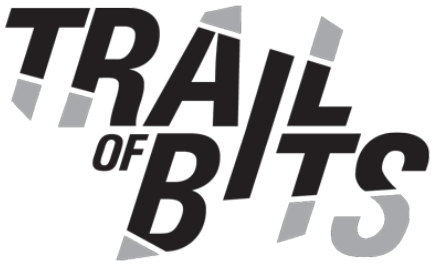

Prysm

Security Assessment

October 6, 2020

Prepared For:
Raul Jordan | *Prysmatic Labs*
raul@prysmaticlabs.com

Preston Van Loon | *Prysmatic Labs*
preston@prysmaticlabs.com

Danny Ryan | *Ethereum Foundation*
danny@ethereum.org

Prepared By:
Johanna Ratliff | *Trail of Bits*
johanna.ratliff@trailofbits.com

Sam Caccavale | *Trail of Bits*
sam.caccavale@trailofbits.com

Sam Moelius | *Trail of Bits*
sam.moelius@trailofbits.com

Executive Summary

From September 8 through October 2, 2020, the Ethereum Foundation and Prysmatic Labs engaged Trail of Bits to review the security of Prysm. Trail of Bits conducted this assessment over the course of 8 person-weeks with three engineers working from 245c18784eda370ea3218e8704651edad763978d from the Prysm repository.

During the first week, Trail of Bits familiarized ourselves with the codebase build and test systems, investigated fuzzing targets, and ran the existing fuzzers with no crashes. We ran automated tools to produce jumping off points for investigation including, but not limited to, `gosec`, `errcheck`, `ineffassign`, `semgrep`, and the unit tests with the race detector enabled.

Week two included shadowing investigation, arithmetic operation and overflow checks, and manual review of the slashing code. We also inspected the results from the previous weeks' tools with deeper scrutiny.

Week three Trail of Bits continued assessing the slashing and slashing protection code. We began mapping Eth2.0 specs against the project and manual code reading of the `beacon-chain/core` package. Finally, week three encompassed manual checks for best Go practices and known libraries whose misuse could produce bugs.

During the fourth week, we continued to review the Eth2.0 specification, looking for areas where Prysm may have diverged. We focused on the p2p and sync packages for potential denial of service attacks. Finally, we revisited the copy-on-write mechanism used for beacon-chain states.

Our efforts resulted in 26 findings ranging from high to informational severity. The single high-severity issue describes a failure scenario that causes a user's raw password to be logged (TOB-PRYSM-008). A few interesting medium-severity issues include missed deposits due to a premature loop exit (TOB-PRYSM-015) and preserving genesis state even on method failure (TOB-PRYSM-011).

In addition to our findings, there is an appendix (C. Code Quality Recommendations) that addresses issues that have not produced bugs, but are of concern.

Overall, the Prysm project can improve by focusing on best Go practices and tools to support them in order to remove risk of logic errors. We recommend extending the test suite to include more failing cases and develop more fuzzers to improve coverage of multiple scenarios.

Project Dashboard

Application Summary

Name	Prysm
Version	245c18784eda370ea3218e8704651edad763978d
Type	Go
Platforms	POSIX

Engagement Summary

Dates	September 8 - October 2, 2020
Method	Whitebox
Consultants Engaged	3
Level of Effort	8 person-weeks

Vulnerability Summary

Total High-Severity Issues	1	■
Total Medium-Severity Issues	10	■■■■■■■■■■
Total Low-Severity Issues	7	■■■■
Total Informational-Severity Issues	7	■■■■■■■■
Total Undetermined-Severity Issues	1	■
Total	26	

Category Breakdown

Access Controls	2	■■
Auditing and Logging	1	■
Cryptography	1	■
Data Validation	5	■■■■■
Denial of Service	5	■■■■■
Documentation	1	■
Error Reporting	4	■■■■
Patching	2	■■

Timing	2	■■
Undefined Behavior	3	■■■
Total	26	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Weak.** Higher severity issues related to improper handling of password management ranging from brute force vulnerability to plaintext password logging in rare circumstances. File and directory permissions were also not validated.
Arithmetic	**Strong.** Unit tests and existing fuzzers were run with arithmetic operations instrumented to check for overflow. No issues were found.
Assembly Use	**Not Applicable.**
Centralization	**Satisfactory.** Some concerns relating to operational practices such as committed bytecode (TOB-PRYSM-022). Additionally the consequences of raw password logging (TOB-PRYSM-008) and premature deposit exit (TOB-PRYSM-015) can cause issues but not system-wide.
Upgradeability	**Not Applicable.**
Function Composition	**Moderate.** The separation of responsibilities on an architecture level was clearly defined. However, on a more granular level the code was inconsistent in how DRY principle was intended. Concerning examples noted in the code quality section include single use functions defined in odd scopes that confused readability.
Front-Running	**Further Investigation Required.** Front-running opportunities could arise by deviating from the spec or by adhering to it (i.e., if the specification itself was flawed). No opportunities of the former kind were noted. Further investigation would be required for the latter.
Monitoring	**Not Considered.**
Specification	**Satisfactory.** With the exception of some operating system exceptions on the build documentation, the architectural diagrams and intended product purpose were well specified.
Testing & Verification	**Moderate.** When run against certain undocumented operating systems, the build failed. Additionally, on successful build, the eth1 chain had starting issues during the end-to-end tests. The fuzzers ran successfully and finally, there were a few common memory leaks in the tests.

Engagement Goals

The engagement was scoped to provide a security assessment of the beacon node, slashing, the validator client, and the `beacon-chain/core` package.

Specifically, we sought to answer the following questions:

- Is there a divergence in the code from the intended Eth2.0 specifications?
- Is the core beacon-node code vulnerable to consensus faults or other critical problems such as arithmetic overflows?
- Are there any logic errors in the block processing path from the moment a block is received in the beacon node and processed by the blockchain package?
- Does the slasher implementation contain flawed assumptions or risk false positives?
- What risk of denial-of-service attacks exist in the p2p and/or sync packages?

Coverage

Alignment with Eth2.0 Specifications. Through a manual investigation of the core components and their intent to ensure they aligned with the Eth2.0 specifications as read.

beacon-chain/core. Began with automated analysis tools such as gosec, ineffassign, errcheck, and go vet to surface any common Go issues. Continued with extensive manual code reading for logic validation and specific checks for Go issues, including concurrency behavior and shadowing issues.

Beacon Node. Automated analysis tools such as gosec, ineffassign, errcheck, and go vet to surface any common Go issues. Manually reviewed.

Block Processing. Subject to static analysis by `gosec`, `errcheck`, `ineffassign`, `shadow`, and `semgrep` (under various configurations). Unit tests verified to pass. Unit tests run: with Go's race detector enabled, with all uses of `recover` disabled, and with arithmetic operations instrumented to check for overflow. Fuzzed using existing fuzzers, but with arithmetic operations instrumented to check for overflow. Manually reviewed.

Concurrency. Using automated analysis tools such as `go test -race` and gosec, we discovered locations that had the potential to contain concurrency bugs. We then dove deeper into those scenarios for manual analysis to validate the assumptions made. Additionally, we investigated every goroutine that was created to ensure Go best practices in use of concurrency.

Slashing. Subject to static analysis by `gosec`, `errcheck`, `ineffassign`, `shadow`, and `semgrep` (under various configurations). Unit tests verified to pass. Unit tests run: with Go's race detector enabled, with all uses of `recover` disabled, and with arithmetic operations instrumented to check for overflow. Manually reviewed.

Validator Client. Automated analysis tools such as gosec, ineffassign, errcheck, and go vet to surface any common Go issues. Manually reviewed.

Dependencies. Checked for currency using `go-mod-outdated`. Checked for outstanding CVEs using `nancy`.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

□ **Adjust the return statements in Figures 1.1 through 1.4 to return an appropriate error.** This will prevent callers from assuming that returned values are computed without error. TOB-PRYSM-001

□ **Protect access to the `featureConfig` global variable with a `RWMutex` and have `featureconfig.Get` return a copy of the `Flags` struct rather than a pointer to it.** This will eliminate a data race that could lead to undefined behavior or data corruption. TOB-PRYSM-002

□ **Adjust the loop beginning in Figure 3.1 to use `defer span.End()`, as the official OpenCensus documentation suggests.** This will eliminate a `Context` leak that could lead to denial-of-service conditions. Consider adopting this pattern in `beacon-chain/sync/pending_blocks_queue.go`, as well. TOB-PRYSM-003

□ **Eliminate the use of variable shadowing in Figures 4.1 and 4.2.** This will help ensure that the code produces correct results, and will improve the code's readability. TOB-PRYSM-004

□ **Adjust the declaration of `sync.Service` so that `chainStarted` uses an `atomic` data type rather than a bool.** This will stop the Go race detector from producing reports concerning `chainStarted`, potentially enabling legitimate races to be found. TOB-PRYSM-005

□ **Document the operating systems that are explicitly supported to run these tests.** This will ensure users set up their own environment that matches a known good state and don't try to run the tests locally in an unsupported environment. TOB-PRYSM-007

□ **Remove any logging of passwords to replace it with a failure to access or similar.** TOB-PRYSM-008

□ **Update dependencies to the latest version wherever possible.** Verify that all unit tests pass following such updates. Using out-of-date dependencies could mean critical bug fixes are missed. TOB-PRYSM-009

□ **Upgrade to the most recent versions of `libp2p/go-libp2p-kad-dht` (`v0.10.0`) and `k8s.io/apimachinery` (`v11.0.0`).** Doing so will eliminate reliance on vulnerable code. TOB-PRYSM-010

□ **Store the result of the call to `ProcessPreGenesisDeposits` in a local variable, and use it to set `s.preGenesisState` only if `err` is not `nil`.** This will prevent previous work reflected in `s.preGenesisState` from being thrown away. TOB-PRYSM-011

□ **Consider adding a call to `snappy.DecodedLen` in `kv.decode`, similar to the one in `ssz.SszNetworkEncoder.DecodeGossip`.** This will provide defense in depth should an attacker find a way to write data into a beacon node's key-value store. TOB-PRYSM-012

□ **Move the use of `recover` from `hashutil.HashProto` to `TestHashProtoFuzz`.** Verify that the error is a `nil`-pointer dereference. These steps will help ensure that this particular use of `recover` does not unintentionally suppress errors. TOB-PRYSM-013

□ **Adjust the implementation of `goodFetcher.HeaderByNumber` so that it returns an error when `g.backend.Blockchain().CurrentHeader()` is `nil`.** This will eliminate a `nil`-pointer dereference that currently exists in the `TestLatestMainchainInfo_OK` test. TOB-PRYSM-014

□ **If a depositor's public key cannot be processed, skip over it and continue processing the remaining deposits.** This will eliminate a bug that currently allows valid deposits to be ignored. TOB-PRYSM-015

□ **Set a `Committees` struct's `Seed` when inserting proposer indices into the cache.** This will eliminate a bug that currently makes the struct unretrievable. TOB-PRYSM-016

□ **When inserting a shuffled list into the cache, add it to an existing `Committees` struct if one already exists with the relevant seed.** This will eliminate a bug that currently causes a node to build `Committees` structs unncessarily. TOB-PRYSM-017

□ **Refactor `BeaconProposerIndex`, `UpdateProposerIndicesInCache`, and `precomputeProposerIndices` so that the work done to compute the active validator indices and the proposer index for one slot is not duplicated.** Be careful to avoid introducing data races involving the presence of data in the cache in the process. These steps will improve the overall efficiency of the cache. TOB-PRYSM-018

□ **Adjust SlotSignature so that it uses `DOMAIN_SELECTION_PROOF` for selection proofs, rather than `DOMAIN_BEACON_ATTESTER`.** This will bring the code in-line with the Eth 2.0 current specification. TOB-PRYSM-019

□ **Adjust the `AncestorRoot`'s for loop condition so that it accounts for `i` not within the range of `f.store.nodes's indices`.** This will eliminate potential "index out of range" errors that could be used for denial of service. TOB-PRYSM-020

□ **Implement the `on_tick` method from the Eth 2.0 specification.** This will eliminate a discrepancy that currently exists between the Eth 2.0 specification and Prysm's implementation of it. TOB-PRYSM-021

□ **Read in the bytecode for deposits only from the bytecode.bin file that currently stores it.** Update pipelines so that only automation changes this value and it is never expected to be edited directly by developers. We also recommend putting in a git pre-commit hook to prevent developers from committing to this file. TOB-PRYSM-022

□ **When using `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on them.** This will help avoid situations where sensitive information is written to a pre-existing attacker-controlled path. TOB-PRYSM-023

□ **Validate that all file handlers are closed to avoid putting undue stress on the garbage collector.** TOB-PRYSM-024

□ **Limit the number of password guesses a user can attempt by implementing an exponential backoff strategy on password acceptance timing.** TOB-PRYSM-025

□ **Adjust `ancestor` so that if `s.forkChoiceStore.AncestorRoot` returns a `"node index out of range"` error, `beaconDB` is queried.** This will eliminate a bug that can cause `ancestor` to incorrectly return an error. TOB-PRYSM-026

Long term

□ **Ensure that unit tests are written for each of their "sad" (i.e., failing) paths.** This will help to expose functions that do not properly report errors. TOB-PRYSM-001

□ **Regularly run your unit tests with Go's race detector enabled.** Doing so can help reveal similar bugs in the future. TOB-PRYSM-002, TOB-PRYSM-005

□ **Ensure that the `defer span.End()` pattern is used where span is involved.** This will help prevent denial-of-service conditions. TOB-PRYSM-003

□ **Regularly run Go's `shadow` tool over your codebase.** These bugs are flagged by `shadow`, and regularly running shadow could reveal similar ones. TOB-PRYSM-004

□ **Regularly run ineffassign over the codebase as a part of a pre-commit action.** All results should either be explicitly underscored or used. TOB-PRYSM-006

□ **Include a Dockerfile of install and setup against a base image that can consistently run the end to end tests independent of operating system or local dependencies.** This instills confidence from the community in the open source project. TOB-PRYSM-007

□ **Validate that any engagements with passwords are closely monitored and restricted.** Consider using Vault for management of sensitive information. TOB-PRYSM-008

□ **Regularly run `go-mod-outdated` over the codebase to help ensure that the project stays up to date with its dependencies.** TOB-PRYSM-009

□ **Regularly run `nancy` over the codebase to reveal vulnerable dependencies.** TOB-PRYSM-010

□ **Whenever an error could be returned, avoid assigning to a field before first checking for an error.** Such practice could result in unintentional state modifications. TOB-PRYSM-011

□ **Protect calls related to serialization/deserialization with additional error checks.** Such functions are a common source of vulnerabilities and deserve elevated scrutiny. TOB-PRYSM-012

□ **Regularly run your unit tests with all uses of `recover` disabled.** Verify that all panics that would have been recovered are expected. These steps will help ensure that Prysm's uses of `recover`, generally, do not unintentionally suppress errors. TOB-PRYSM-013, TOB-PRYSM-014

□ **Be wary of returning from inside of a loop on an error condition.** Such a practice is a frequent source of errors. TOB-PRYSM-015

□ **Regularly run Go's `shadow` tool over your codebase.** This bug is due, in part, to variable shadowing in a unit test. Regularly running Go's shadow can help to reveal such problems. TOB-PRYSM-016

□ **Consider ways that the cache code may be consolidated.** This bug is, in part, the result of a discrepancy in how the cache handled proposer indices vs. shuffled indices. Consolidating that code might have prevented this bug. Furthermore, you may want to include other types of data in the cache in the future. Consolidating the cache code could help to facilitate such extensions. TOB-PRYSM-017

□ **As new types of data are added to the cache, avoid duplicating computations across functions that update the cache and their callers.** This will help maintain the cache's effectiveness. TOB-PRYSM-018

□ **When reviewing PRs, consider not only the code in the PR, but other code to which similar changes should be applied.** This issue appears to be the result of an incomplete fix. A more expansive approach to PR review could help to prevent similar situations. TOB-PRYSM-019

□ **Consider developing a fuzzer for `AncestorRoot` or one of its predecessors in the call graph.** Fuzzing might have caught this bug, and could catch similar bugs in the future. TOB-PRYSM-020

□ **Maintain a mapping of Eth 2.0 function to their corresponding implementations in Prysm.** This will make it easier for developers to understand how Prysm implements the specification. It will also help to identify gaps in the implementation. TOB-PRYSM-021

□ **Remove any manual processes required for releasing code and automate for consistency and reliability purposes.** TOB-PRYSM-022

□ **Enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use.** Ideally, this validation should be centrally defined and used throughout the applications as a whole. TOB-PRYSM-023

□ **Standardize idiomatic Go practices such as proper handling of response bodies.** TOB-PRYSM-024

 Implement monitoring surrounding excessively high password guess frequency against an account and consider locking it until the user can validate their identity. TOB-PRYSM-025

 Require additional review of functions whose implementations do not obviously match the specification. For example, as can be seen from Figure 26.1, Prysm's `ancestor` does not directly correspond to the specification's `get_ancestor`, though they are meant to achieve the same functionality. Functions such as `ancestor` warrant additional scrutiny. TOB-PRYSM-026

Findings Summary

#	Title	Type	Severity
1	Failure to propagate errors in `beacon-chain`	Error Reporting	Low
2	Data race in `featureconfig`	Timing	Low
3	Context leak in validator client	Denial of Service	Medium
4	Variable shadowing in beacon-chain history regeneration code	Undefined Behavior	Medium
5	Data race in beacon-chain syncing code	Timing	Informational
6	Failure to error check could cause undefined behavior	Error Reporting	Informational
7	Improvements needed to Bazel end-to-end testing documentation	Documentation	Informational
8	Raw password logged	Auditing and Logging	High
9	Dependencies are out of date	Patching	Informational
10	Code relies on two dependencies with OSS advisories	Patching	Low
11	Overwrite of `powchain.Service.preGenesisState` even when deposits cannot be processed	Denial of Service	Medium
12	Excessive memory usage by `snappy.Decode`	Denial of Service	Informational
13	Improper use of `recover` in `hashutil.HashProto`	Error Reporting	Informational

14	Improper use of recover in `powchain.Service`	Error Reporting	Low
15	Premature loop exit causes validator deposits to be missed	Data Validation	Medium
16	Proposer indices cannot be retrieved from cache	Data Validation	Medium
17	Shuffled list is not added to cache if proposer indices are present	Data Validation	Informational
18	Inefficiencies surrounding use of cache	Data Validation	Informational
19	Wrong domain used in `helpers.SlotSignature`	Cryptography	Medium
20	Wrong loop exit condition in `store.ForkChoice.AncestorRoot`	Denial of Service	Medium
21	Handler `on_tick` unimplemented	Undefined Behavior	Undetermined
22	Hardcoded contract bytecode	Data Validation	Medium
23	Files and directories created with os.MkdirAll and ioutil.WriteFile are not ensured to have correct permissions	Access Controls	Low
24	Response body not closed in tests	Denial of Service	Low
25	Password guesses should be limited to prevent brute force	Access Controls	Medium
26	Premature exit under certain conditions in `blockchain.Service.ancestor`	Undefined Behavior	Medium

1. Failure to propagate errors in `beacon-chain`

Severity: Low
Type: Error Reporting
Target: `beacon-chain`

Difficulty: Undetermined
Finding ID: TOB-PRYSM-001

Description
Many locations within the `beacon-chain` subdirectory fail to return applicable errors and instead return a nil error value. Failing to report errors can hide an ongoing attack, or make it difficult to discover the source of an error.

Examples where errors are not propagated appear in Figures 1.1 through 1.4.

```
func UpdateProposerIndicesInCache(state *stateTrie.BeaconState, epoch uint64) error {
	indices, err := ActiveValidatorIndices(state, epoch)
	if err != nil {
		return nil
	 }
```

Figure 1.1: beacon-chain/core/helpers/committee.go#L349-L353.

```
func ComputeForkDigest(version []byte, genesisValidatorsRoot []byte) ([4]byte, error) {
	dataRoot, err := computeForkDataRoot(version, genesisValidatorsRoot)
	if err != nil {
		return [4]byte{}, nil
	}
```

Figure 1.2: beacon-chain/core/helpers/signing_root.go#L246-L250.

```
func ComputeProposerIndex(bState *stateTrie.BeaconState, activeIndices []uint64, seed
[32]byte) (uint64, error) {
	...
		v, err := bState.ValidatorAtIndexReadOnly(candidateIndex)
		if err != nil {
			return 0, nil
		}
```

Figure 1.3: beacon-chain/core/helpers/validators.go#L232-L254.

```
func (kv *Store) saveArchivedInfo(ctx context.Context,
	currentState *stateTrie.BeaconState,
	blocks []*ethpb.SignedBeaconBlock,
) error {
	...
	lastBlocksRoot, err := blocks[len(blocks)-1].Block.HashTreeRoot()
	if err != nil {
		return nil
	}
```

Figure 1.4: beacon-chain/db/kv/regen_historical_states.go#L217-L232.

Exploit Scenario

Eve discovers an exploitable bug in the Prysm's `beacon-chain` implementation and attempts to exploit it on Alice's machine. Because errors are not properly reported, Alice is initially unaware that an attack is occuring. Once she becomes aware, she is unable to determine the source of the errors.

Recommendation
Short term, adjust the return statements in Figures 1.1 through 1.4 to return an appropriate error. This will prevent callers from assuming that returned values are computed without error.

Long term, as new functions are added to the code base, ensure that unit tests are written for each of their "sad" (i.e., failing) paths. This will help to expose functions that do not properly report errors.

2. Data race in `featureconfig`

Severity: Low
Type: Timing
Target: `shared/featureconfig/config.go`

Difficulty: High
Finding ID: TOB-PRYSM-002

Description
The `featureconfig` package contains a global variable `featureConfig` (with a capital 'C') that points to a `Flags` struct. This variable is modified in some threads and read in others with no synchronization. Unsynchronized, concurrent access to shared resources can lead to undefined behavior and possibly data corruption.

The `featureConfig` variable is set by calling `Init`, `InitWithReset`, or the "reset" function returned by `InitWithReset`. The variable is read by calling `featureconfig.Get` (Figure 2.1). An example where `featureconfig.Get` is called from a separate thread appears in Figure 2.2. A second example appears in Figures 2.3 through 2.5.

```
// Get retrieves feature config.
func Get() *Flags {
	if featureConfig == nil {
		return &Flags{}
	}
	return featureConfig
}

// Init sets the global config equal to the config that is passed in.
func Init(c *Flags) {
	featureConfig = c
}

// InitWithReset sets the global config and returns function that is used to reset
configuration.
func InitWithReset(c *Flags) func() {
	resetFunc := func() {
		Init(&Flags{})
	}
	Init(c)
	return resetFunc
}
```

Figure 2.1: shared/featureconfig/config.go#L100-L112.

```
	go func() {
		...
				recalibrateRoughtime()
		...
	}()
...
func recalibrateRoughtime() {
	t0 := time.Now()
	...
}
...
```

```
func Now() time.Time {
        if featureconfig.Get().EnableRoughtime {
                return time.Now().Add(offset)
        }
```

Figure 2.2: shared/roughtime/roughtime.go#L45-L117.

```
func (s *Service) Start() {
        ...
        runutil.RunEvery(s.ctx, syncMetricsInterval, s.updateMetrics)
}
```

Figure 2.3: beacon-chain/sync/service.go#L144-L162.

```
func RunEvery(ctx context.Context, period time.Duration, f func()) {
        ...
        go func() {
                ...
                                f()
                ...
        }()
}
```

Figure 2.4: shared/runutil/every.go#L15-L31.

```
func (s *Service) updateMetrics() {
        ...
        if featureconfig.Get().DisableDynamicCommitteeSubnets {
```

Figure 2.5: beacon-chain/sync/metrics.go#L83-L97.

Exploit Scenario
Eve finds a way to exploit the race in Figures 2.1 through 2.5 to cause data corruption. Eve carries out the attack on Alice's machine, causing Alice to suffer data loss.

Recommendation
Short term, protect access to the `featureConfig` global variable with a RWMutex and have `featureconfig.Get` return a copy of the `Flags` struct rather than a pointer to it. This will eliminate a data race that could lead to undefined behavior or data corruption.

Long term, regularly run your unit tests with Go's race detector enabled. Doing so can help reveal similar bugs in the future.

References
- Introducing the Go Race Detector
- Data Race Detector

3. Context leak in validator client

Severity: Medium
Type: Denial of Service
Target: `validator/client/runner.go`

Difficulty: Low
Finding ID: TOB-PRYSM-003

Description
The function `Validator.run` allocates a `Span` but does not call `End` on it along all control-flow paths. Thus, its associated `Context` can be leaked. An attacker could exploit this leak for denial of service.

The allocation appears in Figure 3.1. Code paths that do not call `End` include those in Figure 3.2 and 3.3.

```
	for {
		ctx, span := trace.StartSpan(ctx, "validator.processSlot")

		select {
```

Figure 3.1: validator/client/runner.go#L84-L85.

```
			if featureconfig.Get().LocalProtection {
				if err := v.UpdateProtections(ctx, slot); err != nil {
					log.WithError(err).Error("Could not update validator
protection")
					continue
				}
			}
```

Figure 3.2: validator/client/runner.go#L84-L85.

```
			allRoles, err := v.RolesAt(ctx, slot)
			if err != nil {
				log.WithError(err).Error("Could not get validator roles")
				continue
			}
```

Figure 3.3: validator/client/runner.go#L125-L129.

OpenCensus's official documentation recommends using the following pattern:

```
ctx, span := trace.StartSpan(ctx, "cache.Get")
defer span.End()
```

This pattern is not adhered to in the above code. Based on Slack discussion, choosing not to adhere to this pattern may have led to other bugs in `beacon-chain/sync/pending_blocks_queue.go`, as well.

Exploit Scenario

Eve discovers a code path that exercises the loop beginning in Figure 3.1 repeatedly. Eve uses this code path to cause denial-of-service conditions in targeted nodes.

Recommendation
Short term, adjust the loop beginning in Figure 3.1 to use `defer span.End()`, as the official OpenCensus documentation suggests. This will eliminate a `Context` leak that could lead to denial-of-service conditions. Consider adopting this pattern in `beacon-chain/sync/pending_blocks_queue.go`, as well.

Long term, as new code that uses `Span`s is added to the codebase, ensure that the `defer span.End()` pattern is used. This will help prevent denial-of-service conditions.

4. Variable shadowing in beacon-chain history regeneration code

Severity: Medium | Difficulty: Undetermined
Type: Undefined Behavior | Finding ID: TOB-PRYSM-004
Target: `beacon-chain/db/kv/regen_historical_states.go`

Description
In two locations in `regen_historical_states.go`, a local variable declaration shadows another declaration. The affected code appears to at best perform unnecessary work.

The code in question appears in Figures 4.1 and 4.2. In Figure 4.1, the local variable `currentState` shadows another variable of the same name. Following the if statement with condition "`lastArchivedSlot > 0`," `currentState` will refer to a copy of the genesis state. This appears unintentional. Under certain conditions, that copy of the genesis state could be passed to `regenHistoricalStateProcessSlots`, resulting in unnecessary work.

```
func (kv *Store) regenHistoricalStates(ctx context.Context) error {
	...
	genesisState, err := kv.GenesisState(ctx)
	if err != nil {
		return err
	}
	currentState := genesisState.Copy()
	startSlot := genesisState.Slot()
	...
	lastArchivedSlot, err := kv.LastArchivedSlot(ctx)
	if err != nil {
		return err
	}
	if lastArchivedSlot > 0 {
		archivedIndexStart := lastArchivedSlot - 1
		archivedRoot := kv.ArchivedPointRoot(ctx, archivedIndexStart)
		currentState, err := kv.State(ctx, archivedRoot)
		if err != nil {
			return err
		}
		startSlot = currentState.Slot()
	}
	...
	for slot := lastArchivedSlot; slot <= lastSavedBlockArchivedSlot; slot++ {
		...
		targetSlot := startSlot + slotsPerArchivedPoint
		filter := filters.NewFilter().SetStartSlot(startSlot +
1).SetEndSlot(targetSlot)
		blocks, err := kv.Blocks(ctx, filter)
		if err != nil {
			return err
		}

		// Replay blocks and replay slots if necessary.
		if len(blocks) > 0 {
			...
		}
		if targetSlot > currentState.Slot() {
```

```
                        currentState, err = regenHistoricalStateProcessSlots(ctx,
currentState, targetSlot)
                        if err != nil {
                                return errors.Wrap(err, "could not regenerate historical
process slot")
                        }
                }
```

Figure 4.1: beacon-chain/db/kv/regen_historical_states.go#L26-L113.

In Figure 4.2, the local variable `state` shadows a function argument. The local variable is initialized with a value returned by `transition.ProcessSlot`. In this case, there does not appear to be a problem, since `transition.ProcessSlot` returns either `nil` or the `state` passed to it. Nonetheless, the shadowing should be eliminated so that it does not break if `transition.ProcessSlot`'s behavior changes.

```
func regenHistoricalStateProcessSlots(ctx context.Context, state *stateTrie.BeaconState,
slot uint64) (*stateTrie.BeaconState, error) {
        ctx, span := trace.StartSpan(ctx, "BeaconDB.regenHistoricalStateProcessSlots")
        defer span.End()
        if state == nil {
                return nil, errors.New("state can't be nil")
        }
        if state.Slot() > slot {
                err := fmt.Errorf("expected state.slot %d < slot %d", state.Slot(), slot)
                return nil, err
        }
        if state.Slot() == slot {
                return state, nil
        }
        for state.Slot() < slot {
                state, err := transition.ProcessSlot(ctx, state)
                if err != nil {
                        return nil, errors.Wrap(err, "could not process slot")
                }
                if transition.CanProcessEpoch(state) {
                        state, err = transition.ProcessEpochPrecompute(ctx, state)
                        if err != nil {
                                return nil, errors.Wrap(err, "could not process epoch with
optimizations")
                        }
                }
                if err := state.SetSlot(state.Slot() + 1); err != nil {
                        return nil, err
                }
        }
        return state, nil
}
```

Figure 4.2: beacon-chain/db/kv/regen_historical_states.go#L165-L194.

Exploit Scenario

Alice runs a slasher that is unable to identify slashable offenses because its associated beacon node cannot properly reconstruct the beacon chain's history.

Recommendation

Short term, eliminate the use of variable shadowing in Figures 4.1 and 4.2. This will help ensure that the code produces correct results, and will improve the code's readability.

Long term, regularly run Go's `shadow` tool over your codebase. These bugs are flagged by `shadow`, and regularly running shadow could reveal similar ones.

5. Data race in `beacon-chain` syncing code

Severity: Informational | Difficulty: Not applicable
Type: Timing | Finding ID: TOB-PRYSM-005
Target: `beacon-chain/sync/service.go`, `beacon-chain/sync/subscriber.go`

Description
The `beacon-chain` `sync.Service` sets a flag `chainStarted` to indicate that no more subscription requests should be issued. Access to this flag is unsynchronized.

The flag is set in Figure 5.1, and read in Figure 5.2. The race concerning this flag appears to be innocuous. Nonetheless, eliminating the race by using an [atomic] data type should add minimal overhead. Moreover, it would eliminate noise produced by the Go race detector, potentially allowing legitimate races to be discovered.

```
func (s *Service) registerHandlers() {
        ...
        for !s.chainStarted {
                select {
                case event := <-stateChannel:
                        if event.Type == statefeed.Initialized {
                                ...
                                s.chainStarted = true
```

Figure 5.1: beacon-chain/sync/service.go#L229-L253.

```
func (s *Service) subscribeDynamicWithSubnets(
        topicFormat string,
        validate pubsub.ValidatorEx,
        handle subHandler,
) {
        ...
        go func() {
                for {
                        select {
                        case <-s.ctx.Done():
                                ticker.Done()
                                return
                        case currentSlot := <-ticker.C():
                                if s.chainStarted && s.initialSync.Syncing() {
                                        continue
                                }
                                ...
                        }
```

Figure 5.2: beacon-chain/sync/subscriber.go#L228-L275.

Exploit Scenario
Alice is a Prysm developer tasked with identifying data races. Alice misses a legitimate data race because it is hidden among innocuous races such as the one in Figures 5.1 and 5.2.

Recommendation

Short term, adjust the declaration of `sync.Service` so that `chainStarted` uses an `atomic` data type rather than a bool. This will stop the Go race detector from producing reports concerning `chainStarted`, potentially enabling legitimate races to be found.

Long term, regularly run your unit tests with Go's race detector enabled. Doing so can help reveal similar bugs in the future.

References

- Introducing the Go Race Detector
- Data Race Detector

6. Failure to error check could cause undefined behavior

Severity: Low
Type: Error Reporting
Target: `multiple locations`

Difficulty: Undetermined
Finding ID: TOB-PRYSM-006

Description
Throughout the codebase, there are multiple locations where a returned error value goes unchecked before using the (possibly nil) associated result. One such instance of this (shown in figure 6.1) may cause a validator to fail to acknowledge an error and could cause cascading undefined behavior. These errors should be checked for a failure state before using the function results.

```
uAtts, err = vs.filterAttestationsForBlockInclusion(ctx, latestState, uAtts)
atts = append(atts, uAtts...)
```

Figure 6.1: beacon-chain/rpc/validator/proposer.go#L692.

Appendix D is the output of filtering ineffassign with the command in figure 6.2. However, there isn't a particularly good reason to leave ineffectual assignments in test code.

```
ineffassign ./... | grep -v "test" | sed -E "s;.*src([^:]+):([0-9]+):[^ ]+ (.*);\3 |
$GITHUB_URL\1#L\2;"
```

Figure 6.2.

Exploit Scenario
An error in `filterAttestationsForBlockInclusion` will result in `uAtts` being `nil`. Go will panic attempting to append the explosion of nil on the next line.

Recommendation
In the short term, items in the appendix should be manually checked and fixed. This is especially important for `err` values.

Long term, regularly run ineffassign over the codebase as a part of a pre-commit action. All results should either be explicitly underscored or used.

7. Improvements needed to Bazel end-to-end testing documentation

Severity: Informational
Type: Documentation
Target: `bazel test on Mac OS Catalina`

Difficulty: Not applicable
Finding ID: TOB-PRYSM-007

Description
A difference between Mac OS Catalina and linux caused the error in Figure 7.1 when running the end-to-end tests using Bazel.This error caused an inability to run the tests locally on a Mac.

```
=== RUN   TestEndToEnd_Slashing_MinimalConfig
   TestEndToEnd_Slashing_MinimalConfig: endtoend_test.go:36: Shard index: 0
   TestEndToEnd_Slashing_MinimalConfig: endtoend_test.go:37: Starting time: 2020-09-15
18:59:19.377007 +0000 UTC m=+87.985860479
   TestEndToEnd_Slashing_MinimalConfig: endtoend_test.go:38: Log Path:
/private/var/tmp/_bazel_johanna/8795d77cc667fd99d49d6319fba8c083/sandbox/darwin-sandbox/267/
execroot/prysm/bazel-out/darwin-fastbuild/testlogs/endtoend/go_default_test/test.outputs
   TestEndToEnd_Slashing_MinimalConfig: eth1.go:73: mining log not found, this means the
eth1 chain had issues starting: bufio.Scanner: token too long
--- FAIL: TestEndToEnd_Slashing_MinimalConfig (2.01s)
```

Figure 7.1 end-to-end test error message

Recommendation
Short term, document the operating systems that are explicitly supported to run these tests. This will ensure users set up their own environment that matches a known good state and don't try to run the tests locally in an unsupported environment.

Long term, include a Dockerfile of install and setup against a base image that can consistently run the end to end tests independent of operating system or local dependencies. This instills confidence from the community in the open source project.

8. Raw password logged

Severity: High
Type: Auditing and Logging
Target: `tools/sendDepositTx/sendDeposits.go`

Difficulty: Medium
Finding ID: TOB-PRYSM-008

Description
Within the `main` function of `sendDeposits`, the `rawPassword` is loaded from a text file for use on the keystore and on failure, logs the `rawPassword`.

```
// Load from keystore
        store := prysmKeyStore.NewKeystore(prysmKeystorePath)
        rawPassword := loadTextFromFile(passwordFile)
        prefix := params.BeaconConfig().ValidatorPrivkeyFileName
        validatorKeys, err = store.GetKeys(prysmKeystorePath, prefix, rawPassword, false /*
warnOnFail */)
        if err != nil {
                log.WithField("path", prysmKeystorePath).WithField("password",
rawPassword).Errorf("Could not get keys: %v", err)

        }
```

Figure 8.1: sendDepositTx/sendDeposits.go#L179-L186.

Exploit Scenario
Bob is a Prysm user sending a deposit. `store.GetKeys` fails causing a log containing the `rawPassword` from the password textfile to be emitted. Eve has access to the logs and retrieves Bob's password.

Recommendation
Short term, remove any logging of passwords to replace it with a failure to access or similar.

Long term, validate that any engagements with passwords are closely monitored and restricted. Consider using Vault for management of sensitive information.

9. Dependencies are out of date

Severity: Informational
Type: Patching
Target: Various

Difficulty: Undetermined
Finding ID: TOB-PRYSM-009

Description
Updated versions of many of Prysm's dependencies are available. Since silent bug fixes are common, the dependencies should be reviewed and updated wherever possible.

Dependencies that can be updated appear in Table 9.1. Assuming semantic versioning is followed, many of the updates are bug fixes (highlighted).

Dependency	Version currently in use	Latest version available
`contrib.go.opencensus.io/exporter/jaeger`	`v0.2.0`	`v0.2.1`
`github.com/aristanetworks/goarista`	`v0.0.0-20200521140103-6c3304613b30`	`v0.0.0-20200812190859-4cb0e71f3c0e`
`github.com/bazelbuild/buildtools`	`v0.0.0-20200528175155-f4e8394f069d`	`v0.0.0-20200925145723-e6efbf6df90b`
`github.com/bazelbuild/rules_go`	`v0.23.2`	`v0.24.3`
`github.com/btcsuite/btcd`	`v0.20.1-beta`	`v0.21.0-beta`
`github.com/cloudflare/roughtime`	`v0.0.0-20200205191924-a69ef1dab727`	`v0.0.0-20200911173848-eb42b5b8e068`
`github.com/emicklei/dot`	`v0.11.0`	`v0.14.0`
`github.com/ethereum/go-ethereum`	`v0.0.0-20200626171358-a933315235ec`	`v0.0.0-20200922100503-3794d0d7ca25`
`github.com/fsnotify/fsnotify`	`v1.4.7`	`v1.4.9`
`github.com/gofrs/flock`	`v0.7.1`	`v0.8.0`
`github.com/golang/gddo`	`v0.0.0-20200528160355-8d077c1d8f4c`	`v0.0.0-20200831202555-721e228c7686`
`github.com/golang/mock`	`v1.4.3`	`v1.4.4`

github.com/golang/snappy	v0.0.1	v0.0.2
github.com/google/gofuzz	v1.1.0	v1.2.0
github.com/google/uuid	v1.1.1	v1.1.2
github.com/grpc-ecosystem/go-grpc-middleware	v1.2.0	v1.2.2
github.com/grpc-ecosystem/grpc-gateway	v1.14.6	v1.15.0
github.com/herumi/bls-eth-go-binary	v0.0.0-20200722032157-41fc56eba7b4	v0.0.0-20200923072303-32b29e5d8cbf
github.com/kr/pretty	v0.2.0	v0.2.1
github.com/libp2p/go-libp2p	v0.10.2	v0.11.0
github.com/libp2p/go-libp2p-circuit	v0.3.1	v0.4.0
github.com/libp2p/go-libp2p-core	v0.6.1	v0.7.0
github.com/libp2p/go-libp2p-kad-dht	v0.8.3	v0.10.0
github.com/libp2p/go-libp2p-pubsub	v0.3.3	v0.3.5
github.com/libp2p/go-libp2p-swarm	v0.2.8	v0.3.1
github.com/manifoldco/promptui	v0.7.0	v0.8.0
github.com/minio/highwayhash	v1.0.0	v1.0.1
github.com/multiformats/go-multiaddr	v0.2.2	v0.3.1
github.com/multiformats/go-multiaddr-net	v0.1.5	v0.2.0
github.com/pborman/uuid	v1.2.0	v1.2.1

github.com/prysmaticlabs/ethereumapis	v0.0.0-20200827165051-58ccb36e36b9	v0.0.0-20200923224139-64c46fb1b0fa
github.com/schollz/progressbar/v3	v3.3.4	v3.5.1
github.com/sirupsen/logrus	v1.6.0	v1.7.0
github.com/wealdtech/eth2-signer-api	v1.3.0	v1.5.2
github.com/wealdtech/go-eth2-util	v1.5.0	v1.6.0
github.com/wealdtech/go-eth2-wallet	v1.12.0	v1.14.0
github.com/wealdtech/go-eth2-wallet-types/v2	v2.6.0	v2.7.0
go.etcd.io/bbolt	v1.3.4	v1.3.5
golang.org/x/crypto	v0.0.0-20200728195943-123391ffb6de	v0.0.0-20200820211705-5c72a883971a
golang.org/x/exp	v0.0.0-20200513190911-00229845015e	v0.0.0-20200924195034-c827fd4f18b9
golang.org/x/tools	v0.0.0-20200528185414-6be401e3f76e	v0.0.0-20200928112810-42b62fc93869
google.golang.org/genproto	v0.0.0-20200730144737-007c33dbd381	v0.0.0-20200925023002-c2d885f95484
google.golang.org/grpc	v1.29.1	v1.32.0
k8s.io/api	v0.18.3	v0.19.2
k8s.io/apimachinery	v0.18.3	v0.19.2
k8s.io/client-go	v0.18.3	v11.0.0+incompatible

Table 9.1: Dependencies for which updates are available.

Exploit Scenario
Eve learns of a vulnerability in an old version of a Prysm dependency and uses it to exploit Alice's Prysm installation.

Recommendation

Short term, update dependencies to the latest version wherever possible. Verify that all unit tests pass following such updates. Using out-of-date dependencies could mean critical bug fixes are missed.

Long term, regularly run `go-mod-outdated` over the codebase to help ensure that the project stays up to date with its dependencies.

10. Code relies on two dependencies with OSS advisories

Severity: Low | Difficulty: High
Type: Patching | Finding ID: TOB-PRYSM-010
Target: `tools/{bootnode-query/main.go, cluster-pk-manager/server/watchtower.go, deployContract/deployContract.go}`

Description
By depending on `etcd@3.3.10`, the `bootnode-query` tool is potentially vulnerable to CVE-2018-17142, CVE-2018-17143, CVE-2018-17846, CVE-2018-17847, and CVE-2018-17848. By depending on `golang.org/x/net v0.0.0-20170114055629-f2499483f923`, the `cluster-pk-manager` and `deployContract` tools are potentially vulnerable to CVE-2020-15114, CVE-2020-15115, and CVE-2020-15136.

The dependency chain leading to `etcd@3.3.10` appears in Figure 10.1. The dependency chain leading to `golang.org/x/net v0.0.0-20170114055629-f2499483f923` appears in Figure 10.2.

```
github.com/coreos/etcd@v3.3.10+incompatible
github.com/spf13/viper@v1.3.2
github.com/spf13/cobra@v0.0.5
github.com/dgraph-io/badger@v1.6.1
github.com/ipfs/go-ds-badger@v0.2.3
github.com/libp2p/go-libp2p-peerstore@v0.2.6
github.com/libp2p/go-libp2p-kad-dht@v0.8.3
```

Figure 10.1: Dependency chain leading to `etcd@3.3.10`.

```
golang.org/x/net@v0.0.0-20170114055629-f2499483f923
k8s.io/kube-openapi@v0.0.0-20200410145947-61e04a5be9a6
k8s.io/apimachinery@v0.18.3
```

Figure 10.2: Dependency chain leading to `golang.org/x/net v0.0.0-20170114055629-f2499483f923`.

We've given this finding low severity since it affects only Prysm's "tools," and not its core components (i.e., the beacon node, validator, or slasher).

Exploit Scenario
Alice, a Prysm developer, regularly uses Prsym tools. Eve uses vulnerabilities in those tools to exploit Alice's machine.

Recommendation
Short term, upgrade to the most recent versions of `libp2p/go-libp2p-kad-dht` (`v0.10.0`) and `k8s.io/apimachinery` (`v11.0.0`). Doing so will eliminate reliance on vulnerable code.

Long term, regularly run nancy over the codebase to reveal vulnerable dependencies.

11. Overwrite of `powchain.Service.preGenesisState` even when deposits cannot be processed

Severity: Medium
Type: Denial of Service
Target: `beacon-chain/powchain/deposit.go`

Difficulty: High
Finding ID: TOB-PRYSM-011

Description
If an error occurs processing Eth 1.0 deposits, the `powchain` service's `preGenesisState` field may be overwritten with `nil`, throwing away previous work.

The relevant code appears in Figure 11.1 and 11.2. Note that following the call to `ProcessPreGenesisDeposits`, `s.preGenesisState` is set even if `err` is not `nil`. Further note that there are several paths through `ProcessPreGenesisDeposits` where `nil` is returned in conjunction with an error.

```
func (s *Service) processDeposit(eth1Data *ethpb.Eth1Data, deposit *ethpb.Deposit) error {
	var err error
	if err := s.preGenesisState.SetEth1Data(eth1Data); err != nil {
		return err
	}
	s.preGenesisState, err = blocks.ProcessPreGenesisDeposits(context.Background(),
s.preGenesisState, []*ethpb.Deposit{deposit})
	return err
}
```

Figure 11.1: beacon-chain/powchain/deposit.go#L10-L17.

```
func ProcessPreGenesisDeposits(
	ctx context.Context,
	beaconState *stateTrie.BeaconState,
	deposits []*ethpb.Deposit,
) (*stateTrie.BeaconState, error) {
	var err error
	beaconState, err = ProcessDeposits(ctx, beaconState, &ethpb.SignedBeaconBlock{
		Block: &ethpb.BeaconBlock{Body: &ethpb.BeaconBlockBody{Deposits: deposits}}})
	if err != nil {
		return nil, errors.Wrap(err, "could not process deposit")
	}
	for _, deposit := range deposits {
		pubkey := deposit.Data.PublicKey
		index, ok := beaconState.ValidatorIndexByPubkey(bytesutil.ToBytes48(pubkey))
		if !ok {
			return beaconState, nil
		}
		balance, err := beaconState.BalanceAtIndex(index)
		if err != nil {
			return nil, err
		}
		validator, err := beaconState.ValidatorAtIndex(index)
		if err != nil {
			return nil, err
		}
```

```
                validator.EffectiveBalance =
mathutil.Min(balance-balance%params.BeaconConfig().EffectiveBalanceIncrement,
params.BeaconConfig().MaxEffectiveBalance)
                if validator.EffectiveBalance ==
                        params.BeaconConfig().MaxEffectiveBalance {
                        validator.ActivationEligibilityEpoch = 0
                        validator.ActivationEpoch = 0
                }
                if err := beaconState.UpdateValidatorAtIndex(index, validator); err != nil {
                        return nil, err
                }
        }
        return beaconState, nil
}
```

Figure 11.2: beacon-chain/core/blocks/deposit.go#L22-L58.

Exploit Scenario

Eve sends traffic to exercise one of the highlighted paths in Figure 11.2 just prior to the launch of Eth 2.0. This puts Prysm nodes into an invalid state, making them unable to participate in the launch.

Recommendation

Short term, store the result of the call to `ProcessPreGenesisDeposits` in a local variable, and use it to set `s.preGenesisState` only if `err` is not `nil`. This will prevent previous work reflected in `s.preGenesisState` from being thrown away.

Long term, whenever an error could be returned, avoid assigning to a field before first checking for an error. Such practice could result in unintentional state modifications.

12. Excessive memory usage by `snappy.Decode`

Severity: Informational
Type: Denial of Service
Target: `beacon-chain/db/kv/encoding.go`

Difficulty: High
Finding ID: TOB-PRYSM-012

Description
Prysm beacon nodes limit the maximum size of data they will `snappy.Decode` when the data is received via gossip (Figure 12.1). However, they do not exercise the same precautions in regard to data read from the key-value store (Figure 12.2). Exercising such precautions would provide defense in depth.

```
func (e SszNetworkEncoder) DecodeGossip(b []byte, to interface{}) error {
        size, err := snappy.DecodedLen(b)
        if uint64(size) > MaxGossipSize {
                return errors.Errorf("gossip message exceeds max gossip size: %d bytes > %d
bytes", size, MaxGossipSize)
        }
        b, err = snappy.Decode(nil /*dst*/, b)
        if err != nil {
                return err
        }
        return e.doDecode(b, to)
}
```

Figure 12.1: beacon-chain/p2p/encoder/ssz.go#L89-L99.

```
func decode(ctx context.Context, data []byte, dst proto.Message) error {
        ctx, span := trace.StartSpan(ctx, "BeaconDB.decode")
        defer span.End()

        data, err := snappy.Decode(nil, data)
        if err != nil {
                return err
        }
        if isSSZStorageFormat(dst) {
                return dst.(fastssz.Unmarshaler).UnmarshalSSZ(data)
        }
        return proto.Unmarshal(data, dst)
}
```

Figure 12.2: beacon-chain/db/kv/encoding.go#L16-L28.

Exploit Scenario
Eve finds a way to store data of her choosing in a beacon node's key-value store. Eve writes a malicious payload to the store of Alice's node. When the node tries to decode the payload, it exhausts its memory and crashes.

Recommendation
Short term, consider adding a call to `snappy.DecodedLen` in `kv.decode`, similar to the one in `ssz.SszNetworkEncoder.DecodeGossip`. This will provide defense in depth should an attacker find a way to write data into a beacon node's key-value store.

Long term, protect calls related to serialization/deserialization with additional error checks. Such functions are a common source of vulnerabilities and deserve elevated scrutiny.

13. Improper use of `recover` in `hashutil.HashProto`

Severity: Informational
Type: Error Reporting
Target: `shared/hashutil/hash.go`

Difficulty: Not applicable
Finding ID: TOB-PRYSM-013

Description
The code and comments in `hashutil.HashProto` suggest the only way a panic can occur is through a `nil`-pointer dereference (Figures 13.1 and 13.2). This assumption appears invalid, since many other error types seem just as plausible.

```
var ErrNilProto = errors.New("cannot hash a nil protobuf message")
```

Figure 13.1: shared/hashutil/hash.go#L20.

```
func HashProto(msg proto.Message) (result [32]byte, err error) {
	// Hashing a proto with nil pointers will cause a panic in the unsafe
	// proto.Marshal library.
	defer func() {
		if r := recover(); r != nil {
			err = ErrNilProto
		}
	}()
```

Figure 13.2: shared/hashutil/hash.go#L106-L113.

It appears that the use of `recover` in Figure *13.2* may have been added to deal with panics caused by `TestHashProtoFuzz`. Putting that use of `recover` directly in `TestHashProtoFuzz` would reduce its likelihood of unintentionally suppressing errors.

```
func TestHashProtoFuzz(t *testing.T) {
	f := fuzz.New().NilChance(.2)

	for i := 0; i < 1000; i++ {
		msg := &pb.AddressBook{}
		f.Fuzz(msg)
		_, err := hashutil.HashProto(msg)
		_ = err
	}
}
```

Figure 13.2: shared/hashutil/hash_test.go#L72-L81.

Exploit Scenario
Eve discovers a bug in `hashutil.HashProto`, and attempts to exploit it in Alice's Prysm installation. Such attempts can fail, so Eve tries repeatedly, relying on the fact that the `hashutil.HashProto` incorrectly reports her attempts as `nil`-pointer dereferences.

Recommendation

Short term, move the use of `recover` from `hashutil.HashProto` to `TestHashProtoFuzz`. Verify that the error is a `nil`-pointer dereference. These steps will help ensure that this particular use of `recover` does not unintentionally suppress errors.

Long term, regularly run your unit tests with all uses of `recover` disabled. Verify that all panics that would have been recovered are expected. These steps will help ensure that Prysm's uses of `recover`, generally, do not unintentionally suppress errors.

14. Improper use of `recover` in `powchain.Service`

Severity: Low | Difficulty: Undetermined
Type: Error Reporting | Finding ID: TOB-PRYSM-014
Target: `beacon-chain/powchain/service.go`, `beacon-chain/powchain/service_test.go`

Description
The `powchain` service uses `recover` to handle panics that could occur in `Service.handleETH1FollowDistance`. This use of `recover` is hiding a `nil`-pointer dereference that currently exists in the `TestLatestMainchainInfo_OK` test.

Go-ethereum's `core.BlockChain.GetHeaderByNumber` method does not return an error, but merely returns `nil` under abnormal conditions. This can cause the `goodFetcher.HeaderByNumber` to return `nil, nil` (Figures 14.1 through 14.3).

```
func (g *goodFetcher) HeaderByNumber(ctx context.Context, number *big.Int)
(*gethTypes.Header, error) {
	if g.backend == nil {
		return &gethTypes.Header{
			Number: big.NewInt(15),
			Time:   150,
		}, nil
	}
	if number == nil {
		return g.backend.Blockchain().CurrentHeader(), nil
	}
	return g.backend.Blockchain().GetHeaderByNumber(number.Uint64()), nil
}
```

Figure 14.1: beacon-chain/powchain/service_test.go#L90-L101.

```
// GetHeaderByNumber retrieves a block header from the database by number,
// caching it (associated with its hash) if found.
func (bc *BlockChain) GetHeaderByNumber(number uint64) *types.Header {
	return bc.hc.GetHeaderByNumber(number)
}
```

Figure 14.2: go-ethereum/core/blockchain.go#L2426-L2430.

```
// GetHeaderByNumber retrieves a block header from the database by number,
// caching it (associated with its hash) if found.
func (hc *HeaderChain) GetHeaderByNumber(number uint64) *types.Header {
	hash := rawdb.ReadCanonicalHash(hc.chainDb, number)
	if hash == (common.Hash{}) {
		return nil
	}
	return hc.GetHeader(hash, number)
}
```

Figure 14.3: go-ethereum/core/headerchain.go#L461-L469.

A return of `nil, nil` from `goodFetcher.HeaderByNumber` causes a `nil`-pointer dereference in the `TestLatestMainchainInfo_OK` test (Figures 14.4 through 14.6). The `nil`

`header` is passed to `s.headerCache.AddHeader`. The `nil`-pointer dereference results when `s.headerCache.AddHeader` tries to fetch `header`'s `Number` field.

```
	header, err := s.eth1DataFetcher.HeaderByNumber(ctx, height)
	if err != nil {
		return [32]byte{}, errors.Wrap(err, fmt.Sprintf("could not query header with
height %d", height.Uint64()))
	}
	if err := s.headerCache.AddHeader(header); err != nil {
```

Figure 14.4: beacon-chain/powchain/block_reader.go#L51-L55.

```
func (b *headerCache) AddHeader(hdr *gethTypes.Header) error {
	b.lock.Lock()
	defer b.lock.Unlock()

	hInfo := headerToHeaderInfo(hdr)
```

Figure 14.5: beacon-chain/powchain/block_cache.go#L150-L154.

```
func headerToHeaderInfo(hdr *gethTypes.Header) *headerInfo {
	return &headerInfo{
		Hash:   hdr.Hash(),
		Number: new(big.Int).Set(hdr.Number),
		Time:   hdr.Time,
	}
}
```

Figure 14.6: beacon-chain/powchain/block_cache.go#L48-L54.

Finally, the `nil`-pointer dereference is suppressed by the use of `safelyHandlePanic()` inside `powchain.Service.handleETH1FollowDistance` (Figures 14.7 through 14.10).

```
func TestLatestMainchainInfo_OK(t *testing.T) {
	...
	go func() {
		web3Service.run(web3Service.ctx.Done())
```

Figure 14.7: beacon-chain/powchain/block_reader_test.go#L29-L52.

```
func (s *Service) run(done <-chan struct{}) {
	...
	for {
		select {
		...
		case <-s.headTicker.C:
			...
			s.handleETH1FollowDistance()
```

Figure 14.8: beacon-chain/powchain/service.go#L679-L701.

```
func (s *Service) handleETH1FollowDistance() {
	defer safelyHandlePanic()
```

Figure 14.9: beacon-chain/powchain/service.go#L608-L609.

```
// safelyHandleHeader will recover and log any panic that occurs from the
// block
func safelyHandlePanic() {
        if r := recover(); r != nil {
                log.WithFields(logrus.Fields{
                        "r": r,
                }).Error("Panicked when handling data from ETH 1.0 Chain! Recovering...")

                debug.PrintStack()
        }
}
```

Figure 14.10: beacon-chain/powchain/service.go#L596-L606.

Exploit Scenario
The `powchain` service is unable to fetch data it requires. The error goes unnoticed because it is suppressed by `safelyHandlePanic`. The `powchain` service continues in an invalid state.

Recommendation
Short term, adjust the implementation of `goodFetcher.HeaderByNumber` so that it returns an error when `g.backend.Blockchain().CurrentHeader()` is `nil`. This will eliminate a `nil`-pointer dereference that currently exists in the `TestLatestMainchainInfo_OK` test.

Long term, regularly run your unit tests with all uses of `recover` disabled. Verify that all panics that would have been recovered are expected. These steps will help ensure that Prysm's uses of `recover`, generally, do not unintentionally suppress errors.

15. Premature loop exit causes validator deposits to be missed

Severity: Medium
Type: Data Validation
Target: `beacon-chain/core/blocks/deposit.go`

Difficulty: High
Finding ID: TOB-PRYSM-015

Description
The `ProcessPreGenesisDeposits` processes Eth 1.0 deposits in preparation for the Eth 2.0 mainnet launch. If a depositor's public key cannot be processed, then the function returns the result of processing all deposits up to that point. Thus, deposits after that point are lost.

The code in question appears in Figure 15.1. The function loops over a sequence of deposits. Work performed by the function is accumulated in a `beaconState` variable. If the function `ValidatorIndexByPubkey` returns an error (e.g., because the depositor's public key could not be found), then `beaconState` is returned as is. Deposits that would have been processed after that point are dropped.

```
func ProcessPreGenesisDeposits(
	ctx context.Context,
	beaconState *stateTrie.BeaconState,
	deposits []*ethpb.Deposit,
) (*stateTrie.BeaconState, error) {
	var err error
	beaconState, err = ProcessDeposits(ctx, beaconState, &ethpb.SignedBeaconBlock{
		Block: &ethpb.BeaconBlock{Body: &ethpb.BeaconBlockBody{Deposits: deposits}}})
	if err != nil {
		return nil, errors.Wrap(err, "could not process deposit")
	}
	for _, deposit := range deposits {
		pubkey := deposit.Data.PublicKey
		index, ok := beaconState.ValidatorIndexByPubkey(bytesutil.ToBytes48(pubkey))
		if !ok {
			return beaconState, nil
		}
		balance, err := beaconState.BalanceAtIndex(index)
		if err != nil {
			return nil, err
		}
		validator, err := beaconState.ValidatorAtIndex(index)
		if err != nil {
			return nil, err
		}
		validator.EffectiveBalance =
mathutil.Min(balance-balance%params.BeaconConfig().EffectiveBalanceIncrement,
params.BeaconConfig().MaxEffectiveBalance)
		if validator.EffectiveBalance ==
			params.BeaconConfig().MaxEffectiveBalance {
			validator.ActivationEligibilityEpoch = 0
			validator.ActivationEpoch = 0
		}
		if err := beaconState.UpdateValidatorAtIndex(index, validator); err != nil {
			return nil, err
		}
	}
```

```
        return beaconState, nil
}
```

Figure 15.1: beacon-chain/core/blocks/deposit.go#L22-L58.

Exploit Scenario
Alice submits an Eth 1.0 deposit in order to become a validator. Eve arranges for her bogus public key to be processed before Alice's deposit. Alice's deposit is ignored.

Recommendation
Short term, if a depositor's public key cannot be processed, skip over it and continue processing the remaining deposits. This will eliminate a bug that currently allows valid deposits to be ignored.

Long term, be wary of returning from inside of a loop on an error condition. Such a practice is a frequent source of errors.

16. Proposer indices cannot be retrieved from cache

Severity: Medium | Difficulty: Low
Type: Data Validation | Finding ID: TOB-PRYSM-016
Target: `beacon-chain/cache/committee.go`, `beacon-chain/cache/committee_test.go`

Description
`Committees` structs are cached using their `Seed` field as a key (Figure 16.1). Proposer indices are inserted into the cache in `Committees` structs with the `Seed` field unset. As such, they are unretrievable.

```
// Committees defines the shuffled committees seed.
type Committees struct {
	CommitteeCount  uint64
	Seed            [32]byte
	ShuffledIndices []uint64
	SortedIndices   []uint64
	ProposerIndices []uint64
}
```

Figure 16.1: beacon-chain/cache/committee.go#L35-L42.

Figure 16.2 shows what happens when a set of a proposer indices cannot be found in the cache. A new `Committees` struct is created to contain the proposer indices, but its `Seed` field is not set.

```
// AddProposerIndicesList updates the committee shuffled list with proposer indices.
func (c *CommitteeCache) AddProposerIndicesList(seed [32]byte, indices []uint64) error {
	c.lock.Lock()
	defer c.lock.Unlock()

	obj, exists, err := c.CommitteeCache.GetByKey(key(seed))
	if err != nil {
		return err
	}
	if !exists {
		committees := &Committees{ProposerIndices: indices}
		if err := c.CommitteeCache.Add(committees); err != nil {
			return err
		}
```

Figure 16.2: beacon-chain/cache/committee.go#L118-L131.

There is a unit test for `AddProposerIndicesList`, but it contains an error. The `indices` local variable is overwritten with `nil` while verifying that those indices are not already in the cache. Consequently, `nil` is inserted into the cache. Later, `nil` is retrieved from the cache (as a result of this bug) and stored in the `received` local variable. The local variables `received` and `indices` are compared and they match (they are both `nil`), and the test passes.

```
func TestCommitteeCache_AddProposerIndicesList(t *testing.T) {
```

```
	cache := NewCommitteesCache()

	seed := [32]byte{'A'}
	indices := []uint64{1, 2, 3, 4, 5}
	indices, err := cache.ProposerIndices(seed)
	require.NoError(t, err)
	if indices != nil {
		t.Error("Expected committee count not to exist in empty cache")
	}
	require.NoError(t, cache.AddProposerIndicesList(seed, indices))

	received, err := cache.ProposerIndices(seed)
	require.NoError(t, err)
	assert.DeepEqual(t, received, indices)
```

Figure 16.3: beacon-chain/cache/committee_test.go#L90-L104.

Exploit Scenario
Alice runs a Prysm node. Alice's node wastes work maintaining a cache unnecessarily. This combined with other factors makes her node unable to keep up with the network. Alice's node is slashed as a result.

Recommendation
Short term, set a `Committees` struct's `Seed` when inserting proposer indices into the cache. This will eliminate a bug that currently makes the struct unretrievable.

Long term, regularly run Go's shadow tool over your codebase. This bug is due, in part, to variable shadowing in a unit test. Regularly running Go's shadow can help to reveal such problems.

17. Shuffled list is not added to cache if proposer indices are present

Severity: Informational | Difficulty: Undetermined
Type: Data Validation | Finding ID: TOB-PRYSM-017
Target: `beacon-chain/cache/committee.go`, `beacon-chain/core/helpers/committee.go`

Description
Attempting to insert a shuffled list into the cache after a set of proposer indices with the same seed fails. This wastes work and is inconsistent with how proposer indices are inserted into the cache.

The relevant code appears in Figure 17.1 and 17.2. `UpdateCommitteeCache` is called to add a shuffled list to the cache. It builds a `Committees` struct and calls `AddCommitteeShuffledList` to add it to the cache. `AddCommitteeShuffledList` does so using `AddIfNotPresent`. Thus, if a `Committees` struct was already present with the same seed, say because proposer indices with that seed were already inserted, then the shuffled list would not be inserted.

```
// UpdateCommitteeCache gets called at the beginning of every epoch to cache the committee
shuffled indices
// list with committee index and epoch number. It caches the shuffled indices for current
epoch and next epoch.
func UpdateCommitteeCache(state *stateTrie.BeaconState, epoch uint64) error {
	for _, e := range []uint64{epoch, epoch + 1} {
		...
		if err := committeeCache.AddCommitteeShuffledList(&cache.Committees{
			ShuffledIndices: shuffledIndices,
			CommitteeCount:  count * params.BeaconConfig().SlotsPerEpoch,
			Seed:            seed,
			SortedIndices:   sortedIndices,
		}); err != nil {
			return err
		}
```

Figure 17.1: beacon-chain/core/helpers/committee.go#L307-L342.

```
// AddCommitteeShuffledList adds Committee shuffled list object to the cache. T
// his method also trims the least recently list if the cache size has ready the max cache
size limit.
func (c *CommitteeCache) AddCommitteeShuffledList(committees *Committees) error {
	c.lock.Lock()
	defer c.lock.Unlock()

	if err := c.CommitteeCache.AddIfNotPresent(committees); err != nil {
		return err
	}
	trim(c.CommitteeCache, maxCommitteesCacheSize)
	return nil
}
```

Figure 17.2: beacon-chain/cache/committee.go#L105-L116.

This behavior is inconsistent with how proposer indices are inserted into the cache (Figures 17.3 and 17.4). `UpdateProposerIndicesInCache` is called to insert proposer indices into the cache. `UpdateProposerIndicesInCache`, in turn, calls `AddProposerIndicesList`. If a `Committees` struct with the relevant seed already exists in the cache, then the proposer indices are added to that struct.

```
// UpdateProposerIndicesInCache updates proposer indices entry of the committee cache.
func UpdateProposerIndicesInCache(state *stateTrie.BeaconState, epoch uint64) error {
	...
	if err := committeeCache.AddProposerIndicesList(seed, proposerIndices); err != nil {
		return err
	}
```

Figure 17.3: beacon-chain/core/helpers/committee.go#L348-L365.

```
// AddProposerIndicesList updates the committee shuffled list with proposer indices.
func (c *CommitteeCache) AddProposerIndicesList(seed [32]byte, indices []uint64) error {
	c.lock.Lock()
	defer c.lock.Unlock()

	obj, exists, err := c.CommitteeCache.GetByKey(key(seed))
	if err != nil {
		return err
	}
	if !exists {
		committees := &Committees{ProposerIndices: indices}
		if err := c.CommitteeCache.Add(committees); err != nil {
			return err
		}
	} else {
		committees, ok := obj.(*Committees)
		if !ok {
			return ErrNotCommittee
		}
		committees.ProposerIndices = indices
		if err := c.CommitteeCache.Add(committees); err != nil {
			return err
		}
	}
```

Figure 17.4: beacon-chain/cache/committee.go#L118-L141.

Currently, this finding does not pose a problem, because proposer indices are inserted into the cache without an associated seed (TOB-PRYSM-016). However, if that bug is fixed in the obvious way, then this bug will become an issue. This can be verified with a minor modification of the test in Figure 16.3.

Exploit Scenario

Alice runs a Prysm node with a patch for TOB-PRYSM-016. Alice's node wastes work computing shuffled lists that it does not insert into the cache. This combined with other factors makes her node unable to keep up with the network. Alice's node is slashed as a result.

Recommendation

Short term, when inserting a shuffled list into the cache, add it to an existing `Committees` struct if one already exists with the relevant seed. This will eliminate a bug that currently causes a node to build `Committees` structs unncessarily.

Long term, consider ways that the cache code may be consolidated. This bug is, in part, the result of a discrepancy in how the cache handled proposer indices vs. shuffled indices. Consolidating that code might have prevented this bug. Furthermore, you may want to include other types of data in the cache in the future. Consolidating the cache code could help to facilitate such extensions.

18. Inefficiencies surrounding use of cache

Severity: Informational
Type: Data Validation
Target: `beacon-chain/core/helpers/validators.go`

Difficulty: Not applicable
Finding ID: TOB-PRYSM-018

Description
Work is unnecessarily duplicated when inserting items into the cache.

Consider how proposer indices are inserted into the cache (Figures 18.1 through 18.3). The active valid indices are computed for the given epoch. Then, the function `UpdateProposerIndicesInCache` is called *without the results of the just mentioned computation*. `UpdateProposerIndicesInCache` recomputes the `ActiveValidatorIndices` and calls `precomputeProposerIndices`, which computes the proposer indices for all slots within the epoch. Eventually, control returns to `BeaconProposerIndex`, which *recomputes the proposer index* for the slot of interest. Thus, work done to compute the active validator indices and the proposer index for one slot is duplicated.

```
func BeaconProposerIndex(state *stateTrie.BeaconState) (uint64, error) {
	...
	proposerIndices, err := committeeCache.ProposerIndices(seed)
	if err != nil {
		return 0, errors.Wrap(err, "could not interface with committee cache")
	}
	if proposerIndices != nil {
		return proposerIndices[state.Slot()%params.BeaconConfig().SlotsPerEpoch], nil
	}
	...
	indices, err := ActiveValidatorIndices(state, e)
	if err != nil {
		return 0, errors.Wrap(err, "could not get active indices")
	}

	if err := UpdateProposerIndicesInCache(state, e); err != nil {
		return 0, errors.Wrap(err, "could not update committee cache")
	}

	return ComputeProposerIndex(state, indices, seedWithSlotHash)
}
```

Figure 18.1: beacon-chain/core/helpers/validators.go#L176-L209.

```
func UpdateProposerIndicesInCache(state *stateTrie.BeaconState, epoch uint64) error {
	indices, err := ActiveValidatorIndices(state, epoch)
	if err != nil {
		return nil
	}
	proposerIndices, err := precomputeProposerIndices(state, indices)
	if err != nil {
		return err
	}
```

Figure 18.2: beacon-chain/core/helpers/committee.go#L349-L357.

```
func precomputeProposerIndices(state *stateTrie.BeaconState, activeIndices []uint64)
([]uint64, error) {
        ...
        for i := uint64(0); i < params.BeaconConfig().SlotsPerEpoch; i++ {
                seedWithSlot := append(seed[:], bytesutil.Bytes8(slot+i)...)
                seedWithSlotHash := hashFunc(seedWithSlot)
                index, err := ComputeProposerIndex(state, activeIndices, seedWithSlotHash)
                if err != nil {
                        return nil, err
                }
                proposerIndices[i] = index
        }
```

Figure 18.3: beacon-chain/core/helpers/committee.go#L377-L398.

A similar comment applies to `ActiveValidatorIndices` and `UpdateCommitteeCache`.

Recommendation

Short term, refactor `BeaconProposerIndex`, `UpdateProposerIndicesInCache`, and `precomputeProposerIndices` so that the work done to compute the active validator indices and the proposer index for one slot is not duplicated. Be careful to avoid introducing data races involving the presence of data in the cache in the process. These steps will improve the overall efficiency of the cache.

Long term, as new types of data are added to the cache, avoid duplicating computations across functions that update the cache and their callers. This will help maintain the cache's effectiveness.

19. Wrong domain used in `helpers.SlotSignature`

Severity: Medium
Difficulty: Low
Type: Cryptography
Finding ID: TOB-PRYSM-019
Target: `beacon-chain/core/helpers/attestation.go`

Description
The current Eth 2.0 specification says that selection proofs should be signed using `DOMAIN_SELECTION_PROOF`. Prysm deviates from the specification by using `DOMAIN_BEACON_ATTESTER` (Figure 19.1).

```
// SlotSignature returns the signed signature of the hash tree root of input slot.
//
// Spec pseudocode definition:
//   def get_slot_signature(state: BeaconState, slot: Slot, privkey: int) -> BLSSignature:
//    domain = get_domain(state, DOMAIN_SELECTION_PROOF, compute_epoch_at_slot(slot))
//    signing_root = compute_signing_root(slot, domain)
//    return bls.Sign(privkey, signing_root)
func SlotSignature(state *stateTrie.BeaconState, slot uint64, privKey bls.SecretKey)
(bls.Signature, error) {
       d, err := Domain(state.Fork(), CurrentEpoch(state),
params.BeaconConfig().DomainBeaconAttester, state.GenesisValidatorRoot())
```

Figure 19.1: beacon-chain/core/helpers/attestation.go#L16-L24.

This discrepancy appears to be the result of an incomplete fix in pull request 5604.

Exploit Scenario
Alice runs a Prysm node. Her node is selected for aggregation. Her node produces a selection proof with an invalid signature and is slashed.

Recommendation
Short term, adjust SlotSignature so that it uses `DOMAIN_SELECTION_PROOF` for selection proofs, rather than `DOMAIN_BEACON_ATTESTER`. This will bring the code in-line with the Eth 2.0 current specification.

Long term, when reviewing PRs, consider not only the code in the PR, but other code to which similar changes should be applied. This issue appears to be the result of an incomplete fix. A more expansive approach to PR review could help to prevent similar situations.

20. Wrong loop exit condition in `store.ForkChoice.AncestorRoot`

Severity: Medium | Difficulty: Low
Type: Denial of Service | Finding ID: TOB-PRYSM-020
Target: `beacon-chain/forkchoice/protoarray/store.go`

Description
The `AncestorRoot` function iterates through nodes until it finds one with a slot less than a slot of interest. The loop does not account for the possibility that it could run out of nodes to check.

The definition of `AncestorRoot` appears in Figure 20.1. Nearby code suggests that `f.store.nodes[i].parent` could take the value `NonExistentNode`, which is defined as `^uint64(0)`. If the loop variable `i` were to take this value, an "index out of range" error would result. Thus, the the loop condition should be something like:

```
i < uint64(len(f.store.nodes)) && f.store.nodes[i].slot > slot
```

```
// AncestorRoot returns the ancestor root of input block root at a given slot.
func (f *ForkChoice) AncestorRoot(ctx context.Context, root [32]byte, slot uint64) ([]byte,
error) {
        i, ok := f.store.nodesIndices[root]
        if !ok {
                return nil, errors.New("node does not exist")
        }
        if i >= uint64(len(f.store.nodes)) {
                return nil, errors.New("node index out of range")
        }

        for f.store.nodes[i].slot > slot {
                if ctx.Err() != nil {
                        return nil, ctx.Err()
                }

                i = f.store.nodes[i].parent
        }
        if i >= uint64(len(f.store.nodes)) {
                return nil, errors.New("node index out of range")
        }

        return f.store.nodes[i].root[:], nil
}
```

Figure 20.1: beacon-chain/forkchoice/protoarray/store.go#L156-L178.

Exploit Scenario
Eve finds a way to call `AncestorRoot` with an exceedingly small slot value, thereby causing an "index out of range" error. Eve uses the bug to crash Prysm nodes.

Recommendation

Short term, adjust the `AncestorRoot`'s for loop condition so that it accounts for `i` not within the range of `f.store.nodes's indices`. This will eliminate potential "index out of range" errors that could be used for denial of service.

Long term, consider developing a fuzzer for `AncestorRoot` or one of its predecessors in the call graph. Fuzzing might have caught this bug, and could catch similar bugs in the future.

21. Handler `on_tick` unimplemented

Severity: Undetermined
Type: Undefined Behavior
Target: beacon node

Difficulty: Undetermined
Finding ID: TOB-PRYSM-021

Description
The Eth 2.0 specification dictates that `on_time` (Figure 21.1) be called "whenever `time > store.time` where `time` is the current Unix time." `on_tick` does not appear to be implemented in Prysm.

```
def on_tick(store: Store, time: uint64) -> None:
    previous_slot = get_current_slot(store)

    # update store time
    store.time = time

    current_slot = get_current_slot(store)
    # Not a new epoch, return
    if not (current_slot > previous_slot and compute_slots_since_epoch_start(current_slot)
== 0):
        return
    # Update store.justified_checkpoint if a better checkpoint is known
    if store.best_justified_checkpoint.epoch > store.justified_checkpoint.epoch:
        store.justified_checkpoint = store.best_justified_checkpoint
```

Figure 21.1: specs/phase0/fork-choice.md#on_tick.

Note from Figure 21.1 that `on_tick` updates `store.justified_checkpoint` to `store.best_justified_checkpoint` under certain conditions. There is only one function in Prysm that performs such an update: `updateHead`. `updateHead` is called in `ReceiveAttestationNoPubsub` in `beacon-chain/blockchain/receive_attestation.go` and in `ReceiveBlock` in `beacon-chain/blockchain/receive_block.go`, but nowhere else.

```
// Determined the head from the fork choice service and saves its new data
// (head root, head block, and head state) to the local service cache.
func (s *Service) updateHead(ctx context.Context, balances []uint64) error {
        ctx, span := trace.StartSpan(ctx, "blockChain.updateHead")
        defer span.End()

        // To get the proper head update, a node first checks its best justified
        // can become justified. This is designed to prevent bounce attack and
        // ensure head gets its best justified info.
        if s.bestJustifiedCheckpt.Epoch > s.justifiedCheckpt.Epoch {
                s.justifiedCheckpt = s.bestJustifiedCheckpt
                if err := s.cacheJustifiedStateBalances(ctx,
bytesutil.ToBytes32(s.justifiedCheckpt.Root)); err != nil {
                        return err
                }
        }
```

Figure 11.2: beacon-chain/blockchain/head.go#L30-L44.

Exploit Scenario
Alice runs a Prysm node. Because her node does not implement `on_tick`, it falls out of sync with other nodes on the network. Alice suffers financial loss as a result.

Recommendation
Short term, implement the `on_tick` method from the Eth 2.0 specification. This will eliminate a discrepancy that currently exists between the Eth 2.0 specification and Prysm's implementation of it.

Long term, maintain a mapping of Eth 2.0 function to their corresponding implementations in Prysm. This will make it easier for developers to understand how Prysm implements the specification. It will also help to identify gaps in the implementation.

22. Hardcoded contract bytecode

Severity: Medium | Difficulty: Low
Type: Data Validation | Finding ID: TOB-PRYSM-022
Target: `contracts/deposit-contract/depositContract.go`

Description
The compiled bytecode is included in `depositContract.go` as a variable instead of read in from a machine-only writable file such as `bytecode.bin`.

```
// DepositContractBin is the compiled bytecode used for deploying new contracts.
var DepositContractBin =
"0x740100000000000000000000000000000000000000000000006020526f7fffffffffffffffffffffffffffff6040527ffffff..."

// DeployDepositContract deploys a new Ethereum contract, binding an instance of
DepositContract to it.
func DeployDepositContract(auth *bind.TransactOpts, backend bind.ContractBackend,
_drain_address common.Address) (common.Address, *types.Transaction, *DepositContract, error)
{
        parsed, err := abi.JSON(strings.NewReader(DepositContractABI))
        if err != nil {
                return common.Address{}, nil, nil, err
        }

        address, tx, contract, err := bind.DeployContract(auth, parsed,
common.FromHex(DepositContractBin), backend, _drain_address)
```

Figure 22.1 Use of the `DepositContractBin` variable (depositContract.go#L32-L42)

```
0x740100000000000000000000000000000000000000000000006020526f7fffffffffffffffffffffffffffff...
```

Figure 22.2 bytecode.bin that should be used instead of `DepositContractBin`

Exploit Scenario
Alice is a Prysm developer that accidentally commits a change to the bytecode variable that is used in the deposit contract. Invalid bytecode is used for a deposit which causes the deposits to fail.

Recommendation
Short term, read in the bytecode for deposits only from the bytecode.bin file that currently stores it. Update pipelines so that only automation changes this value and it is never expected to be edited directly by developers. We also recommend putting in a git pre-commit hook to prevent developers from committing to this file.

Long term, remove any manual processes required for releasing code and automate for consistency and reliability purposes.

23. Files and directories created with os.MkdirAll and ioutil.WriteFile are not ensured to have correct permissions

Severity: Low | Difficulty: High
Type: Access Controls | Finding ID: TOB-PRYSM-023
Target: `tools/genesis-state-gen/main.go`, `tools/enr-calculator/main.go`, `tools/interop/export-genesis/main.go`, `beacon-chain/core/state/interop/write_block_to_disk.go`, `beacon-chain/core/state/interop/write_state_to_disk.go`, `validator/accounts/v2/wallet.go`, `beacon-chain/db/kv/kv.go`, `validator/db/kv/db.go`

Description
The `os.MkdirAll` function does not enforce permissions on directories already present on the filesystem. Therefore, if an attacker were to pre-populate a directory with overly-permissive permissions, those permissions will not be altered.

Similarly, the `ioutil.WriteFile` and `os.OpenFile` functions have the same attributes, but applied to files. If an attacker pre-populates a file with overly-permissive permissions, the permissions will not be changed.

This could lead to an attacker pre-populating a file or directory with overly-permissive permissions, allowing modifications or reading after the program has started.

Exploit Scenario
Eve has local access to a machine where generation of the genesis state occurs. Eve pre-populates directories with overly permissive permissions where `prysm` will write potentially sensitive values, or perform sensitive operations.

Recommendation
Short term, when using `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on them. This will help avoid situations where sensitive information is written to a pre-existing attacker-controlled path.

Long term, enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the applications as a whole.

24. Response body not closed in tests

Severity: Low
Difficulty: Medium
Type: Denial of Service
Finding ID: TOB-PRYSM-024
Target: `shared/prometheus/service_test.go`, `shared/prometheus/logrus_collector_test.go`, `endtoend/helpers/helpers.go`

Description
Multiple tests within Prysm neglect necessary Go hygiene practices such as closing the response body after an http request. If tests are run on a machine frequently enough, this memory leak could have unintended consequences.

```
func writeURLRespAtPath(url string, filePath string) error {
	resp, err := http.Get(url)
	if err != nil {
		return err
	}
	body, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		return err
	}
	file, err := os.Create(filePath)
	if err != nil {
		return err
	}
	if _, err := file.Write(body); err != nil {
		return err
	}
	return nil
}
```

Figure 24.1 Failure to call `resp.Body.Close()` in a heavily used helper function (endtoend/helpers/helpers.go#L165-L182).

Exploit Scenario
A Prysm developer is regularly running the tests locally on the same machine they are using to engage with the beacon-chain and make deposits. Rapidly running the tests continually outpaces the garbage collection and an out of memory (OOM) error causes their node to crash.

Recommendation
Short term, validate that all file handlers are closed to avoid putting undue stress on the garbage collector.

Long term, standardize idiomatic Go practices such as proper handling of response bodies.

25. Password guesses should be limited to prevent brute force

Severity: Medium
Difficulty: Low
Type: Access Controls
Finding ID: TOB-PRYSM-025
Target: `validator/accounts/v2/accounts_deposit.go`, `validator/keymanager/v2/direct/direct.go`

Description
Throughout the codebase, passwords are minimally validated for baseline requirements and are not checked for number of attempts. In a situation like an account deposit, this makes it vulnerable to a brute force attack trying all possible passwords for another user's account.

```
func PasswordPrompt(promptText string, validateFunc func(string) error) (string, error) {
	var responseValid bool
	var response string
	for !responseValid {
		fmt.Printf("%s: ", au.Bold(promptText))
		bytePassword, err := terminal.ReadPassword(int(os.Stdin.Fd()))
		if err != nil {
			return "", err
		}
		response = strings.TrimRight(string(bytePassword), "\r\n")
		if err := validateFunc(response); err != nil {
			fmt.Printf("\nEntry not valid: %s\n", au.BrightRed(err))
		} else {
			fmt.Println("")
			responseValid = true
		}
	}
	return response, nil
}
```

Figure 25.1 PasswordPrompt doesn't track number of attempts, nor do any of its references track attempts (shared/promptutil/prompt.go#L99-L117).

Exploit Scenario
Eve has a validator's wallet address. She attempts to direct transfer funds into her wallet with their password. She is able to iterate through many guesses for passwords and steals funds.

Recommendation
Short term, limit the number of password guesses a user can attempt by implementing an exponential backoff strategy on password acceptance timing.

Long term, implement monitoring surrounding excessively high password guess frequency against an account and consider locking it until the user can validate their identity.

26. Premature exit under certain conditions in `blockchain.Service.ancestor`

Severity: Medium | Difficulty: Undetermined
Type: Undefined Behavior | Finding ID: TOB-PRYSM-026
Target: `beacon-chain/blockchain/process_block_helpers.go`

Description
The function `blockchain.Service.ancestor` is supposed to find an ancestral root from a given root and slot. The function can incorrectly return an error when the root is not in the service's `forkChoiceStore`.

The code in question appears in Figure 26.1. If `s.forkChoiceStore.HasParent(r)` is true, then `s.forkChoiceStore.AncestorRoot` is called to find the root in `forkChoiceStore`. `AncestorRoot` can fail with a `"node index out of range"` error (recall Figure 20.1). In such a case, the service's `beaconDB` should be queried, as the root could lie there. Currently, `beaconDB` is not queried.

```
// ancestor returns the block root of an ancestry block from the input block root.
//
// Spec pseudocode definition:
//   def get_ancestor(store: Store, root: Root, slot: Slot) -> Root:
//    block = store.blocks[root]
//    if block.slot > slot:
//        return get_ancestor(store, block.parent_root, slot)
//    elif block.slot == slot:
//        return root
//    else:
//        # root is older than queried slot, thus a skip slot. Return most recent root prior
to slot
//        return root
func (s *Service) ancestor(ctx context.Context, root []byte, slot uint64) ([]byte, error) {
        ...
        // Get ancestor root from fork choice store instead of recursively looking up blocks
in DB.
        // This is most optimal outcome.
        if s.forkChoiceStore.HasParent(r) {
                return s.forkChoiceStore.AncestorRoot(ctx, r, slot)
        }

        signed, err := s.beaconDB.Block(ctx, r)
        if err != nil {
                return nil, errors.Wrap(err, "could not get ancestor block")
        }
```

Figure 26.1: beacon-chain/blockchain/process_block_helpers.go#L245-L276.

Exploit Scenario
The above bug in `ancestor` makes it impossible for Prysm nodes to reach consensus with other Eth 2.0 nodes. A fork in the network results.

Recommendation

Short term, adjust `ancestor` so that if `s.forkChoiceStore.AncestorRoot` returns a `"node index out of range"` error, `beaconDB` is queried. This will eliminate a bug that can cause `ancestor` to incorrectly return an error.

Long term, require additional review of functions whose implementations do not obviously match the specification. For example, as can be seen from Figure 26.1, Prysm's `ancestor` does not directly correspond to the specification's `get_ancestor`, though they are meant to achieve the same functionality. Functions such as `ancestor` warrant additional scrutiny.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Documentation	Related to documenting or recording use scenarios
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for

	client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **beacon-chain/gateway/server/main.go#L44 and many other locations:** Use net.JoinHostPort instead of fmt.Sprintf for IPv6 and future compatibility.

- **The Bazel resource file (.bazelrc) insists on an outdated version of Bazel (3.2.0).** Since September 2, 2020, version 3.5.0 is available. Consider upgrading to the newer version.

- **epoch/precompute/slashing_test.go#L99 and several other locations: An improper conversion from an int to a string is performed:**

  ```
  t.Run(string(i), func(t *testing.T) {
  ```

  Use `fmt.Sprint(...)` instead.

- **Verify that your implementations satisfy their interfaces using global declarations of the form:**

  ```
  var _ Interface = (*Implementation)(nil)
  ```

  Such declarations have the additional benefit of alerting readers to which implementations satisfy which interfaces. For instance, one could add the following line to `beacon-chain/sync/initial-sync/service.go`:

  ```
  var _ sync.Checker = (*Service)(nil)
  ```

- **validator/node/node.go#L55-L64, validator/node/node.go#L338-L349, other locations:** Overusing `contexts` to store values that aren't relevant to their use in concurrency and cancelling patterns breaks the Single Responsibility Principle and muddies the code waters. Instead of storing values for setup on the context getting passed in, consider the WithOption pattern that is idiomatic for Go.

- **shared/sliceutil/slice.go#L51, shared/sliceutil/slice.go#L166, shared/sliceutil/slice.go#L272:** A condition of the form `(X-Y) == 0` appears. Such a condition could be written more simply as `X == Y`.

- **fuzz/BUILD.bazel#L110-L116:** The `block_fuzz_test` incorrectly uses Sigma Prime's `block_header` corpus. The test should use Sigma Prime's `block` corpus.

  ```
  go_fuzz_test(
      name = "block_fuzz_test",
      srcs = [
          "block_fuzz.go",
  ```

```
            ] + COMMON_SRCS,
            corpus = "@sigp_beacon_fuzz_corpora//:current_mainnet_block_header",
            corpus_path =
"external/sigp_beacon_fuzz_corpora/0-11-0/mainnet/block_header",
```

- **beacon-chain/core/helpers/slot_epoch.go#L77-L83, slasher/detection/testing/utils.go#L51-L53:** The function `StartSlot` is implemented in two places, once with overflow checking and once without. Consider consolidating the implementations.

- **beacon-chain/core/helpers/validators.go#L284:** Consider moving `ComputeProposerIndexWithValidators` to `beacon-chain/core/helpers/validators_test.go`. The function is marked as deprecated, and `beacon-chain/core/helpers/validators_test.go` is the only place where it is used. Consider also adding tests to verify that `ComputeProposerIndex` and `ComputeProposerIndexWithValidators` produce the same results.

- **shared/attestationutil/attestation_utils.go#L157-L171:** The function `attestationutil.IsValidAttestationIndices` implements a complicated algorithm for checking whether an array of indices is sorted. It should be sufficient to check that each index in the array is less than the next.

- **beacon-chain/p2p/handshake.go#L29-L143**: Convoluted nesting of function definitions makes code unreadable and unnecessarily complex.

D. Ineffectual Assignments

```
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/blockchain/head.go#L214
ineffectual assignment to set |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/blockchain/process_block.go#L238
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L82
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L104
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L116
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L147
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L157
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L164
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L176
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L193
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L209
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/deposits_cache.go#L228
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/pending_deposits.go#L33
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/pending_deposits.go#L70
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/pending_deposits.go#L94
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/cache/depositcache/pending_deposits.go#L132
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/core/epoch/precompute/new.go#L20
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kafka/export_wrapper.go#L50
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
```

```
chain/db/kv/archived_point.go#L13
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/archived_point.go#L28
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/archived_point.go#L46
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/archived_point.go#L63
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L111
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L230
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L265
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L385
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L452
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/blocks.go#L480
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/deposit_contract.go#L15
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/deposit_contract.go#L30
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/encoding.go#L17
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/encoding.go#L31
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/finalized_block_roots.go#L163
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/operations.go#L59
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/operations.go#L72
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/powchain.go#L14
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/powchain.go#L29
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/regen_historical_states.go#L198
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
```

```
chain/db/kv/slashings.go#L59
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/slashings.go#L72
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/slashings.go#L128
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/slashings.go#L141
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/state.go#L229
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/state.go#L335
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/state_summary.go#L69
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/utils.go#L19
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/utils.go#L38
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/db/kv/utils.go#L64
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/helpers.go#L19
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/nodes.go#L17
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/nodes.go#L65
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/nodes.go#L117
ineffectual assignment to newParentChild |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/nodes.go#L212
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/nodes.go#L280
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/forkchoice/protoarray/store.go#L65
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/operations/slashings/service.go#L33
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/operations/slashings/service.go#L75
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/operations/slashings/service.go#L175
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
```

```
chain/operations/voluntaryexits/service.go#L59
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/p2p/encoder/ssz.go#L90
ineffectual assignment to currentSlot |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/p2p/fork.go#L91
ineffectual assignment to requestedEpoch |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L38
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L200
ineffectual assignment to activatedIndices |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L399
ineffectual assignment to exitedIndices |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L400
ineffectual assignment to slashedIndices |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L401
ineffectual assignment to ejectedIndices |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L402
ineffectual assignment to activationQueueChurn |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L593
ineffectual assignment to exitQueueChurn |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L598
ineffectual assignment to b |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/beacon/validators.go#L807
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/node/server.go#L105
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/rpc/validator/proposer.go#L692
ineffectual assignment to fTrie |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/state/state_trie.go#L439
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/state/stategen/getter.go#L209
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/state/stategen/setter.go#L61
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/sync/initial-sync/blocks_fetcher_peers.go#L127
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/sync/rpc_goodbye.go#L39
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
chain/sync/rpc_metadata.go#L21
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/beacon-
```

```
chain/sync/validate_aggregate_proof.go#L235
ineffectual assignment to offset |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/fuzz/ge
nerated.ssz.go#L27
ineffectual assignment to offset |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/fuzz/ge
nerated.ssz.go#L121
ineffectual assignment to offset |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/fuzz/ge
nerated.ssz.go#L215
ineffectual assignment to offset |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/proto/b
eacon/p2p/v1/generated.ssz.go#L866
ineffectual assignment to offset |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/proto/b
eacon/p2p/v1/generated.ssz.go#L1462
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/shared/
depositutil/deposit.go#L138
ineffectual assignment to neighbor |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/shared/
trieutil/sparse_merkle.go#L105
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/block_header.go#L19
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/block_header.go#L52
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/block_header.go#L103
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/block_header.go#L117
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/chain_data.go#L15
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/chain_data.go#L34
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/kv.go#L43
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/proposer_slashings.go#L17
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/proposer_slashings.go#L63
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/proposer_slashings.go#L82
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/proposer_slashings.go#L106
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/proposer_slashings.go#L126
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
```

```
/db/kv/spanner.go#L81
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner.go#L163
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner.go#L202
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner.go#L261
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner.go#L316
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner.go#L333
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/spanner_new.go#L100
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/validator_id_pubkey.go#L15
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/validator_id_pubkey.go#L28
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/db/kv/validator_id_pubkey.go#L43
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/detection/detect.go#L129
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/detection/service.go#L221
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/detection/service.go#L229
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/slasher
/rpc/server.go#L151
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/tools/k
eystores/main.go#L101
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/tools/k
eystores/main.go#L138
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/tools/p
cli/main.go#L179
ineffectual assignment to keystoresToBackup |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/accounts/v2/accounts_backup.go#L106
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/accounts/v2/wallet.go#L113
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/db/kv/manage.go#L34
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
```

```
or/db/kv/manage.go#L47
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/db/kv/proposal_history.go#L19
ineffectual assignment to ctx |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/db/kv/proposal_history.go#L44
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/keymanager/v2/derived/derived.go#L293
ineffectual assignment to err |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/keymanager/v2/direct/direct.go#L244
ineffectual assignment to password |
https://github.com/prysmaticlabs/prysm/blob/245c18784eda370ea3218e8704651edad763978d/validat
or/keymanager/v2/direct/direct.go#L372
```