SLOWMIST

Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.11.01, the SlowMist security team received Ethereum community's security audit application for Prysm, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "black, grey box lead, white box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for the chain includes two steps:

Chain codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The codes are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the chain:

NO.	Audit Items	Result
1	SAST	Some Risks
2	[Encryption]Insecure Encryption Library Audit	Some Risks
3	[Encryption]Hash strength security	Passed
4	[Encryption]Length Extension Attack	Passed
5	[Encryption]Transaction Malleability Attack	Passed
6	[Ledger]Transaction Replay Attack	Passed
7	[Off-Chain]False Top-up Audit	Passed
8	[RPC]The Ethereum Black Valentine's Day Vulnerability	Passed
9	[P2P]Sybil Attack	Passed
10	[P2P]Eclipse Attack	Passed
11	[P2P]Eavesdropping Attack	Passed
12	[P2P]Denial of Service Attack	Passed
13	[P2P]BGP Hijack Attack	Passed
14	[P2P]Alien Attack	Some Risks
15	[RPC]Http Input Attack	Passed
16	[RPC Layer]Cross-Domain attack	Passed

NO.	Audit Items	Result
17	[Off-Chain]Transaction time-locked attack	Passed
18	Rug Pull Attack	Passed
19	[Ledger]Timejacking	Passed
20	[RPC]Denial of Service Attack	Passed
21	[RPC]Eavesdropping Attack	Passed

3 Project Overview

3.1 Project Introduction

Go implementation of Ethereum proof of stake.

3.2 Coverage

Target Code and Revision:

https://github.com/prysmaticlabs/prysm

Version: v3.1.2

SAST

Gosec scan:

Summary:

Issues : 0

Govulncheck scan:

Found 11 known vulnerabilities.

[Encryption]Insecure Encryption Library Audit

Signature:

ecdsa (crypto/ecdsa)

BLS12-381 (github.com/supranational/blst)

Hash:

highwayhash (github.com/minio/highwayhash)

sha256 (github.com/minio/sha256-simd)

sha3 (golang.org/x/crypto/sha3)

Serialize:

fastssz (github.com/prysmaticlabs/fastssz)

[Encryption]Hash strength security

In systems with high security requirements, SHA series algorithms (such as SHA-224, SHA256, SHA-384 and SHA-512) with hash values >= 224 bits should be used to ensure the integrity of sensitive data.

Weak hash functions such as md5 and sha1 are not used.

[Encryption]Length Extension Attack

Algorithms like MD5, SHA-1, and SHA-2 that are based on the Merkle–Damgard construction are susceptible to length extension attack. The SHA-3 algorithm is not susceptible.

[Encryption]Transaction Malleability Attack

The `secp256k1` / `blst` algorithm have malleability vulnerabilities, Prysm does not use them for transaction signatures.

Reference: https://gist.github.com/wadeAlexC/2490d522e81a796af9efcad1686e6754

`SSZ` can be ambiguous in Go when dealing with zero values, however, we did not find an exploitable ssz vulnerability in prysm.

[Ledger]Transaction Replay Attack

The beacon chain does not support normal transfers for now, we mainly audit the security of ETH asset flow for the creation and exit of validators.

When a user needs to create a validator, he needs to transfer 32ETH to the specified contract on the execution layer, and then the beacon chain will detect the contract recharge event

`DepositEvent(bytes,bytes,bytes,bytes,bytes)` and process the recharge.

The main logic is in the following locations.

- beacon-chain/execution/log_processing.go
- beacon-chain/execution/deposit.go
- beacon-chain/core/blocks/deposit.go
- contracts/deposit/deposit.go

All validators iterate through all blocks of ETH1 looking for `DepositEvent` events, so if a few nodes replay the recharge event it will not succeed.

```
// ProcessLog is the main method which handles the processing of all
// logs from the deposit contract on the eth1 chain.
func (s *Service) ProcessLog(ctx context.Context, depositLog gethtypes.Log) error {
	s.processingLock.RLock()
	defer s.processingLock.RUnlock()
	// Process logs according to their event signature.
	if depositLog.Topics[0] == depositEventSignature {
		if err := s.ProcessDepositLog(ctx, depositLog); err != nil {
			return errors.Wrap(err, "Could not process deposit log")
		}
		if s.lastReceivedMerkleIndex%eth1DataSavingInterval == 0 {
			return s.savePowchainData(ctx)
		}
		return nil
	}
	log.WithField("signature", fmt.Sprintf("%#x",
depositLog.Topics[0])).Debug("Not a valid event signature")
	return nil
}
```

[Off-Chain]False Top-up Audit

In some systems with weaknesses can be fake recharge by forging events.

No ETH false top-up vulnerability from ETH1 to beacon was found.

[RPC]The Ethereum Black Valentine's Day Vulnerability

When we start a node with default config:

```
$ bazel run //beacon-chain --config=release -- --execution-
endpoint=http://localhost:8545
//...
INFO gateway: Starting gRPC gateway address=127.0.0.1:3500
```

RPC is enabled by default, but the RPC port is only open locally and will not be operated remotely.

[P2P]Sybil Attack

A sybil attack on a node usually involves deploying a large number of sybil nodes through a small number of hosts

Prysm increases the cost of nodes by limiting the number of connections to a single IP.

- beacon-chain/p2p/connection_gater.go

```
const (
        // Limit for rate limiter when processing new inbound dials.
        ipLimit = 4

        // Burst limit for inbound dials.
        ipBurst = 8

        // High watermark buffer signifies the buffer till which
        // we will handle inbound requests.
        highWatermarkBuffer = 10
)
//SlowMist// ...snip code...
func (s *Service) validateDial(addr multiaddr.Multiaddr) bool {
        ip, err := manet.ToIP(addr)
        if err != nil {
                return false
        }
        remaining := s.ipLimiter.Remaining(ip.String())
        if remaining <= 0 {
                return false
        }
        s.ipLimiter.Add(ip.String(), 1)
        return true
}
```

[P2P]Eclipse Attack

The prevention method is the same as the witch attack mentioned in the context.

[P2P]Eavesdropping Attack

The Libp2p-noise secure channel handshake with secp256k1 identities will be used for encryption.

- cmd/prysmctl/p2p/client.go

```
func newClient(beaconEndpoints []string, clientPort uint) (*client, error) {
        ipAdd := ipAddr()
        priv, err := privKey()
        if err != nil {
                return nil, errors.Wrap(err, "could not set up p2p private key")
        }
    //SlowMist//...snip code...
    options := []libp2p.Option{
                privKeyOption(priv),
                libp2p.ListenAddrs(listen),
                libp2p.UserAgent(version.BuildData()),
                libp2p.Transport(tcp.NewTCPTransport),
        }
```

Message will be encrypted with `GCM` Algorithm.

- p2p/discover/v5wire/encoding.go (go-ethereum)

```
func (c *Codec) encryptMessage(s *session, p Packet, head *Header, headerData []byte)
([]byte, error) {
        // Encode message plaintext.
        c.msgbuf.Reset()
        c.msgbuf.WriteByte(p.Kind())
        if err := rlp.Encode(&c.msgbuf, p); err != nil {
                return nil, err
        }
        messagePT := c.msgbuf.Bytes()

        // Encrypt into message ciphertext buffer.
        messageCT, err := encryptGCM(c.msgctbuf[:0], s.writeKey, head.Nonce[:],
messagePT, headerData)
        if err == nil {
                c.msgctbuf = messageCT
        }
        return messageCT, err
}
```

[P2P]Denial of Service Attack

We reviewed the message structures of the P2P protocol and found no structures that could construct huge network requests, and the message length was reasonably constrained.

[P2P]BGP Hijack Attack

BGP hijacking in blockchain refers to an attacker who has taken control of the network traffic by attacking the backbone network in a certain region to carry out interference or disruption of node block production and synchronization.

Too few nodes in the network or too centralized physical location can easily lead to successful BGP hijacking. In Prysm, a production node can be deployed with a recharge of 32 ETH, and the total number of nodes is so huge that BGP attacks have less impact on the network.

- config/params/mainnet_config.go

```
MaxEffectiveBalance:       32 * 1e9,
```

[P2P]Alien Attack

Communication with peer nodes based on libp2p and discv5.

- github.com/ethereum/go-ethereum/p2p/discover
- github.com/libp2p/go-libp2p

[RPC]Http Input Attack

We examined the RPC interfaces：

```
"/eth/v1/beacon/genesis",
"/eth/v1/beacon/states/{state_id}/root",
"/eth/v1/beacon/states/{state_id}/fork",
"/eth/v1/beacon/states/{state_id}/finality_checkpoints",
"/eth/v1/beacon/states/{state_id}/validators",
"/eth/v1/beacon/states/{state_id}/validators/{validator_id}",
"/eth/v1/beacon/states/{state_id}/validator_balances",
"/eth/v1/beacon/states/{state_id}/committees",
"/eth/v1/beacon/states/{state_id}/sync_committees",
"/eth/v1/beacon/headers",
"/eth/v1/beacon/headers/{block_id}",
"/eth/v1/beacon/blocks",
"/eth/v1/beacon/blinded_blocks",
"/eth/v1/beacon/blocks/{block_id}",
"/eth/v2/beacon/blocks/{block_id}",
"/eth/v1/beacon/blocks/{block_id}/root",
"/eth/v1/beacon/blocks/{block_id}/attestations",
"/eth/v1/beacon/blinded_blocks/{block_id}",
"/eth/v1/beacon/pool/attestations",
"/eth/v1/beacon/pool/attester_slashings",
"/eth/v1/beacon/pool/proposer_slashings",
"/eth/v1/beacon/pool/voluntary_exits",
"/eth/v1/beacon/pool/sync_committees",
"/eth/v1/beacon/weak_subjectivity",
"/eth/v1/node/identity",
"/eth/v1/node/peers",
"/eth/v1/node/peers/{peer_id}",
"/eth/v1/node/peer_count",
"/eth/v1/node/version",
"/eth/v1/node/syncing",
"/eth/v1/node/health",
"/eth/v1/debug/beacon/states/{state_id}",
"/eth/v2/debug/beacon/states/{state_id}",
"/eth/v1/debug/beacon/heads",
"/eth/v2/debug/beacon/heads",
"/eth/v1/debug/beacon/forkchoice",
"/eth/v1/config/fork_schedule",
"/eth/v1/config/deposit_contract",
"/eth/v1/config/spec",
"/eth/v1/events",
"/eth/v1/validator/duties/attester/{epoch}",
"/eth/v1/validator/duties/proposer/{epoch}",
"/eth/v1/validator/duties/sync/{epoch}",
"/eth/v1/validator/blocks/{slot}",
"/eth/v2/validator/blocks/{slot}",
"/eth/v1/validator/blinded_blocks/{slot}",
"/eth/v1/validator/attestation_data",
"/eth/v1/validator/aggregate_attestation",
"/eth/v1/validator/beacon_committee_subscriptions",
"/eth/v1/validator/sync_committee_subscriptions",
"/eth/v1/validator/aggregate_and_proofs",
"/eth/v1/validator/sync_committee_contribution",
"/eth/v1/validator/contribution_and_proofs",
"/eth/v1/validator/prepare_beacon_proposer",
"/eth/v1/validator/register_validator",
```

No traditional web security vulnerabilities such as XSS, SQL injection, CRE, etc. were found

[RPC Layer]Cross-Domain attack

```
$ curl -i http://127.0.0.1:3500/eth/v1/beacon/headers
HTTP/1.1 200 OK
Content-Length: 651
Content-Type: application/json
Date: Sat, 19 Nov 2022 15:05:14 GMT
Vary: Origin
```

Tested locally, cross-domain access is not enabled.

[Off-Chain]Transaction time-locked attack

Prysm does not have UTXO transaction model, no time lock vulerability.

Rug Pull Attack

Reserving administrator privileges for specific accounts was not found in the source code.

[Ledger]Timejacking

Time hijacking is an attack method that prevents other miners from producing blocks properly due to malicious modification of block header timestamps. Prysm is a beacon chain where block headers do not contain timestamps, no security issues were found.

```
type BeaconBlockHeader struct {
        state         protoimpl.MessageState
        sizeCache     protoimpl.SizeCache
        unknownFields protoimpl.UnknownFields

        Slot
github_com_prysmaticlabs_prysm_v3_consensus_types_primitives.Slot
`protobuf:"varint,1,opt,name=slot,proto3" json:"slot,omitempty" cast-
type:"github.com/prysmaticlabs/prysm/v3/consensus-types/primitives.Slot"`
        ProposerIndex
github_com_prysmaticlabs_prysm_v3_consensus_types_primitives.ValidatorIndex
`protobuf:"varint,2,opt,name=proposer_index,json=proposerIndex,proto3"
json:"proposer_index,omitempty" cast-
type:"github.com/prysmaticlabs/prysm/v3/consensus-types/primitives.ValidatorIndex"`
        ParentRoot     []byte
`protobuf:"bytes,3,opt,name=parent_root,json=parentRoot,proto3"
json:"parent_root,omitempty" ssz-size:"32"`
        StateRoot      []byte
`protobuf:"bytes,4,opt,name=state_root,json=stateRoot,proto3"
json:"state_root,omitempty" ssz-size:"32"`
        BodyRoot       []byte
`protobuf:"bytes,5,opt,name=body_root,json=bodyRoot,proto3"
json:"body_root,omitempty" ssz-size:"32"`
}
```

[RPC]Denial of Service Attack

Tried to construct various malformed data to request RPC interface, no interface crash.

```
posturl = "http://127.0.0.1:3500/eth/v1/beacon/headers"
data = '{"' + '}' * 0x101000 + '":' + '{"x":' * 0x10000 + '"}'
print post(posturl, data)
```

[RPC]Eavesdropping Attack

During testing, it was found that the RPC interface does not use https encryption, but by default the port is only open in the local environment and there is no risk of communication listening.

3.3 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	P2P alien attack risk	[P2P]Alien Attack	Low	Acknowledged
N2	blst library has a malleability issue	[Encryption]Insecure Encryption Library Audit	Low	Ignored
N3	Third-party library security	SAST	Suggestion	Acknowledged

4 Findings

4.1 Vulnerability Summary

[N1] [Low] P2P alien attack risk

Category: [P2P]Alien Attack

Content

Prysm uses libp2p for network communication and the discv5 protocol for peers discovery.

For example, when a new peer is discovered, one node tries to shake hands with the target peer via the ping protocol, and if it gets the expected pong response, it adds the address to the libp2p communication address pool.

The Ping/Pong structure of discv5 is

```
// PING is sent during liveness checks.
Ping struct {
    ReqID  []byte
    ENRSeq uint64
}

// PONG is the reply to PING.
Pong struct {
    ReqID  []byte
    ENRSeq uint64
    ToIP   net.IP // These fields should mirror the UDP envelope address of the ping
    ToPort uint16 // packet, which provides a way to discover the external address
(after NAT).
}
```

There is a flaw in the protocol that it cannot distinguish the network to which a node belongs.

We know that Ethereum has a large number of sidechains/sisterchains that use the same network protocol as Ethereum. If the p2p communication protocol cannot distinguish the network to which a node belongs, then it may cause the node to keep trying to communicate with the wrong node, resulting in degraded performance or DoS of the node.

Although there are options such as `DepositChainID/DepositNetworkID` in the node's configuration file, it is not used for the p2p communication protocol.

- config/params/config.go

Solution

Add `NetworkID` to P2P protocol.

Status

Acknowledged

[N2] [Low] blst library has a malleability issue

Category: [Encryption]Insecure Encryption Library Audit

Content

- crypto/bls/blst/signature.go

```
func VerifyCompressed(signature, pub, msg []byte) bool {
        // Validate signature and PKs since we will uncompress them here
        return new(blstSignature).VerifyCompressed(signature, true, pub, true, msg,
dst)
}
```

This function is malleable because the `sig` parameter of `VerifyCompressed` in the library "github.com/supranational/blst/bindings/go" supports both `serialize` and `compress` formats, this means that the signed value can be transformed into another value by a third party without knowledge, leading to unexpected results such as malicious construction of slashing headers or double spending. However, fortunately the function have not been referenced in critical logic.

Solution

Remove `VerifyCompressed` function.

Status

Ignored; The vulnerability exists in blst-v0.2.0, not exist in blst-v0.3.8.

[N3] [Suggestion] Third-party library security

Category: SAST

Content

```
Vulnerability #1: GO-2022-0524
  Calling Reader.Read on an archive containing a large number of
  concatenated 0-length compressed files can cause a panic due to
  stack exhaustion.

  Call stacks in your code:
      testing/endtoend/components/web3remotesigner.go:194:22:
github.com/prysmaticlabs/prysm/v3/testing/endtoend/components.Web3RemoteSigner.Public
Keys calls io.ReadAll, which eventually calls compress/gzip.Reader.Read
```

```
  Found in: compress/gzip@go1.18.3
  Fixed in: compress/gzip@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0524

Vulnerability #2: GO-2022-0523
  Unmarshaling an XML document into a Go struct which has a nested
  field that uses the 'any' field tag can panic due to stack
  exhaustion.

  Call stacks in your code:
      beacon-chain/p2p/service.go:130:22: github.com/prysmaticlabs/prysm/v3/beacon-
chain/p2p.NewService calls github.com/libp2p/go-libp2p.New, which eventually calls
encoding/xml.Decoder.Decode

  Found in: encoding/xml@go1.18.3
  Fixed in: encoding/xml@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0523

Vulnerability #3: GO-2022-0521
  Calling Decoder.Skip when parsing a deeply nested XML document
  can cause a panic due to stack exhaustion.

  Call stacks in your code:
      beacon-chain/p2p/service.go:130:22: github.com/prysmaticlabs/prysm/v3/beacon-
chain/p2p.NewService calls github.com/libp2p/go-libp2p.New, which eventually calls
encoding/xml.Decoder.Decode

  Found in: encoding/xml@go1.18.3
  Fixed in: encoding/xml@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0521

Vulnerability #4: GO-2021-0061
  Due to unbounded alias chasing, a maliciously crafted YAML file
  can cause the system to consume significant system resources. If
  parsing user input, this may be used as a denial of service
  vector.

  Call stacks in your code:
      testing/spectest/shared/phase0/shuffling/core/shuffle/shuffle.go:30:37:
github.com/prysmaticlabs/prysm/v3/testing/spectest/shared/phase0/shuffling/core/shuff
le.RunShuffleTests$1 calls github.com/go-yaml/yaml.Unmarshal

  Found in: github.com/go-yaml/yaml@v2.1.0+incompatible
  Fixed in: github.com/go-yaml/yaml@v2.2.3
  More info: https://pkg.go.dev/vuln/GO-2021-0061

Vulnerability #5: GO-2020-0036
  Due to unbounded aliasing, a crafted YAML file can cause
  consumption of significant system resources. If parsing user
  supplied input, this may be used as a denial of service vector.

  Call stacks in your code:
      testing/spectest/shared/phase0/shuffling/core/shuffle/shuffle.go:30:37:
github.com/prysmaticlabs/prysm/v3/testing/spectest/shared/phase0/shuffling/core/shuff
le.RunShuffleTests$1 calls github.com/go-yaml/yaml.Unmarshal

  Found in: github.com/go-yaml/yaml@v2.1.0+incompatible
  Fixed in: github.com/go-yaml/yaml@v2.2.8
  More info: https://pkg.go.dev/vuln/GO-2020-0036

Vulnerability #6: GO-2022-0515
  Calling any of the Parse functions on Go source code which
  contains deeply nested types or declarations can cause a panic
  due to stack exhaustion.

  Call stacks in your code:
      tools/analyzers/ineffassign/ineffassign.go:40:11:
github.com/prysmaticlabs/prysm/v3/tools/analyzers/ineffassign.builder.walk calls
go/ast.Walk, which eventually calls go/parser.resolver.Visit
      tools/specs-checker/check.go:47:31:
github.com/prysmaticlabs/prysm/v3/tools/specs-checker.inspectFile calls
go/parser.ParseFile

  Found in: go/parser@go1.18.3
  Fixed in: go/parser@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0515

Vulnerability #7: GO-2022-0969
  HTTP/2 server connections can hang forever waiting for a clean
  shutdown that was preempted by a fatal error. This condition can
  be exploited by a malicious client to cause a denial of service.

  Call stacks in your code:
      monitoring/prometheus/simple_server.go:17:30:
github.com/prysmaticlabs/prysm/v3/monitoring/prometheus.RunSimpleServerOrDie calls
net/http.Server.ListenAndServe
      tools/bootnode/bootnode.go:117:31:
github.com/prysmaticlabs/prysm/v3/tools/bootnode.main calls net/http.ListenAndServe

  Found in: net/http@go1.18.3
  Fixed in: net/http@go1.19.1
  More info: https://pkg.go.dev/vuln/GO-2022-0969

Vulnerability #8: GO-2022-0525
  The HTTP/1 client accepted some invalid Transfer-Encoding
  headers as indicating a "chunked" encoding. This could
  potentially allow for request smuggling, but only if combined
  with an intermediate server that also improperly failed to
  reject the header as invalid.

  Call stacks in your code:
      testing/endtoend/helpers/helpers.go:314:7:
github.com/prysmaticlabs/prysm/v3/testing/endtoend/helpers.WaitOnNodes calls
golang.org/x/sync/errgroup.Group.Go, which eventually calls net/http.ReadResponse

  Found in: net/http@go1.18.3
  Fixed in: net/http@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0525

Vulnerability #9: GO-2022-0520
  Client IP adresses may be unintentionally exposed via
  X-Forwarded-For headers. When httputil.ReverseProxy.ServeHTTP is
  called with a Request.Header map containing a nil value for the
  X-Forwarded-For header, ReverseProxy sets the client IP as the
  value of the X-Forwarded-For header, contrary to its
  documentation. In the more usual case where a Director function
  sets the X-Forwarded-For header value to nil, ReverseProxy
  leaves the header unmodified as expected.

  Call stacks in your code:
      beacon-chain/execution/engine_client.go:318:32:
github.com/prysmaticlabs/prysm/v3/beacon-chain/execution.Service.ExecutionBlockByHash
calls github.com/ethereum/go-ethereum/rpc.Client.CallContext, which eventually calls
net/http.Header.Clone

  Found in: net/http@go1.18.3
  Fixed in: net/http@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0520

Vulnerability #10: GO-2022-0522
  Calling Glob on a path which contains a large number of path
  separators can cause a panic due to stack exhaustion.

  Call stacks in your code:
      validator/accounts/wallet/wallet.go:384:31:
github.com/prysmaticlabs/prysm/v3/validator/accounts/wallet.Wallet.FileNameAtPath
calls path/filepath.Glob

  Found in: path/filepath@go1.18.3
  Fixed in: path/filepath@go1.18.4
  More info: https://pkg.go.dev/vuln/GO-2022-0522

Vulnerability #11: GO-2022-1039
  Programs which compile regular expressions from untrusted
  sources may be vulnerable to memory exhaustion or denial of
  service. The parsed regexp representation is linear in the size
  of the input, but in some cases the constant factor can be as
  high as 40,000, making relatively small regexps consume much
  larger amounts of memory. After fix, each regexp being parsed is
  limited to a 256 MB memory footprint. Regular expressions whose
  representation would use more space than that are rejected.
  Normal use of regular expressions is unaffected.

  Call stacks in your code:
      api/gateway/gateway.go:107:27:
github.com/prysmaticlabs/prysm/v3/api/gateway.Gateway.Start calls
github.com/gorilla/mux.Router.PathPrefix, which eventually calls regexp/syntax.Parse

  Found in: regexp/syntax@go1.18.3
  Fixed in: regexp/syntax@go1.19.2
  More info: https://pkg.go.dev/vuln/GO-2022-1039
```

Solution

Decide whether to do defensive coding based on actual business.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002211210001	SlowMist Security Team	2022.11.01 - 2022.11.21	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 low risk, 1 suggestion vulnerabilities. And 1 low risk vulnerabilities were ignored;

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.