker has the mapping of public keys and IPs, the attacker can target them based on their roles. The more complete the mapping is, the higher possibility for an attacker to DDoS the system from proposing new blocks. There's no need, however, for this mapping to be complete in order to harm the system. When an attacker learns the IP of a validator, the attacker could have the validator slashed by DDoSing the validator when it plays a role of a committee member.

Exploit Scenario: The mapping between public key and the IP of a validator could be learned by checking all the attestations sent in by the other peers. If a peer sent in an attestation which is not an aggregated attestation (i.e., could check this using the function `IsAggregated()`), one can assume that the peer is the attester of that attestation. Thus, we know the peer's IP and also their public key.

Recommendation: We recommend designing a mechanism in which attesters send only aggregated attestations. The more attestations are in the aggregated attestations, the harder it is for an attacker to learn the mapping between the public key of a peer and its IP.
Update: it is an issue with the ETH 2.0 specification and needs to be resolved there first.

QSP-12 Message encoding is changeable: that could lead to network partitions

Severity: *Medium Risk*

Status: Fixed

Description: According to specification implementations MUST use a single encoding. Changing an encoding will require coordination between participating implementations.
While the current testnet is using a single encoding, clients are currently allowed to switch encodings for a given client from regular SSZ to Snappy, which could lead to network partitions.

Recommendation: The risk could be mitigated by better documentation and defining a strategy for mainnet transition. When transitioning to mainnet that uses Snappy, two options are possible: a fallback support for non-snappy, or a coordination of network participants to switch to Snappy and changing the implementation to use Snappy by default.
Update: the issue is fixed in PR#6415.

QSP-13 Opening BoltDB and backup file varies on the permission

Severity: *Medium Risk*

Status: Fixed

Description: Opening BoltDB and backup file varies on the permission - sometimes it is `0666`, others `0600`.

Recommendation: Be consistent about permissions.
Update: the issue is fixed in PR#6378.

QSP-14 Lack of unit tests

Severity: *Medium Risk*

Status: Acknowledged

Description: Overall, many pieces of the code lack unit tests. The lack of tests always adds risks to a project.

Recommendation: We highly recommend adding unit tests.

QSP-15 No support for IPv6

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/discovery.go`

Description: When booting, the node ignores IPv6 nodes. Also, currently the implementation tries to convert IP address to IPv4, if it fails, then the node exits with a fatal message in `main.go`. Note that not all IPv6 IPs can be converted to IPv4, thus essentially forces the node to use IPv4.
In `discovery.go`, the function `convertToSingleMultiAddr()` expects IPv4 only and would throw when an IPv6 is passed in. The function is used in several places, mostly related when booting up the node. This has the following consequences:

- Boot node has to be in IPv4
- Static nodes has to be in IPv4
- Any peers returned by the boot node has to be in IPv4

According to the specification *Clients MAY choose to listen only on IPv6, but MUST be capable of dialing both IPv4 and IPv6 addresses.*

Recommendation: Allow the node to use IPv6.

Update: the issue is fixed in PR#6363.

QSP-16 Presence of the test-only code

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/service.go`

Description: In L62, the handling of the "protocol not supported" error can currently lead to maintaining connection with a peer that does not support the given protocol, and it may not necessarily be a relayer or a DHT node.

Recommendation: We recommend removing the logic and handling the unsupported protocol exception properly.
Update: the issue is fixed in PR#6425.

QSP-17 Possible parent block with same slot

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/core/state/transition.go`

Description: A design flaw in the ETH 2.0 specification allows for producing a block with a parent block of the same slot.

Recommendation: The issue has been fixed in PR#5842 and PR#5846. We have no other recommendation.

QSP-18 No header length validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/encoder/ssz.go`

Description: According to the specification *Before reading the payload, the header MUST be validated: The unsigned protobuf varint used for the length-prefix MUST not be longer than 10 bytes, which is sufficient for any uint64.* Despite the 1MB limit in the code (L18, L21), the 10B header does not appear to be validated anywhere in the code.

Recommendation: We recommend adding header length validation.
Update: the issue is fixed in PR#6577.

QSP-19 Memory resources are never freed in `newBlocksFetcher()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/sync/initial-sync/block_fetcher.go`

Description: Line 86 uses a hardcoded `false` argument on `newBlocksFetcher()` so that empty buckets are never deleted. As per the comment in `leakybucket-go`, this will not be freeing memory resources.

Recommendation: We recommend verifying that this is the intended behavior, and, perhaps documenting rationale for it.
Update: the issue is fixed in PR#6339.

QSP-20 No meaningful code in `removeDisconnectedPeerStatus()`

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `beacon-chain/sync/rpc_status`

Description: Line 130 is missing any meaningful code in `removeDisconnectedPeerStatus()`. It just returns `nil`.

Recommendation: If this is a bug, include a todo and fix. If this is intended, as a best practice, comment on this.
Update: the issue is mitigated in PR#6370.

QSP-21 Disk space exhaustion attack

Severity: *Low Risk*

Status: Acknowledged

Description: When the chain goes through a long period of finality, it keeps many recent beacon states on disk. Beacon states are expensive in size and 100 epochs worth can easily inflate the DB to large sizes. This is a potential attack vector where the attacker may be able to stall the chain long enough that Prysm nodes run out of storage. Under normal conditions, Prysm cleans up recent states when finality is reached, keeping only checkpoints around.
It is unclear how one could purposely execute such an attack without controlling a large portion of the network, but it can be feasible during low participation periods. As the attack would progress and nodes would be unable to operate, the control effect would compound, possibly leading to a temporary network takeover.
This issue has been reported as Issue#6215

Recommendation: We recommend devising a mechanism for freeing disk resources when the space is running low.
Update: the issue is acknowledged in PR#6215.

QSP-22 Disconnected, stale, or bad peer status records are not cleaned up

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/peers/status.go`, `beacon-chain/p2p/service.go`

Description: We did not find any mechanism for cleaning up the peer status information. This could lead to the map `Status.status` growing over time and containing too many items defining peers that no longer exist or that are bad/malicious. This is also a potential DoS attack vector. As it is easy to create a new libp2p peer id, an attacker can keep creating new peer id and connect to the same node over and over again, causing the node to grow the records rapidly and cause memory issues.

Recommendation: Cleanup the peer info periodically. Potentially, introduce a periodic task similar to `Decay()` and run it using `runEvery()` in `service.go`.
Update: the issue is fixed in PR#6614.

QSP-23 Connection manager options are not always initialized correctly

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon/chain/p2p/options.go`

Description: Connection manager is always initialized as follows: `libp2p.ConnectionManager(connmgr.NewConnManager(int(cfg.MaxPeers+2), int(cfg.MaxPeers+2), 1*time.Second))`. The comment says the `2` extra peers are for the bootnode and the relay, however, the number of bootnodes could (and should be) be above one, while having a relay is optional, making the constant of `2` inaccurate.

Recommendation: Adjust the number of peers based on the number of bootnodes and the relay flag.
Update: connection manager has been replaced by connection gater in PR#6184 and PR#6243.

QSP-24 Boot nodes availability and centralization risks

Severity: *Low Risk*

Status: Acknowledged

Description: By default, the testnet seems to be equipped with two boot nodes, but for different protocols: Discovery V5 and Kademlia DHT:

```
INFO p2p: Bootnodes dv5Nodes=[enr:-Ku4QAGwOT9StqmwI5LHaIymIO4ooFKfNkEjWa0f1P8OsElgBh2Ijb-GrD_-b9W4kcPFcwmHQEy5RncqXNqdpVo1heoBh2F0dG5ldHOIAAAAAAAAAAACEZXRoMpAAAAAAAAAAAP__________gmlkgnY0gmlwhBL
f22SJc2VjcDI1NmsxoQJxCnE6v_x2ekgY_uoE1rtwzvGy40mq9eD66XfHPBWgIIN1ZHCCD6A] kadDHTNodes=[/dns4/prylabs.net/tcp/30001/p2p/16Uiu2HAm7Qwe19vz9WzD2Mxn7fXd1vgHHp4iccuyq7TxwRXoAGfc]
```

Since Kademlia DHT is not a part of the spec, for peers that only support discovery-V5, there is only one bootnode available.

Exploit Scenario: The single discovery V5 boot node could be DoS'ed or just lose network connectivity, which could lead to new peers being unable to join the network.

Recommendation: To improve connection availability, we recommend to have at least two Discovery V5 boot nodes located in different "availability zones" (or data centers). Also, we recommend documenting the bootnodes on the parameter page: https://docs.prylabs.network/docs/prysm-usage/parameters/.

QSP-25 Relay option is enabled in libp2p even when unused in Prysm

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon/chain/p2p/options.go`

Description: In L26: `libp2p.EnableRelay()` is called regardless of whether `RelayNodeAddr` is provided or not. While we do not see a risk, it is difficult to guarantee that libp2p's behavior is equivalent (or will always be equivalent in the future) in cases when `libp2p.EnableRelay()` is called (but relayer address not provided) and when `libp2p.EnableRelay()` is not called.

Recommendation: Enabling this option only when `RelayNodeAddr` is provided.
Update: the issue is fixed in PR#6386.

QSP-26 Potential discrepancy between `NextForkVersion` and `ForkVersionSchedule`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `shared/p2putils/fork.go`

Description: In L36, `currentForkVersion` is determined by iterating over items in `ForkVersionSchedule`. However, in `beacon-chain/p2p/fork.go`, L92 (`addForkEntry()`), for the ENR, `NextForkVersion` is taken from the `NextForkVersion` constant. While it could work for many cases (including the scenario when no forks are scheduled), there could be a scenario when `NextForkVersion` does not match any of the items in `ForkVersionSchedule`.

Recommendation: If the given behavior is unexpected, consider adding an assertion or implicitly including `NextForkVersion` in `ForkVersionSchedule`.
Update: the issue is fixed in PR#5997.

QSP-27 Subnet-based whitelisting not working

Severity: *Low Risk*

Status: Fixed

Description: The command line parameter `--p2p-whitelist` does not seem to work properly. When provided "192.168.0.0/16" as the value, the node still managed to connect to seven peers that are outside the current network.

Recommendation: Check the whitelisting logic, ensure this is correct, and fix as necessary.

Update: the issue is fixed in PR#6251.

QSP-28 Rate limiting not implemented for some node communication

Severity: *Low Risk*

Status: Fixed

Description: It is possible to flood a node with valid but useless requests. Specifically, when the node receives a `GetStatus` or `GetMetadata` request, it always responds without rate limiting. This is a potential DoS vector that one node can flood another node with these messages.

Recommendation: Implement rate limiting for node communication to eliminate the possibility of a DoS attack.
Update: the issue is fixed in PR#6606.

QSP-29 Peer descoring and disconnect on failed validation, while being optional in the spec, should be considered

Severity: *Low Risk*

Status: Acknowledged

Description: According to specification *when processing incoming gossip, clients may descore or disconnect peers who fail to observe these constraints.*
The implementation conforms to the spec, i.e., no descoring or disconnect happens at the commit being audited. This opens attack vectors, however, related to the ability of a malicious peer to keep poisoning the network with unsupported topics or invalid messages.

Recommendation: Consider counting invalid topics and messages received from each peer. Once a critical number has reached, the peer needs to be banned for a certain amount of time.
Update: The team responded that this is mostly contained in libp2p's gossipsub library. Descoring of peers is mostly handled in their library but with the current update to v1.1 , gossipsub's topic validators take into account validation results when scoring peers. `ValidationAccept` , `ValidationReject`, `ValidationIgnore` are validation results for pubsub messages which are then used to score peers. The issue is acknowledged in PR#6622.

QSP-30 Incorrect deadline for responses

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/sender.go`, `beacon-chain/sync/rpc_chunked_response.go`

Description: The P2P specification says that: “The requester MUST wait a maximum of `TTFB_TIMEOUT` for the first response byte to arrive (time to first byte—or TTFB—timeout). On that happening, the requester allows a further `RESP_TIMEOUT` for each subsequent response_chunk received.” According to specification `TTFB_TIMEOUT` is 5s. It is the maximum time to wait for the first byte of request response (time-to-first-byte).
However, the deadline is set to:

```
// TTFB_TIME (5s) + RESP_TIMEOUT (10s).
var deadline = params.BeaconNetworkConfig().TtfbTimeout + params.BeaconNetworkConfig().RespTimeout
```

on `L24` in `beacon-chain/p2p/sender.go`. This deadline is used by the following topics:

- `beacon-chain/sync/rpc_beacon_blocks_by_root.go#21`
- `beacon-chain/sync/rpc_goodbye.go#50`
- `beacon-chain/sync/rpc_metadata.go#36`
- `beacon-chain/sync/rpc_ping.go#57`
- `beacon-chain/sync/rpc_chunked_response.go#44`
- `beacon-chain/sync/rpc_status.go#92`.

Recommendation: We recommend verifying the behavior and fixing it if it is incorrect.
Update: the issue is fixed in PR#6583.

QSP-31 Maximum response chunk size not checked for all topics

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/p2p/encoder/ssz.go`

Description: The P2P specification says that *the encoded-payload of a* `response_chunk` *has a maximum uncompressed byte size of* `MAX_CHUNK_SIZE`.
However, this is only checked by the `EncodeWithMaxLength()` function in `beacon-chain/p2p/encoder/ssz.go`, which is used by 2 out of 6 topics, namely:

- `beacon-chain/sync/rpc_beacon_blocks_by_range.go#L143`
- `beacon-chain/sync/rpc_beacon_blocks_by_root.go`, see L116, 26.

The size of the response chunk is not checked explicitly for the other topics.

Recommendation: Check that the size of the response chunk is less than `MAX_CHUNK_SIZE` for all other topics.
Update: the issue is fixed in PR#6424.

QSP-32 Number of bad responses is not incremented in the `Goodbye` topic

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `beacon-chain/sync/rpc_goodbye.go`

Description: According to the specification *in case of an invalid input, a reader MUST:*

- *From responses: ignore the response, the response MUST be considered bad server behavior.*

The number of bad responses is incremented for all topics with the exception of the `Goodbye` topic, whose handlers are in `beacon-chain/sync/rpc_goodbye.go`

Recommendation: Increment number of bad responses in `beacon-chain/sync/rpc_goodbye.go`.

QSP-33 `finalized_root` not checked in `Status` topic

Severity: *Low Risk*

Status: Fixed

File(s) affected: `beacon-chain/sync/rpc_status.go`

Description: According to the specification *if the (`finalized_root`, `finalized_epoch`) shared by the peer is not in the client's chain at the expected epoch*. For example, if Peer 1 sends (root, epoch) of (A, 5) and Peer 2 sends (B, 3) but Peer 1 has root C at epoch 3, then Peer 1 would disconnect because it knows that their chains are irreparably disjoint.
In the following code of the Status topic handler, only the `finalized_epoch` is checked:

```
func (r *Service) validateStatusMessage(msg *pb.Status, stream network.Stream) error {
    forkDigest, err := r.forkDigest()
    if err != nil {
        return err
    }
    if !bytes.Equal(forkDigest[:], msg.ForkDigest) {
        return errWrongForkDigestVersion
    }
    genesis := r.chain.GenesisTime()
    maxEpoch := slotutil.EpochsSinceGenesis(genesis)
    // It would take a minimum of 2 epochs to finalize a
    // previous epoch
    maxFinalizedEpoch := maxEpoch - 2
    if msg.FinalizedEpoch > maxFinalizedEpoch {
        return errInvalidEpoch
    }
    return nil
}
```

Recommendation: Check the `finalized_root` in `beacon-chain/sync/rpc_status.go`.
Update: the issue is fixed in PR#5811 and PR7364.

QSP-34 Weak passwords allowed for validator accounts

Severity: *Low Risk*

Status: Fixed

Description: The reviewed code allows weak passwords for validator accounts. Validators might have their passwords cracked. After running the script `./prysm.sh validator accounts create`, the following is displayed:

```
[2020-05-06 18:08:54]  WARN flags: Using default mainnet config
[2020-05-06 18:08:54]  WARN flags: Disabled validator proposal slashing protection.
[2020-05-06 18:08:54]  WARN flags: Disabled validator attestation slashing protection.
[2020-05-06 18:08:54]  INFO accounts: Create a new validator account for eth2
[2020-05-06 18:08:54]  INFO accounts: Enter a password:
```

Recommendation: This should be prohibited by enforcing a password policy on the client side.
Update: the issue is fixed in commit 9916476.

QSP-35 Minimal system requirements

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `beacon-chain/blockchain/service.go`

Description: Upon start, beacon-node does not check if minimal system requirements are met. Validators may not be aware of system requirements; if many validators face the same issue, the whole network may not perform as expected.

Recommendation: We recommend issuing a warning message if minimal requirements are not met.

QSP-36 Running out of disk space

Severity: *Low Risk*

Status: Unresolved

Description: When the beacon node removes data in the database (e.g., when pruning garbage state), the corresponding space in disk is not freed, as BoltDB does NOT perform any clean-up.

Recommendation: As a means to prevent large databases as much as possible, document how users can properly claim space for any freed data; otherwise, their node can eventually run out of disk space.

QSP-37 `areEth1DataEqual()` returns `false` when both `a` and `b` are `nil`

Severity: *Informational*

Status: Fixed

File(s) affected: `beacon-chain/core/blocks/block_operations.go`

Description: If both `a` and `b` are `nil`, `areEth1DataEqual()` returns `false` despite the arguments being equal.

Recommendation: We checked the behavior of the function with the team. The function is supposed to return `false` when both `a` and `b` are `nil`. We recommend, however, renaming the function to avoid any confusion.
Update: the issue is fixed in PR#6372.

QSP-38 Double unsubscribe

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/sync/initial-sync/service.go`

Description: Line 90 defers `unsubscribe()`, and then line 112 unsubscribes again. This looks like a double call.

Recommendation: We recommend providing a rationale for why both statements are required.
Update: the issue is fixed in PR#6368, PR#6376 and PR#7285.

QSP-39 Undefined behaviour of `BestFinalized()` when no peers are connected or some peers have no finalized epochs

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/p2p/peers/status.go`

Description: In line 430, in the `BestFinalized()` method, if `connected` or `finalized` ends up being an empty list, `targetRoot` defined on L477 remains `nil`, which potentially, leads to issues trying to access `targetEpoch := rootToEpoch[targetRoot]`.

Recommendation: Add a check for the number of connected nodes within the method itself, and return what feels appropriate for such a scenario.
Update: the issue is fixed in PR#6402.

QSP-40 Eclipse attacks are still possible

Severity: Informational

Status: Acknowledged

Description: The current implementation of DiscoveryV5 is still susceptible to the Eclipse attack, as there was a previous audit that indicated it is trivial to perform the attack on DiscoveryV5 in October, 2019. From the issues linked in the report, it seems that there is no resolution to this yet. See the references:

- Issue#122
- Issue#109

This is an issue in the current DiscoveryV5 and NOT the implementation of Prysmatic team. However, we still believe that it is important that the Prysmatic team is aware of the issue so that the system can be patched when a fix is ready in the DiscoveryV5.

Recommendation: Monitor the state of DiscoveryV5 and upgrade when the issue is fixed.

QSP-41 Potentially meaningless check

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/p2p/addr_factory.go`

Description: We found that addresses are checked against "/p2p-circuit". As those follow the `multiAddr` format, we believe that they would never match "/p2p-circuit" exactly. Perhaps the original intention is to check whether the string "/p2p-circuit" is part of the provided address.

Recommendation: Verify the intention and fix accordingly. If the original code works as intended, please consider adding comments.
Update: the issue is fixed in PR#6388.

QSP-42 Connection manager does not work properly

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/p2p/connmgr/connmgr.go`, `beacon-chain/p2p/handshake.go`

Description: The connection manager `TrimOpenConns` doesn't trim any node when the peer count exceeds `highWater`. Currently, the number of connections is being controlled during handshake. This can be seen in the `AddConnectionHandler` of `handshake.go`.

Recommendation: Fix the connection manager so that it trims the peers properly and remove the control in `handshake.go`.
Update: connection manager has been replaced by connection gater in PR#6184 and PR#6243.

QSP-43 One-time calibration of `Roughtime`

Severity: Informational

Status: Fixed

File(s) affected: `shared/roughtime/roughtime.go`

Description: In Prysm's wrapper of `Roughtime`, the time difference (offset) between the beacon node's time and `roughtime` server's times is only calibrated once during init (`shared/roughtime/roughtime.go#27`). For long running beacon nodes, a lack of recalibration might lead to more distorted offsets.

Recommendation: `Roughtime` offset should be recalibrated at a set time interval.
Update: the issue is fixed in PR#6393.

QSP-44 Possibly unintentional fallback to TCP encryption

Severity: *Informational*

Status: Fixed

Description: According to the specification *SecIO with secp256k1 identities will be used for initial interoperability testing*.
While in the commit we're auditing default security is `Sec.io` for testnet, in newer versions of libp2p `Sec.io` comes with a default TLS option as a fallback:

```
var DefaultSecurity = ChainOptions(
    Security(secio.ID, secio.New),
    Security(tls.ID, tls.New),
)
```

TLS is not in the spec, and therefore, should not be implicitly or explicitly listed as a fallback option.

Recommendation: We recommend specifying SecIO (without the TLS fallback) explicitly upon initialization of libp2p options. Explicitly specify the desired parameters rather than relying on non-changing defaults.
Update: the issue is fixed in PR#6440.

QSP-45 NOISE support has no fallback

Severity: *Informational*

Status: Fixed

File(s) affected: `prysm/beacon-chain/p2p/options.go`

Description: According to specification the Libp2p-noise secure channel handshake with secp256k1 identities will be used for mainnet.
In the implementation, the NOISE protocol is enabled using the `EnableNoise` flag in `prysm/beacon-chain/p2p/options.go#35`. While it is a valid approach for mainnet transition, it may be difficult to do the transition without a fallback support for SecIO. The Prysm team, however, is already aware of this, since there is an existing GiHub Issue#5410.

Recommendation: Similar to multiple encoding support, we recommend defining a strategy for mainnet transition. If fallback to SecIO makes it easy, consider using that. Otherwise, transition would require coordination of all the nodes to enable the flag simultaneously to avoid partitions.
Update: the issue is fixed in PR#6440.

QSP-46 Signature validation for block attestations is conditional on feature flag

Severity: *Informational*

Status: Mitigated

Description: The specification states that the signature of attestation is valid as a requirement. While signature validation is implemented, it can be disabled by setting `DisableStrictAttestationPubsubVerification` to `true`.

Recommendation: If possible, remove this flag for production. Clearly document the implications of setting this flag.
Update: the feature has not been removed, but all block and attestations signatures are verified on finalized blocks by default in PR#6499.

QSP-47 `DefaultDataDir()` may return empty string

Severity: *Informational*

Status: Fixed

File(s) affected: `shared/cmd/defaults.go`

Description: The function `DefaultDataDir()` may return an empty string, but it is not handled as mentioned in the comments.

Recommendation: Handle the returned empty string as promised in the comments, or throw an error.
Update: the issue is fixed in PR#6394.

QSP-48 Support for extra pubsub protocols

Severity: *Informational*

Status: Fixed

Description: According to specification clients MUST support the gossipsub libp2p protocol.
Gossipsub is supported and is the default protocol, but "flood" and "random" are supported as well. Peers using a different protocol may find themselves incompatible with the rest of the network.

Recommendation: Consider phasing out protocols that do not need to be supported.
Update: the issue is fixed in PR#6419.

QSP-49 Deprecated `jsonpb` dependency

Severity: *Informational*

Status: Acknowledged

File(s) affected: `beacon-chain/db/kafka/export_wrapper.go`

Description: `jsonpb` is a dependency; however, `jsonpb` is currently deprecated.

Recommendation: Switch to `protojson` to have more up to date fixes and/or features.
Update: the issue is acknowledged in PR#6383.

QSP-50 Undocumented Kafka topics

Severity: Informational

Status: Unresolved

File(s) affected: `beacon-chain/db/kafka/export_wrapper.go`

Description: List of Kafka topics is not documented. Thus, if a node operator wants to monitor their nodes (e.g., for abnormality detection), they would have to resort to the `go` code to find out which topics to create and later consume from.

Recommendation: Provide documentation on what topics to setup if one is to stream to a Kafka server.

QSP-51 Inconsistent spans

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/db/kv/archive.go`

Description: Unlike other functions, `marshalBalances()` and `unmarshalBalances()` do not start & end a span.

Recommendation: We recommend following other operations to start & end a span.
Update: the issue is fixed in PR#6352.

QSP-52 Wrong `StartSpan`

Severity: Informational

Status: Fixed

File(s) affected: `beacon-chain/db/kv/archived_point.go`

Description: On L25, `StartSpan` refers to `BeaconDB.SaveHeadBlockRoot`. It should be `BeaconDB.SaveLastArchivedIndex`.
On L75, `StartSpan` refers to `BeaconDB.ArchivePointRoot`. It should be `BeaconDB.ArchivedPointRoot`.

Recommendation: Change string argument to `BeaconDB.SaveLastArchivedIndex` and `BeaconDB.ArchivedPointRoot`, respectively.
Update: the issue is fixed in PR#6352.

QSP-53 Functions do not add summary to the cache

Severity: Informational

Status: Acknowledged

File(s) affected: `beacon-chain/db/kv/state_summary.go`

Description: The functions `SaveStateSummary()` and `SaveStateSummaries()` do not add the summary to the cache, which is unexpected.

Recommendation: Put the summary in the cache or at least provide further documentation in code justifying why that is not the case.
Update: the issue is acknowledged in PR#6384.

QSP-54 Function `processPendingAtts()` finishes prematurely

Severity: Undetermined

Status: Fixed

File(s) affected: `beacon-chain/sync/pending_attestations_queue.go`

Description: In line 124, in a loop, the function terminates (without an error). The current code will represent a success for line 32. Is this intended? Would it make sense to use continue/break instead? Should it return an error?

Recommendation: We recommended documenting the desired behavior and adding a log.
Update: the issue is fixed in PR#6371.

QSP-55 Corrupted clock

Severity: Undetermined

Status: Acknowledged

File(s) affected: `beacon-chain/sync/deadlines.go`

Description: This file uses system time, because `libp2p` uses it too. If the system clock becomes corrupted (the time never passes), what will the consequences be?

Recommendation: We recommend documenting the consequences.
Update: the issue is acknowledged in PR#6404.

QSP-56 The variable `voteCount` gets incremented when it is non-zero

Severity: *Undetermined*

Status: Fixed

File(s) affected: `beacon-chain/core/blocks/block_operations.go`

Description: It is unclear why the variable `voteCount` gets incremented when it is non-zero.

Recommendation: The team removed vote cache and the operation that increments `voteCount` in PR#5792. We have no other recommendation.

QSP-57 Inconsistent `rebuildTrie` updates

Severity: *Undetermined*

Status: Fixed

File(s) affected: `beacon-chain/state/setters.go`

Description: In `state_trie.rootSelector()`, there are 7 cases that rebuild the trie if the corresponding boolean is set: `blockRoots`, `stateRoots`, `eth1DataVotes`, `validators`, `randaoMixes`, `previousEpochAttestations`, `currentEpochAttestations`. For each of these fields, the corresponding setter functions have a statement of the form`b.rebuildTrie[field] = true`. However, this is missing for `SetBlockRoots()`.

Recommendation: We recommend double checking whether `rebuildTrie` should be updated or not.
Update: the issue is fixed in PR#6390.

QSP-58 Using `UnshuffleList()` instead of `ShuffleList()` contradicts the specification

Severity: *Undetermined*

Status: Fixed

File(s) affected: `beacon-chain/core/helpers/committee.go`

Description: The code uses `UnshuffleList()` instead of `ShuffleList()` on:

1. L140, which contradicts the spec in L126.
2. L294, which contradicts the description on L277.

According to the spec a committee is a shuffled set of validators (see the spec in the function's code comment). Note that when the function `BeaconCommitteeFromState()` calls `BeaconCommittee()` which calls `ComputeCommittee()`, the `indices` passed to `ComputeCommittee()` are a sorted slice of indices.
Although `UnshuffleList()` may shuffle a list that wasn't previously shuffled, it is unclear why it is used instead of `ShuffleList()`.

Recommendation: We recommend providing rationale for why `UnshuffleList()` is used instead of `ShuffleList()`, or replacing it with the latter.
Update: the issue is fixed in PR#6381.

Adherence to Specification

1. In `beacon-chain/blockchain/process_attestation.go`, the function `onAttestation()`, many parts of the specification for `on_attestatation()` delay consideration, rather than avoiding consideration. However, the function `onAttestation()` does not seem to ever delay - only returns (possibly with errors). Ensure that the intent of the specification is preserved. **Update:** fixed.

2. In `beacon-chain/blockchain/process_attestation_helpers.go#157`, it may not be entirely clear why the seed impact on attestation verification is omitted from the specification. **Update:** fixed.

3. Typo in `beacon-chain/blockchain/process_block_helpers.go#45`, "feature" -> "future". **Update:** fixed.

4. Typo in `beacon-chain/blockchain/process_block_helpers.go#444`, "couldn't received" -> "couldn't be received". **Update:** fixed.

5. In `beacon-chain/core/validators/validator.go`, although the implementation of `SlashValidator()` conforms to the specification, it is unclear whether `BeaconState.slashings` should represent per-epoch sums of slashed effective balances (i.e., balances before slashing) or only deltas (i.e., the actual slashed balances). **Update:** fixed.

6. In `beacon-chain/core/helpers/rewards_penalties.go#30`, we check if `total == 0`, which relates to the `max()` function in the spec pseudocode. Although it should be semantically equivalent in practice, it may be best to strictly match the spec by changing the line to `total < params.BeaconConfig().EffectiveBalanceIncrement`. **Update:** fixed.

7. In `beacon-chain/core/helpers/committee.go`, it is unclear how the domain of certain calls to `Seed()` are determined. For example, in `UpdateProposerIndicesInCache()` on L344 `DomainBeaconAttester` is used, whereas in `precomputeProposerIndices()` on L371 `DomainBeaconProposer` is used. **Update:** fixed.

8. In `beacon-chain/p2p/encoder/ssz.go`, according to specification *Encoding-dependent header: Req/Resp protocols using the ssz or ssz_snappy encoding strategies MUST encode the length of the raw SSZ bytes, encoded as an unsigned protobuf varint*. On L34, the encoder contains a function `Encode()` that can violate the spec (as opposed to `EncodeWithLength()`). The function seems to be used in tests only. We suggest moving it to testing scope only and removing it from the encoder. **Update:** fixed.

9. Specification pseudocode does not always exactly match the ETH2 specification, including, but not limited to the following examples:
    1. In `beacon-chain/core/helpers/validators.go` in the pseudocode on L116, `MIN_SEED_LOOKAHEAD` is used instead of `MAX_SEED_LOOKAHEAD`. **Update:** fixed.
    2. In `beacon-chain/core/blocks/block_operations.go` in the pseudocode for `process_block_header`, the function call `signing_root(state.latest_block_header)` is used instead of `hash_tree_root(state.latest_block_header)`. **Update:** fixed.
    3. In `beacon-chain/core/helpers/committee.go`, in the pseudocode for `get_beacon_committee`, we have `index=epoch_offset,` instead of `index=(slot % SLOTS_PER_EPOCH) * committees_per_slot + index`. **Update:** fixed.
    4. In `beacon-chain/core/state/transition.go` in the pseudocode for `state_transition`, there are several missing snippets such as `if validate_result: assert verify_block_signature(state, signed_block)`. **Update:** fixed.
    5. In `beacon-chain/core/validators/validator.go`, after L113, the `if-block` is missing. There should be a line for `whistleblower_index = proposer_index`. **Update:** fixed.
    6. The type `Hash` is often used instead of `Bytes32`.
    7. In `beacon-chain/core/block_operations.go#626`, it is unclear where the assertion `assert data.index < get_committee_count_at_slot(state, data.slot)` is enforced.
    8. In `beacon-chain/core/block_operations.go#1038`, it seems the `depth` parameter noted in the spec pseudocode is not used. On the other hand, the `depth` variable is write-only in the specification. We recommend aligning the specification and code with each other, perhaps, mixing in `depth` into the root.
10. In `beacon-chain/core/blocks/block_operations.go`, the comments before `ProcessRandao()`, `VerifyIndexedAttestation()`, `ProcessDeposits()` do not match the specification. **Update:** fixed.
11. In `beacon-chain/core/helpers/attestation.go`, the comment before `SlotSignature()` does not match the specification for `get_slot_signature()`. **Update:** fixed.
12. In `beacon-chain/core/helpers/committee.go`, the comment before `BeaconCommitteeFromState()` does not match the specification. **Update:** fixed.
13. Typo in `beacon-chain/core/helpers/validators.go#116`, “MIN” -> “MAX”. **Update:** fixed.

Code Documentation

Generally, the code is lacking inline developer documentation. For instance, `beacon-chain/p2p/subnets.go`, `beacon-chain/p2p/utils.go` do not contain any documentation, while for other files (such as, `beacon-chain/p2p/service.go`) contain comments, however, do not describe the individual parameters and return values. Furthermore:

1. In `validator/node/node.go`, on L276 and L282, the error message "Could not create DB in dir %s" should say "clear" instead of "create". **Update:** fixed.
2. In `validator/flags/interop.go`, the description for `InteropNumValidators` appears to incorrectly reference itself: "The number of validators to deterministically generate when used in combination with --interop-num-validators.". Likely a copy and paste error between L11 and 17. **Update:** fixed.
3. In `beacon-chain/state/stateutil/state_root.go#196`, copy and paste error -- "could not compute previous epoch attestations merkleization" should be "... current epoch ...". **Update:** fixed.
4. In `beacon-chain/state/types.go#112`, the comment in `MinusRef()` should say "underflow", not "overflow". **Update:** fixed.
5. In `beacon-chain/state/state_trie.go`, in the function `merkleize()`, it is claimed that it *pads the leaves to a power-of-two length*. It does this only up to the length of 32. We recommend adjusting the comment. **Update:** fixed.
6. In `beacon-chain/state/setters.go`, the comment *This PR updates the randao mixes* is used in many places, but is likely only relevant to `UpdateRandaoMixesAtIndex()`. **Update:** fixed.
7. In `beacon-chain/state/setters.go`, unclear why "This PR..." is referenced in many of the comments. **Update:** fixed.
8. In `beacon-chain/state/setters.go`, L135-136 - one or more words are missing in the comments. **Update:** fixed.
9. In `beacon-chain/core/epoch/precompute/justification_finalization.go`, the specification pseudocode from `process_justification_and_finalization` could be included for `ProcessJustificationAndFinalizationPreCompute()`. **Update:** fixed.
10. In `beacon-chain/core/block_operations.go`, the comment on L587-588 *This function returns a list of pending attestations which can then be appended to the BeaconState's latest attestations.* is not accurate. The attestations are appended to the `BeaconState`'s list internal to the function, and the `BeaconState` itself is returned. **Update:** fixed.
11. Cache in `beacon-chain/cache/hot_state_cache.go` is the only cache that doesn't use a mutex. We recommend documenting this design rationale. **Update:** fixed.
12. Typos in `beacon-chain/p2p/service.go`, L383-384, “id’s” -> “ids” **Update:** fixed.
13. Typo in `beacon-chain/blockchain/process_attestation.go#125`, "state to to validate". **Update:** fixed.
14. Typo in `beacon-chain/blockchain/receive_block.go#63`, "preformed" -> "performed". **Update:** fixed.
15. Typo in `beacon-chain/core/helpers/slot_epoch.go#57`, "form" should be "from". **Update:** fixed.
16. In `beacon-chain/blockchain/metrics.go`, should the `Help` strings on L80 and L84 be reversed? **Update:** fixed.
17. In `beacon-chain/blockchain/chain_info.go#146`, "head state" should be "head block". **Update:** fixed.
18. In `beacon-chain/blockchain/head.go#30`, not a correct English sentence. **Update:** fixed.
19. In `beacon-chain/blockchain/head.go#115`, typo "inital-sync-cache-state" should be "initial-sync-cache-state". **Update:** fixed.
20. Typo in `beacon-chain/core/state/transition.go#244`: "skip skips" -> "skip slots". **Update:** fixed.
21. In `shared/bytesutil/bytes.go`, L24-43, the functions implement little-endian format but specifications mention `big`. **Update:** fixed.
22. In `shared/sliceutil/slice.go`, the function `NotUint64()`, returns elements of `b` which are not in `a`, not the other way around. **Update:** fixed.
23. In `shared/sliceutil/slice.go`, the function `UnionByteSlices()`, doesn't return common elements, but all of them. **Update:** fixed.
24. In `beacon-chain/core/epoch/epoch_processing.go`, in the spec pseudocode for `process_final_updates`, after L208 the pseudocode for the `if-block` is missing: `validator.effective_balance = min(balance - balance % EFFECTIVE_BALANCE_INCREMENT, MAX_EFFECTIVE_BALANCE)`. **Update:** fixed.
25. In `beacon-chain/p2p/encoder/ssz.go`, in function `DecodeWithMaxLength()`, `msgLen` stands for the remaining bytes in `r`, but this is NOT the decoded message. Thus the check on L163 and the comment on L164 do not match. The decoded message seems to be on L167 `numOfBytes, err := r.Read(b)`. **Update:** fixed.

26. Typo in `beacon-chain/core/helpers/slot_epoch.go`, “alswo” -> “also”. **Update:** fixed.
27. Typo in `beacon-chain/core/helpers/slot_epoch.go`, “compute_start_slot_of_epoch” -> “compute_start_slot_at_epoch”. **Update:** fixed.
28. There are issues in end-user documentation:
    1. All 3 links are broken in step 1 on this page https://docs.prylabs.network/docs/install/lin/activating-a-validator/ **Update:** fixed.
    2. The link indicated in the following screenshots are broken on this page https://docs.prylabs.network/docs/install/lin/bazel/ **Update:** fixed.
29. In `db/kv*`, document when a function should start & end a trace, and when traces are not required. This shall help other open-source contributors when adding new functions to the system.
30. In `beacon-chain/db/iface/interface.go`, URLs mentioned in file are broken. **Update:** fixed.
31. In `beacon-chain/db/kv/blocks.go`, L585-587, comment is misleading, as no list is being returned. Rather, the return will be a map with a single key-value pair: "block-parent-root-indices” -> parentRoot (array of bytes). Change the comment accordingly. **Update:** fixed.
32. In `beacon-chain/db/kv/blocks.go`, in function `fetchBlockRootsBySlotRange()` header comment says it performs a binary search. However, if `step` is one, the implemented logic won’t skip half of the slots in the range. Fix the comment accordingly. **Update:** fixed.
33. Typo in `beacon-chain/syncservice.go`, “Intialize” -> “Initialize”. **Update:** fixed.

Adherence to Best Practices

1. In `shared/bls/bls.go`, duplicated code between `VerifyAggregate`, `AggregateVerify` and `FastAggregateVerify`. **Update:** acknowledged.
2. In `beacon-chain/state/getters.go`, duplicated code between: **Update:** fixed.
    1. `StateRoots`, `HistoricalRoots`, and `BlockRoots`
    2. `RandaoMixesLength` and `BalancesLength`
    3. `PreviousEpochAttestations` and `CurrentEpochAttestations`
    4. `CurrentJustifiedCheckpoint` and `PreviousJustified`
3. In `beacon-chain/state/setters.go`, duplicated code between: **Update:** fixed.
    1. `SetCurrentJustifiedCheckpoint` and `SetPreviousJustifiedCheckpoint`
    2. `AppendValidator` and `AppendBalance`
4. In `beacon-chain/cache/depositcache/pending_deposits.go`, duplicated code between `pendingDeposits` and `pendingContainers`. **Update:** fixed.
5. In `beacon-chain/interop-cold-start/service.go`, error messages on L182, 189, 192 should read “could not save”. **Update:** fixed.
6. In `beacon-chain/forkchoice/types.go`, field names in `type Node struct` do not follow a consistent naming convention. **Update:** fixed.
7. In `beacon-chain/blockchain/metrics.go#122`, ignores error logging. Should it log it? **Update:** fixed.
8. In `beacon-chain/p2p/encoder/ssz.go`, in function `writeSnappyBuffer()`, when there is an error, the function returns without closing `bufWriter`. We recommend closing it before returning. **Update:** fixed.
9. In `beacon-chain/p2p/beacon-chain/p2p/sender.go#20`, `baseTopic` is used without validation. If the upstream code has an issue or becomes out-of-sync (e.g., attempts to send a topic that is no longer supported), the `baseTopic` string could potentially be a topic that is actually not supported. **Update:** fixed.
10. In `beacon-chain/p2p/peers/status.go#487`, `CurrentEpoch()` could be renamed to `HighestEpoch()` to better reflect the semantics. **Update:** fixed.
11. In `beacon-chain/p2p/peers/status.go#433`, `pidEpochs` could be renamed to `pidToEpoch` for consistency with `rootToEpoch` defined on `L432`. **Update:** fixed.
12. There are multiple instances of in-line constants. We recommend moving these constants elsewhere: a separate constants file or the top of the current file, for better visibility and maintainability. Some instances including, but not limited to: **Update:** fixed.
    1. `beacon-chain/p2p/service.go#277`: `5*time.Second`
    2. `beacon-chain/p2p/service.go#281`: `10*time.Second`
    3. `beacon-chain/p2p/service.go#260`: `30 * time.Second`
    4. `beacon-chain/p2p/peers/watch_peers.go#29`: `30*time.Second`
    5. `beacon-chain/p2p/service.go`, `L100-102`, (the cache parameters `NumCounters`, `MaxCost`, `BufferItems`)
    6. `beacon-chain/sync/rpc_status.go#169`: `50 * time.Millisecond`
    7. `beacon-chain/sync/service.go#158`: `time.Second*10`.
13. Error messages should be more precise **Update:** fixed.
    1. In `beacon-chain/p2p/service.go`
        1. L214, the error message “Could not create peer” should be concretized to specify inability to connect to a peer defined by the relayer address
        2. L226, the error message "Failed to start discovery" should be concretized to specify “discovery V5”
        3. L232, the error message "Could not add bootnode to the exclusion list” should probably state “Could not connect to bootnodes”
        4. L245, the error message “Could not connect to bootnode” should concretize that it is about Kademlia bootstrap
        5. L255, the error message "Could not create peer" should be concretized to specify inability to connect to a peer defined by KademliaBootStrapAddr.
        6. L250, the error should also be logged, like on L245, for instance, “Could not add Kademlia boot node to the exclusion list"
    2. In `beacon-chain/sync/rpc_status.go#152`, the error message "Invalid fork version from peer" is not precise because the error could also be due to an invalid

epoch as seen on `L216`.

14. In `beacon-chain/p2p/watch_peers.go`, there is a 30-second timeout. However, in `beacon-chain/p2p/service.go#555`, and `beacon-chain/p2p/dial_relay_node.go#29`, such a timeout is not present. We recommend to make the logic consistent. **Update:** fixed.

15. We recommend to resolving the TODOs for production readiness: **Update:** fixed.
    1. In `beacon-chain/p2p/watch_peers.go#159`
    2. In `beacon-chain/sync/rpc_beacon_blocks_by_range.go#78`
    3. In `beacon-chain/rpc/service.go#205`
    4. In `beacon-chain/rpc/validator/proposer.go#391`
    5. In `beacon-chain/sync/rpc_block_by_range#78`
    6. And others. `git grep TODO` can be used to obtain a complete list.

16. In `beacon-chain/p2p/discovery.go#33`, the `createListener()` method does not follow the same pattern for error propagation as other methods do: it fails with `Fatal` errors while, for instance, the `createLocalNode()` method returns an error in case of an issue, which seems to be more common in the P2P codebase. We recommend improving consistency of error handling. **Update:** fixed.

17. In `gossip_topic_mappings.go`, the following constant is defined in the `GossipTopicMappings`: `"/eth2/%x/committee_index%d_beacon_attestation": &pb.Attestation{}` . The same constant is being defined again in `broadcaster.go`, `attestationSubnetTopicFormat`. We advise against code clones. **Update:** fixed.

18. The schema version of the libp2p protocol ID is currently hard-coded in the topic strings. To facilitate updating the version it would be beneficial to use a separate `SchemaVersion` state variable. **Update:** fixed.

19. In `validator/client/runner.go#111` since `id` represents a public key, we recommend renaming it to `pubKey`. **Update:** fixed.

20. In `beacon-chain/core/helpers/validators.go#178`, as long as the epoch doesn't change, there is no need to call `CurrentEpoch(state)` since `L151` already retrieves it in the `e`. **Update:** fixed.

21. In `beacon-chain/state/stateutil/arrays.go#118`, inner and outer loops use variables named `i`. Use different names for these variables. **Update:** fixed.

22. In `shared/sliceutil/slice.go`, function `SplitOffset()` should add a check to ensure that `index * chunks < listSize`. **Update:** acknowledged.

23. In `beacon-chain/state/stateutil/blocks.go`, the constants such as 16 on L112, which corresponds to `MAX_DEPOSITS` in the ETH2 spec, should be refactored into named constants. **Update:** fixed.

24. In `beacon-chain/state/state_trie.go`, the constant 21 is used throughout which corresponds to the number of field indices. This should be a named constant. **Update:** fixed.

25. In `beacon-chain/state/setters.go`, the function `SetStateRoots()` does not appear to be used anywhere or tested. **Update:** acknowledged.

26. In `beacon-chain/state/setters.go`, when updating reference counts, `MinusRef()` is used in many places, however `varname.refs--` is used in others. It is not clear what distinguishes these cases, and likely `MinusRef()` could be used throughout. **Update:** acknowledged.

27. In `beacon-chain/state/getters.go`, the function `ValidatorAtIndex()` could reuse `Cloners.CopyValidator()`. **Update:** acknowledged.

28. In `beacon-chain/core/feed/operation/events.go#17`, the constant `AggregatedAttReceived` does not appear to be used anywhere in the codebase. **Update:** acknowledged.

29. In `beacon-chain/core/block_operations.go#818`, the statement: `if len(indices) > 0`, is unnecessary since the nested `for`-loop would not loop if `len(indices) == 0`. **Update:** acknowledged.

30. In `beacon-chain/core/block_operations.go`, the deprecated function `verifyDepositDataSigningRoot()`, which relies upon the function `ssz.SigningRoot()` (which is also deprecated), is still used in `ProcessDeposit()`. **Update:** acknowledged.

31. In `beacon-chain/main.go#115`, the switch statement has breaks in case branches. Those are not needed in Go. **Update:** fixed.

32. In `validator/main.go#152`, the switch statement has breaks in case branches. Those are not needed in Go. **Update:** fixed.

33. In `beacon-chain/sync/subscriber.go#36`, `noopValidator()` does not use the context parameter. It should be removed. **Update:** acknowledged.

34. In `beacon-chain/sync/subscriber.go#135`, uses deprecated `PubSub.Subscribe()`. The code should be updated. **Update:** fixed.

35. In `beacon-chain/sync/initial-sync/block_fetcher.go#405`, the function for selecting a failover peer is inefficient. It does not have to filter out the wrong peer at the beginning by searching for it. It can right away randomly select a new peer. If it happens to find the same peer (the forbidden one), it knows where it is and can drop it. **Update:** fixed.

36. In `beacon-chain/sync/initial-sync/block_fetcher.go#269`, the name `count` is reused (previously used as a name of of the argument). We recommend using different names. **Update:** fixed.

37. In `beacon-chain/sync/initial-sync/blocks_queue.go#310`, there is a switch statement with an empty `default` label. Some logging/comment would be handy. **Update:** fixed.

38. In `beacon-chain/p2p/encoder/ssz.go`, L160 and 163 should be swapped. The length error is known on L160, and the reader is not needed if the error happens. **Update:** fixed.

39. In `beacon-chain/db/iface/interface.go` (and implementations), `DeleteBlock()` and `DeleteBlocks()` are not used outside tests. They are declared in the interface, and supported by the database in `blocks.go` (as well as kafka), but not used otherwise. They should be moved to the test scope only. **Update:** acknowledged.

40. In `beacon-chain/db/kv/operations.go`, the functions `HasVoluntaryExit()` and `VoluntaryExit()` have a lot of duplicated code related to retrieving the exit. Reuse `VoluntaryExit()` in `HasVoluntaryExit()` and return whether it is not `nil`. **Update:** fixed.

41. In `beacon-chain/db/kv/operations.go`, it seems that none of the methods here is called externally. Are they part of the future phase? At the very least, `DeleteVoluntaryExit()` is only called in tests. Consider moving it to the test scope. **Update:** acknowledged.

42. In `beacon-chain/db/kv/slashings.go`, the function `ProposerSlashing()` has a lot of duplicated code with `HasProposerSlashing()`. Reuse the result in `HasProposerSlashing()`. Same holds for `AttesterSlashing()` and `HasAttesterSlashing()`. **Update:** fixed.

43. In `beacon-chain/db/kv/slashings.go`, `Delete[Attester]Slashing()` methods seem to have a testing scope only. Unless they are part of an upcoming phase, consider moving those to the testing scope. **Update:** acknowledged.

44. In `beacon-chain/db/kv/state_summary.go` and `beacon-chain/db/kv/state.go`, the functions `State()` and `HasState()` duplicate the code similarly to `StateSumary()` and `HasStateSummary()`. **Update:** fixed.

45. In package `validator/db`, the code suffers from the same unused code issue as the beacon chain database (the `Delete()` methods). This is present in `DeleteAttestationHistory()` of `attestation_history.go`, `DeleteProposalHistory()` in `proposal_history.go`. **Update:** fixed.

46. In package `validator/db`, there is some code duplicated in `NewKVStore()` methods in both `validator/db` and `beacon_chain/db` packages. Consider removing the duplicates and consolidate the filesystem code. **Update:** acknowledged.

47. Inline constants are used instead of existing named constants. **Update:** fixed.
    1. `TTFB_TIMEOUT` should be used in:
        1. `beacon-chain/p2p/service.go#277`
        2. `beacon-chain/sync/rpc_beacon_blocks_by_root.go#55`
        3. `beacon-chain/sync/rpc_goodbye.go#32`
        4. `beacon-chain/sync/rpc_metadata.go#21`
        5. `beacon-chain/sync/rpc_ping.go#21`
        6. `beacon-chain/sync/rpc_status.go#142`
    2. `RESP_TIMEOUT` should be used in:
        1. `beacon-chain/p2p/service.go#281`
        2. `beacon-chain/sync/rpc_beacon_blocks_by_range.go#25`
        3. `beacon-chain/sync/rpc_beacon_blocks_by_root.go#18`
        4. `beacon-chain/sync/rpc_chunked_response.go#44`
        5. `beacon-chain/sync/rpc_goodbye.go#47`
        6. `beacon-chain/sync/rpc_metadata.go#33`
        7. `beacon-chain/sync/rpc_ping.go#53`
        8. `beacon-chain/sync/rpc_status.go#73`

48. Many DB-related functions that perform a computation on a given entity also have a corresponding function to perform the same computation on a corresponding batch of elements of the same entity kind. Examples: `SaveBlock()`/`SaveBlocks()`, `DeleteBlock()`/`DeleteBlocks()`, `SaveStateSummary()/SaveStateSummaries()`, etc. Such pairs of singular/plural functions share a lot of common code that could be simplified. Since the plural form is the most generic computation, it suffices to rewrite the singular function as a call to the plural one giving it a batch of size one. **Update:** fixed.

49. In `beacon-chain/db/filters/filter.go`, L21-44: use `iota` instead of enumerating sequential values. **Update:** fixed.

50. In `beacon-chain/db/filters/filter.go`, declaring `FilterType` as an alias to `int` seems excessive; rather, `uint8` would have sufficed. **Update:** fixed.

51. In `beacon-chain/db/kafka/export_wrapper.go#36`, prior to returning, issue a log info just to inform that the db was not wrapped. **Update:** fixed.

52. In `beacon-chain/db/kv/archived_point.go#38`, `Update` is being called, but no side effect occurs. Replace it with `View`. **Update:** fixed.

53. In `beacon-chain/db/kv/attestations.go`, function `createAttestationIndicesFromData()` does not start & end a span. Following other functions, start & end a span. **Update:** fixed.

54. In `beacon-chain/db/kv/backup.go#32`, the permissions of the backup directory path (0777) are less strict than the database folder (0700). Make it the same - 0700. **Update:** fixed.

55. In `beacon-chain/db/kv/backup.go#38`, the permissions of the backup db file (0666) are less strict than the target database (0600). Make it the same - 0600. **Update:** fixed.

56. In `beacon-chain/db/kv/backup.go#38`: give `Open` a timeout option. **Update:** fixed.

57. In `beacon-chain/db/kv/backup.go`, receiver name (`kv`) does not match receiver name in other files (`k`). Change all receiver names under `db/*` to the same name. **Update:** fixed.

58. In `beacon-chain/db/kv/backup.go`, `Backup` does not start & end an span. Following other functions in `kv/*`, start & end a span. **Update:** fixed.

59. In `beacon-chain/db/kv/blocks.go`, L527, L531, Rename `k` to `key` for better expressiveness. **Update:** fixed.

60. In `beacon-chain/db/kv/blocks.go`, functions `fetchBlockRootsBySlotRange()`, `createBlockIndicesFromBlock()`, and `createBlockIndicesFromFilters()` do not start & end a span. Following other operations, start & end a span. **Update:** fixed.

61. In `beacon-chain/db/kv/check_historical_state.go`, the function `HistoricalStatesDeleted()` does not start & end a`Span`. Following other operations, start & end a span. **Update:** fixed.

62. In `beacon-chain/db/kv/encoding.go`, `encode` and `decode` functions do not start & end a span. Following other functions in `kv/*`, start & end a span. **Update:** acknowledged.

63. In `beacon-chain/db/kv/kv.go`, functions in `kv.go` do not start & end a span. Following other operations in kv/*, start & end a span. **Update:** acknowledged.

64. In `beacon-chain/db/kv/regen_historical_states.go`, functions `lastSavedBlockArchivedIndex()` and `saveArchivedInfo()` do not start a trace. Following other functions in `kv/*`, start & end a span. **Update:** fixed.

65. In `beacon-chain/db/kv/utils.go`, no function in `utils.go` start & end a span. Following other functions in `kv/*`, start & end a span. **Update:** fixed.

66. In `validator/db/attestation_history.go`, `unmarshalAttestationHistory()` does not start & end a trace. Following other operations, start & end a span. **Update:** fixed.

67. In `validator/db/db.go`, rename receiver name for better reading. Suggestion: instead of `db`, which causes `db.db` to appear in some places, rename it to `store`. **Update:** fixed.

68. In `validator/db/setup_db.go`, as this code is essentially testing infra, put it in a dedicated testing folder (the same way as performed for the `setup_db.go` file in the beacon-chain db). **Update:** acknowledged.

69. In `Beacon-chain::Sync`, across many files, `Service` receiver is named `r`. Suggestion: name `r *Service` as `service *Service`. **Update:** fixed.

70. In `beacon-chain/sync/service.go`, rename receiver to make it more expressive. Suggestion: name `r *Service` as `service *Service`. **Update:** fixed.

71. In `beacon-chain/sync/pending_blocks_queue.go`, lock/unlock in `processPendingBlocksQueue()` is not required, as the internal function execution never overlaps with itself. **Update:** fixed.

72. In `beacon-chain/sync/pending_attestations_queue.go`, lock/unlock in `processPendingAttsQueue()` is not required, as the internal function execution never overlaps with itself. **Update:** fixed.

73. In `beacon-chain/blockchain/service.go#349`, error message is incorrect, as it misses negation. Should be "could not save genesis block root". **Update:** fixed.

Test Results

Test Suite Results

Multiple code pieces lack corresponding unit tests, for example:

1. In `beacon-chain/db/kv/checkpoint_historical_state`, the function `HistoricalStatesDeleted()` is missing a test in the `kv` package.
2. The code in:
    1. `beacon-chain/db/kv/check_historical_state.go`
    2. `beacon-chain/db/kv/powchain.go`
    3. `validator/db/db.go`
    4. `beacon-chain/sync/decode_pubsub.go`
    5. `beacon-chain/db/kv/powchain.go`
    6. `beacon-chain/db/kv/regen_historical_states.go`
    7. `validator/db/db.go`
    8. `beacon-chain/db/kv/state.go`.

Lack of tests always add risks to a project. We highly recommend adding unit tests.

Code Coverage

Test coverage analysis is available on: https://codecov.io/gh/prysmaticlabs/prysm/tree/7bea2373975e8c3714a8e943ff0dba83ca3b0a07

Overall, the code coverage is relatively low. We highly recommend improving the code coverage.

Changelog

- 2020-06-19 - Initial report
- 2020-10-13 - Final report based on PR#6327

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.