Obol Network

Charon

Security Assessment Report

Version: 2.1

June, 2023

Contents

Introduction 2
- Disclaimer . . . 2
- Document Structure . . . 2
- Overview . . . 2

Security Assessment Summary 3
- Findings Summary . . . 3

Detailed Findings 4

Summary of Findings 5
- Kryptology Does Not Verify The Length of Commitments . . . 6
- Frost DKG Does Not Validate Message Source Matches the Sender . . . 8
- `nil` Pointer References From Protobuf Messages . . . 10
- Inadequate Input Parameter Checks Resulting in Slice Bounds Out-of-Range Error . . . 13
- Lack of Size Checks When Slicing Arrays . . . 16
- Insufficient Validation of Consensus Messages Leads to Panics . . . 17
- QBFT Consensus Allows Replay of Justification Messages . . . 19
- Insufficient Error Handling . . . 20
- Panics in `coinbase/kryptology` Frost Protocol . . . 22
- Frost Broadcast Messages Do Not Use Reliable Broadcast . . . 24
- Vulenerable Dependencies . . . 25
- Outdated Dependencies . . . 26
- RLP Length in Bits Rather Than Bytes . . . 28
- Duplicate Keys Allowed in ENR . . . 30
- Insufficient Validation of Consensus Message Types . . . 31
- `sigagg` Does Not Ensure `t` Partials Are Received . . . 32
- `sigagg` Does Not Verify the Reconstructed Signature . . . 33
- Aggregate Lock Only Collects `t` Signatures Rather Than `n` . . . 34
- CreateDKG Allows the Threshold to be Larger Than the Number of Operators . . . 35
- CreateDKG Does Not Validate the Checksum of the Withdrawal Addresses . . . 36
- Miscellaneous General Comments . . . 37

A Test Suite 38

B Vulnerability Severity Classification 41

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Obol Network's distributed validator client (Charon) implementation.

The review focused solely on the security aspects of the Golang implementation of the solution, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contracts or other in-scope items. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Obol Network's distributed validator client (Charon) code contained within the scope of the security review.

A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Obol Network's distributed validator client (Charon) implementation.

Overview

Distributed validator client Charon is a HTTP middleware client for Ethereum staking that enables users to run a single validator across a group of independent nodes.

Charon is accompanied by a web application called the Distributed Validator Launchpad for distributed validator key creation.

Charon is used by stakers to distribute the responsibility of running Ethereum Validators across a number of different instances and client implementations.

Security Assessment Summary

This time-boxed security review was conducted on the files hosted on the Obol Network's Charon repository and were assessed at commit 707b07a.

Note: native Go and Go Ethereum libraries and any external dependencies were excluded from the primary focus of this assessment.

The manual code review section of the report is focused on identifying issues/vulnerabilities associated with the business logic implementation of the components in scope. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Go runtime and Ethereum protocol.

To support this review, the testing team used the following automated testing tools:

- `golangci-lint`: `https://github.com/golangci/golangci-lint`
- `semgrep-go`: `https://github.com/dgryski/semgrep-go`
- native go fuzzing: `https://go.dev/doc/fuzz/`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 21 issues during this assessment. Categorised by their severity:

- Critical: 2 issues.
- High: 5 issues.
- Medium: 3 issues.
- Low: 5 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Obol Network's distributed validator client (Charon) implementation.

Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
OBOL-01	Kryptology Does Not Verify The Length of Commitments	Critical	Resolved
OBOL-02	Frost DKG Does Not Validate Message Source Matches the Sender	Critical	Resolved
OBOL-03	`nil` Pointer References From Protobuf Messages	High	Resolved
OBOL-04	Inadequate Input Parameter Checks Resulting in Slice Bounds Out-of-Range Error	High	Resolved
OBOL-05	Lack of Size Checks When Slicing Arrays	High	Resolved
OBOL-06	Insufficient Validation of Consensus Messages Leads to Panics	High	Resolved
OBOL-07	QBFT Consensus Allows Replay of Justification Messages	High	Resolved
OBOL-08	Insufficient Error Handling	Medium	Closed
OBOL-09	Panics in `coinbase/kryptology` Frost Protocol	Medium	Resolved
OBOL-10	Frost Broadcast Messages Do Not Use Reliable Broadcast	Medium	Resolved
OBOL-11	Vulenerable Dependencies	Low	Closed
OBOL-12	Outdated Dependencies	Low	Closed
OBOL-13	RLP Length in Bits Rather Than Bytes	Low	Resolved
OBOL-14	Duplicate Keys Allowed in ENR	Low	Resolved
OBOL-15	Insufficient Validation of Consensus Message Types	Low	Resolved
OBOL-16	`sigagg` Does Not Ensure `t` Partials Are Received	Informational	Resolved
OBOL-17	`sigagg` Does Not Verify the Reconstructed Signature	Informational	Resolved
OBOL-18	Aggregate Lock Only Collects `t` Signatures Rather Than `n`	Informational	Closed
OBOL-19	CreateDKG Allows the Threshold to be Larger Than the Number of Operators	Informational	Resolved
OBOL-20	CreateDKG Does Not Validate the Checksum of the Withdrawal Addresses	Informational	Resolved
OBOL-21	Miscellaneous General Comments	Informational	Resolved

OBOL-01	Kryptology Does Not Verify The Length of Commitments		
Asset	`github.com/coinbase/kryptology`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

During the Frost DKG, parties participating in the protocol first commit to a polynomial of degree `t`, where `t` is the threshold of malicious nodes. The verification of the polynomial commitments does not ensure the length is `t+1` (a degree `t` polynomial has `t+1` coefficients).

The impact of forgoing these checks is an attacker can arbitrarily increase the length of the polynomial by creating commitments larger than `t+1`.

The vulnerable code can be found in the coinbase/kryptology repository. The following verifier shows the lack of checks to ensure the length of `Commitments` is `t+1`.

```
type FeldmanVerifier struct {
    Commitments []curves.Point
}


func (v FeldmanVerifier) Verify(share *ShamirShare) error {
    curve := curves.GetCurveByName(v.Commitments[0].CurveName())
    err := share.Validate(curve)
    if err != nil {
        return err
    }
    x := curve.Scalar.New(int(share.Id))
    i := curve.Scalar.One()
    rhs := v.Commitments[0]


    for j := 1; j < len(v.Commitments); j++ {
        i = i.Mul(x)
        rhs = rhs.Add(v.Commitments[j].Mul(i))
    }
    sc, _ := curve.Scalar.SetBytes(share.Value)
    lhs := v.Commitments[0].Generator().Mul(sc)


    if lhs.Equal(rhs) {
        return nil
    } else {
        return fmt.Errorf("not equal")
    }
}
```

The severity is rated as high as it undermines the core assumptions of Frost DKG, that polynomial commitments are of length `t+1`. The security proofs provided in the academic paper may no longer hold true.

Recommendations

To resolve the issue, ensure that the length of `Commitments` received from other parties in the DKG is `t+1`. This can be achieved by modifying the `coinbase/kryptology` repository or adding checks to `charon` when the messages are

received.

Note here we are referring to the "threshold" `t` as the degree of the polynomial which has `t+1` coefficients. Some papers and implementations use the terminology of "threshold" as the number of parties required to recreate the shared secret. That is "threshold" may refer to either the number of coefficients or the degree of the polynomial depending on the context.

Resolution

The issue has been resolved in PR #2007 which checks the length of the commitments when they are initially received, during round 1 of Frost DKG.

OBOL-02	Frost DKG Does Not Validate Message Source Matches the Sender		
Asset	`dkg/frostp2p.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The Frost DKG implementation receives messages from peers. Each message contains the type `FrostMsgKey` which contains three fields.

- `ValIdx` : The validator index (0 indexed)
- `SourceId` : The sender share index (1 indexed)
- `TargetId` : The receiver share index (1 indexed)

Below is a snippet of the code used to decode the round 1 messages and pass them to the Frost protocol. There is no validation of the `FrostMsgKey` fields with each message received.

```
tcpNode.SetStreamHandler(round1Protocol(clusterID), func(s network.Stream) {
  ctx = log.WithCtx(ctx, z.Str("peer", p2p.PeerName(s.Conn().RemotePeer())))
  defer s.Close()

  b, err := io.ReadAll(s)
  if err != nil {
    log.Error(ctx, "Read round 1 wire", err)
    return
  }
  msg := new(pb.FrostRound1Msg)
  if err := proto.Unmarshal(b, msg); err != nil {
    log.Error(ctx, "Unmarshal round 1 proto", err)
    return
  }

  mu.Lock()
  defer mu.Unlock()

  pID := s.Conn().RemotePeer()
  if !knownPeers[pID] {
    log.Warn(ctx, "Ignoring unknown round 1 peer", nil, z.Any("peer", p2p.PeerName(pID)))
    return
  } else if dedupRound1[pID] {
    log.Debug(ctx, "Ignoring duplicate round 1 message", z.Any("peer", p2p.PeerName(pID)))
    return
  }
  dedupRound1[pID] = true

  round1Recv <- msg
})
```

Since `SourceId` is not validated, a peer can send messages on behalf of other peers. A malicious attacker could send message on behalf of other peers to modify their commitments. This would allow the attacker to gain control over more than a threshold number of shares and thereby recover the secret key.

The issue is present for messages received in both round 1 and 2. However, round 1 has been used as an example above.

Recommendations

`FrostMsgKey` can be found in each of the following messages.

- `FrostRound1Msg`
  - Each element of `Casts []*FrostRound1Cast`
  - Each element of `P2Ps ShamirShare`
- `FrostRound2Msg`
  - Each element of `Casts []*FrostRound2Cast`

For each `FrostMsgKey` that are listed above perform the following validation.

- `0 <= ValIdx < numValidators`
- `1 <= SourceId <= numNodes`
- `1 <= TargetId <= numNodes`
- `SourceId` matches the peer ID (i.e. can check `frostP2P.peers[sourceId] == s.Conn().RemotePeer()`)
- `TargetId` matches our peer ID

Note that `nil` pointer checks need to be performed before accessing any of the above listed pointers. See OBOL-03 for more details on `nil` pointers in Protobuf.

Resolution

The recommendations have been implemented in PRs #1896 and #2107.

OBOL-03	`nil` Pointer References From Protobuf Messages		
Asset	`dkg/frostp2p.go`, `core/parsigex/parsigex.go`, `core/proto.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Protobuf allows decoding pointer objects to `nil`. A `nil` pointer reference occurs when a program attempts to access or dereference a pointer that has not been initialised, resulting in a runtime panic.

DKG

Charon's DKG functionality is susceptible to unexpected panics due to `nil` pointer references. It is possible for an attacker to exploit this vulnerability by sending a specifically crafted payload as a `FrostRound1Msg` or `FrostRound2Msg` message, causing the service to crash.

The following payloads were found to trigger unexpected panics:

- `\x92\x00\x00` as `FrostRound1Msg`, causing panic in `shamirShareFromProto()`
- `"\x0a\x44\x12\x20" + 66 * "\x41"` as `FrostRound1Msg`, causing panic in `round1CastFromProto()`
- `\x0a\x06\x12\x01\xd6\x1a\x01\xd6` as `FrostRound2Msg`, causing panic in `round2CastFromProto()`

The panics occur in `dkg/frostp2p.go` at:

- `shamirShareFromProto()` line [**314**] due to referencing `shamir.Key`, which is intentially `nil` in the crafted payload
- `round2CastFromProto()` line [**379**] due to referencing `cast.Key`, which is intentially `nil` in the crafted payload
- `round1CastFromProto()` line [**354**] due to referencing `cast.Key`, which is intentially `nil` in the crafted payload

Core

The following functions will result in a `nil` pointer panic in `core/proto.go` if `nil` is passed as one of the parameters.

- `DutyFromProto(duty *pbv1.Duty)` if `duty` is `nil`.
- `ParSignedDataFromProto(typ DutyType, data *pbv1.ParSignedData)` if `data` is `nil`.
- `ParSignedDataSetFromProto(typ DutyType, set *pbv1.ParSignedDataSet)` if `set` is `nil`.
- `UnsignedDataSetFromProto(typ DutyType, set *pbv1.UnsignedDataSet)` if `set` is `nil`.

In the file `core/parsigex/parsigex.go` in the function `handle()` if either `pb.DataSet` or `pb.Duty` are `nil` then the above-mentioned functions `DutyFromProto()` and `ParSigDataSetFromProto()` will panic. Similarly, if the map `pb.DataSet.Set` contains a `nil` value for any of the `*ParSignedData` objects then `ParSignedDataFromProto()` will panic.

Consensus

In addition to the panics that are mentioned in OBOL-06, there are `nil` pointer panics that may arise from decoding protobuf objects as `nil`.

When parsing and executing consensus messages it is possible for `nil` pointer exceptions to arise. The recursive decoding of messages and justifications in `core/consensus/msg.go::newMsg()` will panic if either `pbMsg` or any element in the array `justification` is `nil`. Due to the recursive nature of each justification calling `newMsg(j, nil)` on line [57] it is possible for `nil` pointers to arise on objects not checked by the calling function.

Recommendations

Use defensive programming techniques such as checking pointers for `nil` values before attempting to access or dereference them.

It is recommended to add `nil` checks to all functions that take a pointer as a parameter or decode a struct which is or contains a pointer. However, at a minimum this technique may be added to the following functions.

- `core/proto.go::DutyFromProto()`
- `core/proto.go::ParSignedDataFromProto()`
- `core/proto.go::ParSignedDataSetFromProto()`
- `core/proto.go::UnsignedDataSetFromProto()`
- `core/parsigex/parsigex.go::handle()`
- `dkg/frostp2p.go::round1CastFromProto()`
- `dkg/frostp2p.go::round2CastFromProto()`
- `dkg/frostp2p.go::shamirShareFromProto()`
- `dkg/frostp2p.go::keyFromProto()`
- `dkg/frostp2p.go::newFrostP2P()` (specifically the stream handlers)
- `core/priority/component.go::topicResultFromProto()`
- `core/priority/prioritiser.go::handleRequest()`
- `core/consensus/component.go::handle()`
- `core/consensus/msg.go::newMsg()`

Resolution

A generic `nil` checker package called `protonil` has been developed by the Obol team. The `protonil.Check()` function recursively checks protobuf messages for `nil` values where applicable. The `protonil.Check()` function is executed in Charon for each libp2p message when it is received. Changes can be seen in PR #2346.

Additionally, manual validation of pointers has been added in handler functions to prevent `nil` pointers from being deferenced.

OBOL-04	Inadequate Input Parameter Checks Resulting in Slice Bounds Out-of-Range Error		
Asset	`eth2util/rlp/rlp.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The current implementation of the code lacks sufficient input parameter checks, which results in invalid values generated that can cause unexpected panics when used with slicing arrays. This issue is prevalent in RLP and ENR related parts of the codebase and can cause significant disruptions to the system's functionality.

By using specifically crafted payloads, it is possible to trigger unexpected panics due to slice bounds out of range errors.

The following payloads were found to trigger the panics:

- `[]byte("\xbf\x9b00000000")` in `DecodeBytes()`
- `[]byte("\x87\xff\xff\xff\xff\xff\xff\xff\xff\xff0")` in `DecodeBytesList()`

The errors trigger in `code/charon/eth2util/rlp/rlp.go` line [**88**] and `DecodeBytesList()` line [**54**] respectively.

This is due to both functions calling `decodeLength()`, which in turn calls `fromBigEndian()`, which does not have bounds checks on provided input and may return incorrect `length` and `offset` values. Once these values are used to slice arrays, unexpected bounds out of range errors may occur.

Furthermore, `decodeLength()` may be negative as `fromBigEndian()` casts a `uint64` to `int` which may result in a negative value if the value is larger the 2^{63} (noting only 64 bit Operating Systems are supported). The length of decoded bytes should be non-negative to prevent infinite loop attacks and negative slice indexing.

```go
// DecodeBytesList returns the list of byte slices contained in the given RLP encoded byte slice.
func DecodeBytesList(input []byte) ([][]byte, error) {
  if len(input) == 0 {
    return nil, nil
  }

  offset, length, err := decodeLength(input)
  if err != nil {
    return nil, err
  }

  if offset+length > len(input) { // @audit may overflow or be negative
    return nil, errors.New("input too short")
  }

  var items [][]byte
  for i := offset; i < offset+length; {
    itemOffset, itemLength, err := decodeLength(input[i:]) // @audit unsafe slice index
    if err != nil {
      return nil, err
    }

    start := i + itemOffset
    end := i + itemOffset + itemLength // @audit may overflow or be less than start

    if end > len(input) {
      return nil, errors.New("input too short")
    }

    items = append(items, input[start:end]) // @audit unsafe slice index

    i = end
  }

  return items, nil
}
```

```go
func DecodeBytes(input []byte) ([]byte, error) {
  if len(input) == 0 {
    return nil, nil
  }

  offset, length, err := decodeLength(input)
  if err != nil {
    return nil, err
  }

  if offset+length > len(input) { // @audit may overflow or have negative numbers
    return nil, errors.New("input too short")
  }

  return input[offset : offset+length], nil
}
```

```go
// fromBigEndian returns the integer encoded as big endian at the provided byte slice offset and length.
func fromBigEndian(b []byte, offset int, length int) int { //@audit validate offset and length are non-negative and less than
      ↪  len(b)
  var x uint64
  for i := offset; i < offset+length; i++ {
    x = x<<8 | uint64(b[i]) // @audit unsafe slice indexing
  }

  return int(x) //@audit may cast to negative value
}
```

Recommendations

Implement input parameter checks to prevent invalid values from being used as slice indices. This can be done by checking that the input parameters are within the range of valid values before using them in slicing arrays.

For example:

```
// Add in DecodeBytesList()
if end > len(input) || start < 0 || end < 0 || start > end {
  return nil, errors.New("input too short")
}

// Add in DecodeBytes()
if offset+length > len(input) || length < 0 || offset < 0 || offset > offset+length {
  return nil, errors.New("input too short")
}

// Add in fromBigEndian()
if offset >= len(b) || offset+length >= len(b) {
  // return an error of fixed value, depending on business requirements
}
```

This will prevent unexpected panics and ensure the system functions as intended.

Additionally, prevent `decodeLength()` from returning a negative `length`.

Resolution

This finding has been resolved and recommendations implemented in PR 1990.

OBOL-05	Lack of Size Checks When Slicing Arrays		
Asset	`cmd/relay/p2p.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The reviewed code contains instances where arrays are being sliced without first checking the size of the array.

This can result in the program crashing if the array being sliced is of insufficient size. This is particularly problematic in the context of p2p when receiving data from other connected peers, as malicious peers may intentially send malformed data to crash the system.

In `cmd/relay/p2p.go` on line [220]:

```
199 // getPeerInfo returns the peer's cluster hash and true.
    func getPeerInfo(ctx context.Context, tcpNode host.Host, pID peer.ID, name string) (string, bool, error) {
201   info, rtt, ok, err := peerinfo.DoOnce(ctx, tcpNode, pID)
      if p2p.IsRelayError(err) {
203     // Ignore relay errors, since peer probably not connected anymore.
        return "", false, nil
205   } else if err != nil {
        return "", false, err
207   } else if !ok {
        // Group peers that don't support the protocol with unknown cluster hash.
209     return unknownCluster, true, nil
      }
211
      hash := clusterHash(info.LockHash)
213   peerPingLatency.WithLabelValues(name, hash).Observe(rtt.Seconds() / 2)

215   return hash, true, nil
    }
217
    // clusterHash returns the cluster hash hex from the lock hash.
219 func clusterHash(lockHash []byte) string {
      return hex.EncodeToString(lockHash)[:7]      // @audit No size checks here, if lockHash <= 3 characters, this will panic
221 }
```

The lack of size checks when slicing arrays can result in the program crashing or becoming unstable if the sliced array is of insufficient size. This can lead to unexpected crashes affecting overall availability of the system.

Recommendations

Review all instances where arrays are being sliced to ensure that size checks are being performed prior to slicing. Specifically, the program should check the size of the array before attempting to slice it, and should handle any errors that occur gracefully.

Resolution

This finding has been resolved and recommendations implemented in PR 2077.

OBOL-06	Insufficient Validation of Consensus Messages Leads to Panics		
Asset	`core/consensus/component.go` & `core/consensus/msg.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Multiple panics are reachable in the consensus `handle()` function.

First, a panic is reachable in `verifyMsgSign()` if public key is `nil`. The case where `pubkey` is `nil` arises when a consensus message is sent with a `PeerIdx` which does not have a corresponding public share.

Examining the code snippet below `c.pubkeys[pbMsg.Msg.PeerIdx]` will return `nil` if the `PeerIdx` is not mapped. That is if `PeerIdx <= 0` or `PeerIdx > numValidators`.

A second panic can be reached if `pbMsg.Justification` has an element in the array that is `nil`. That is if `msg` is `nil` on line [**357**]. It will access `msg.PeerIdx` which results in a `nil` pointer panic.

```
351 if ok, err := verifyMsgSig(pbMsg.Msg, c.pubkeys[pbMsg.Msg.PeerIdx]); err != nil { //@audit may pass a `nil` pubkey
        return nil, false, errors.Wrap(err, "verify consensus message signature", z.Any("duty", duty))
353 } else if !ok {
        return nil, false, errors.New("invalid consensus message signature", z.Any("duty", duty))
355 }

357 for _, msg := range pbMsg.Justification {
        if ok, err := verifyMsgSig(msg, c.pubkeys[msg.PeerIdx]); err != nil { // @audit may pass a `nil` pubkey or panics if `msg` is
            ↪  `nil`
359         return nil, false, errors.Wrap(err, "verify consensus justification signature", z.Any("duty", duty))
        } else if !ok {
361         return nil, false, errors.New("invalid consensus justification signature", z.Any("duty", duty))
        }
363 }
```

During `verifyMsgSig()` if `pubkey` is `nil` it will panic when accessed in `recovered.IsEqual(pubkey)`.

```
func verifyMsgSig(msg *pbv1.QBFTMsg, pubkey *k1.PublicKey) (bool, error) {
  if msg.Signature == nil {
        return false, errors.New("empty signature")
        }

        clone := proto.Clone(msg).(*pbv1.QBFTMsg)
  clone.Signature = nil
  hash, err := hashProto(clone)
  if err != nil {
    return false, err
        }

  recovered, err := k1util.Recover(hash[:], msg.Signature)
  if err != nil {
        return false, errors.Wrap(err, "recover pubkey")
        }

        return recovered.IsEqual(pubkey), nil
        }
```

Recommendations

It is recommended to perform `nil` checks in `verifyMsgSig()` to ensure both `pubkey` and `msg` are not `nil`.

Furthermore, ensure each index of `c.pubkeys` is checked through `value, exists := map[index]` syntax and the key exists.

Additionally ensure each element of the array `pbMsg.Justification` is not `nil`.

Resolution

This finding has been resolved and recommendations implemented in PR 2040.

OBOL-07	QBFT Consensus Allows Replay of Justification Messages		
Asset	`core/consensus/component.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

It is possible to replay consensus justification messages from other duties.

There are a lack of checks to ensure the duty of a justification matches the duty of the message.

The check should occur in `core/consensus/component.go` when validating justifications.

```
for _, msg := range pbMsg.Justification {
  if ok, err := verifyMsgSig(msg, c.pubkeys[msg.PeerIdx]); err != nil {
    return nil, false, errors.Wrap(err, "verify consensus justification signature", z.Any("duty", duty))
  } else if !ok {
    return nil, false, errors.New("invalid consensus justification signature", z.Any("duty", duty))
  }
}
```

The impact is a malicious node could have messages justified which were not intended for this duty.

Recommendations

Ensure the duty in each justification exactly matches that of the message.

Resolution

This finding has been resolved and recommendations implemented in PR 2079.

OBOL-08	Insufficient Error Handling		
Asset	`app/eth2wrap/eth2wrap.go`, `app/lifecycle/manager.go`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Several instances in the code were identified where panic is triggered when an error or an edge case is encountered. Insufficient error handling can result in the software crashing or becoming unavailable when errors are encountered.

While panics may help developers identify and fix issues during development, they can have negative impacts on the stability and availability of the software in production. To ensure that the software remains available and operational, it is important to handle errors gracefully, rather than relying on panics to identify issues.

Some examples of panics that should be handled gracefully instead include (note, this may not be exhaustive list of all occurrences in the code):

- `/app/eth2wrap/eth2wrap.go` on line [**376**]:

```
func (s *bestSelector) Increment(i int) {
  s.mu.Lock()
  defer s.mu.Unlock()

  if i < 0 || i >= s.n {
    panic("invalid index") // This should never happen
  }

  if time.Since(s.start) > s.period { // Reset counters after period.
    s.counts = make([]int, s.n)
    s.start = time.Now()
  }

  s.counts[i]++
}
```

- `app/lifecycle/manager.go` on line [**49**] and line [**68**]:

```
if m.started {
  panic("cycle already started")
}
```

Recommendations

Review all instances of panic in the code and replace them with more robust error-handling mechanisms.

Specifically, errors should be handled gracefully, and the system should be designed to continue operating in the event of an error. This will help ensure that the software remains stable and available at all times.

Resolution

The Obol development team has advised that this issue will not be addressed and marked it as `WONTFIX`.

OBOL-09	Panics in `coinbase/kryptology` Frost Protocol		
Asset	`github.com/coinbase/kryptology`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

A reachable panic exists in the external library github.com/coinbase/kryptology repository. The panic exists in the Frost DKG round 2 if there is an ID in basic cast that does not have a Shamir share.

The following code snippet is taken from `kryptology/pkg/dkg/frost/dkg_round2.go` for the function `Round2()`.

```
func (dp *DkgParticipant) Round2(bcast map[uint32]*Round1Bcast, p2psend map[uint32]*sharing.ShamirShare) (*Round2Bcast, error) {

    // ... snipped for brevity

    // Step 2 - for j in 1,...,n
  for id := range bcast {

    // ... snipped for brevity

    // Step 5 - FeldmanVerify
    fji := p2psend[id] // @audit this value may not exist and return nil
    if err = bcast[id].Verifiers.Verify(fji); err != nil { // @audit panics in `Verify(nil)`
      return nil, fmt.Errorf("feldman verify fails for participant with id %d\n", id)
    }
  }
```

The panic occurs if there is a key in the map `bcast` which does not exist in `p2psend`. This will cause the value `fji` to be `nil` and result in a `nil` pointer exception in the function `FeldmanVerifier.Verify()`.

The panic is reachable from the Charon protocol by supplying round 2 messages with invalid IDs. The issue OBOL-02 should prevent this issue by performing validation on the IDs.

Recommendations

The issue should be further mitigated in the upstream `coinbase/krpytology` library. First, ensure that `id` exists in the map `p2psend` by using the `value, exists := map[key]` syntax.

Second, ensure the pointer retrieved is non-nil, that is check `fji != nil`.

Finally, add a check to `FeldmanVerifier.Verify()` to ensure `share != nil`.

Although the second point is sufficient to prevent any `nil` pointer exceptions from arising in `charon`, all three solutions are recommended to increase the robustness of the code. This will prevent the introduction of bugs in future updates or when used by third parties.

Resolution

The finding has been resolved by implementing fix for OBOL-02 (PR 1896 and PR 2107).

OBOL-10	Frost Broadcast Messages Do Not Use Reliable Broadcast		
Asset	`dkg/frostp2p.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Frost protocol can be instantiated using either *reliable broadcast* or a *Signature Aggregator*. The Charon implementation of reliable broadcast for Frost DKG is not correctly implemented. Reliable broadcast requires all parties to receive identical messages during a broadcast. The current implementation of broadcast will send each message directly to a peer and ensure it is received.

The issue with the current implementation is that a malicious party may "broadcast" different messages to each peer. The peers will be unaware they have received a different message to other parties can will continue operating the protocol.

This undermines the security assumptions of the protocol and invalidates the security proof. It is therefore possible for unforeseen attacks to interrupt or expose secrets of the DKG.

Recommendations

To mitigate the issue either implement a *Signature Aggregator* or modify the broadcast to implement *reliable broadcast*.

Resolution

Reliable broadcast has been implemented in PR #1896.

OBOL-11	Vulenerable Dependencies		
Asset	`charon/*`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The reviewed code base indirectly uses a vulnerable dependency `golang/github.com/btcsuite/btcd@v0.22.1`. This version of the dependency is known to be vulnerable to two known issues - CVE-2022-39389: Improper Input Validation and CVE-2022-44797: Improper Restriction of Operations within the Bounds of a Memory Buffer.

`CVE-2022-39389` could cause a node to enter a degraded state if exploited, whilst `CVE-2022-44797` mishandles witness size checking, which could lead to unexpected behaviour or panics.

Note, due to the vulnerable library being an indirect dependency to `Charon`, it is unclear whether it is directly exploitable in the reviewed codebase.

Recommendations

Update the dependency to the latest version that addresses the known vulnerabilities.

Additionally, it is important to regularly review and update all dependencies to ensure that they remain up-to-date and secure.

Resolution

The Obol development team advised the dependency is not used, as it is used for the `secp256k1` curve in the Kryptology library which is not called by Obol.

OBOL-12	Outdated Dependencies		
Asset	`charon/*`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The reviewed code base utilises several external dependencies that are not up-to-date with the latest available versions.

The following lists identified outdated dependencies and their latest versions available as at the time of writing:

1. github.com/bufbuild/buf v1.14.0 (latest version is v1.17.1)
2. github.com/golang/snappy v0.0.4 (latest version is v0.0.5)
3. github.com/gorilla/mux v1.8.0 (latest version is v1.8.1)
4. github.com/spf13/cobra v1.6.1 (latest version is v1.6.3)
5. github.com/spf13/pflag v1.0.5 (latest version is v1.0.8)
6. github.com/spf13/viper v1.15.0 (latest version is v1.16.0)
7. github.com/stretchr/testify v1.8.1 (latest version is v1.8.3)
8. golang.org/x/oauth2 v0.5.0 (latest version is v0.5.2)
9. gopkg.in/cenkalti/backoff.v1 v1.1.0 (latest version is v2.2.1)

The use of outdated dependencies could leave the system vulnerable to exploitation by attackers. As vulnerabilities and bugs are often discovered and patched in later versions of software, using outdated dependencies increases the likelihood of an attacker finding and exploiting these flaws.

A special note is on the dependency coinbase/kryptology which has been marked as archived as of 9th September 2022. It is strongly advised not to rely on archived repositories.

Recommendations

Update external dependencies to their latest versions available to ensure that the system is protected against any known bugs and vulnerabilities.

Additionally, it is important to regularly review and update dependencies to ensure that they remain up-to-date and secure.

It is recommended to either swap out `coinbase/kryptology` for another cryptography library or fork it and actively maintain the fork.

Resolution

The Obol development team advised that all dependencies are tracked by `Dependabot` and are kept up-to-date.

OBOL-13	RLP Length in Bits Rather Than Bytes		
Asset	`eth2util/rlp/rlp.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

RLP decoding validates the length is not more than 64 *bits* before passing this value to `fromBigEndian()` which expects *bytes*.

The function `decodeLength()` performs the deserialisation of `length` as follows.

```
func decodeLength(item []byte) (offset int, length int, err error) {
    if len(item) == 0 {
        return 0, 0, errors.New("input too short")
    }

    prefix := item[0]

    if prefix < 0x80 {
        return 0, 1, nil
    }

    if prefix < 0xb8 {
        return 1, int(prefix - 0x80), nil
    }

    if prefix < 0xc0 {
        length = int(prefix - 0xb7)
        if length > 64 { // @audit 64 bits rather than 8 bytes
            return 0, 0, errors.New("invalid length prefix")
        }

        return 1 + length, fromBigEndian(item, 1, length), nil
    }

    if prefix < 0xf8 {
        return 1, int(prefix - 0xc0), nil
    }

    length = int(prefix - 0xf7)
    if length > 64 { // @audit 64 bits rather than 8 bytes
        return 0, 0, errors.New("invalid length prefix")
    }

    return 1 + length, fromBigEndian(item, 1, length), nil
}
```

The function `fromBigEndian()` expects `length` to be in units of bytes rather than bits. It is therefore possible to have a `length` larger than the `uint64`.

Recommendations

For both cases mentioned above ensure that `length` is less than or equal to 8 rather than 64 and encode values in bytes rather than bits.

Resolution

This finding has been resolved and recommendations implemented in PR 2081.

OBOL-14	Duplicate Keys Allowed in ENR		
Asset	`eth2util/enr/enr.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Duplicate keys may be used in the ENR maping. The last key in the set of the duplicates will overwrite the previously recorded values.

EIP778 states that each key should only occur once in the mapping as seen by the quoted text below.

The key/value pairs must be sorted by key and must be unique, i.e. any key may be present only once.

The implementation in part of the function `Parse()` does not check for unqiue keys and overwrites the `r.kvs` with new values when a duplicate key occurs.

```
82  for i := 2; i < len(elements); i += 2 {
      r.kvs[string(elements[i])] = elements[i+1] // @audit should check existence before overwriting
84
      switch string(elements[i]) {
86    case keySecp256k1:
        r.PubKey, err = k1.ParsePubKey(elements[i+1])
88      if err != nil {
          return Record{}, errors.Wrap(err, "invalid secp256k1 public key")
90      }
      case keyID:
92      if string(elements[i+1]) != valID {
          return Record{}, errors.New("non-v4 identity scheme not supported")
94      }
      }
96  }
```

Recommendations

The recommendation is to add a check in each iteration of the above loop to ensure `r.kvs` does not have an entry for the current key.

Resolution

This finding has been resolved and recommendations implemented in PR 2073.

OBOL-15	Insufficient Validation of Consensus Message Types		
Asset	`core/consensus/component.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There is insufficient validation of `pbv1.ConsensusMsg.Msg.Type` and `pbv1.ConsensusMsg.Justification[i].Msg.Type`. A type value may be supplied which is greater than or equal to `msgSentinel` or less than or equal to `MsgUnknown`.

The impact is a panic will occur in `core/qbft/qbft.go` in the function `isJustified()`, however this panic recovers in `Run()`. As the panic recovers this is rated a low severity issue.

Recommendations

To avoid panics proper validation should occur for each message and justification.

This may be achieved by performing the check `qbft.MsgType(type).Valid()` for each message and justification. Consider adding these checks to the function `core/consensus/msg.go::newMsg()`.

Resolution

Additional checks are performed on each consensus message in PR #2115.

OBOL-16	`sigagg` Does Not Ensure `t` Partials Are Received
Asset	`core/sigagg/sigagg.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Within the `sigagg` module there is lack of validation that `Aggregate()` has received `t` unqiue parital signatures.

The function takes `parSigs` as a parameter, which are converted to the `blsSigs` mapping. There is a lack of validation to ensure there are not duplicate `parSig.ShareIdx` within the `parSigs` array.

If any duplicates occur then `len(blsSigs)` will be below the threshold and the reconstructed aggregate will be invalid.

```
func (a *Aggregator) Aggregate(ctx context.Context, duty core.Duty, pubkey core.PubKey, parSigs []core.ParSignedData) error {
    ctx = log.WithTopic(ctx, "sigagg")

    if len(parSigs) < a.threshold { // @audit does not account for duplicates
        return errors.New("require threshold signatures")
    } else if a.threshold == 0 {
        return errors.New("invalid threshold config")
    }

    // Get all partial signatures.
    blsSigs := make(map[int]tblsv2.Signature)
    for _, parSig := range parSigs {
        sig, err := tblsconv2.SigFromCore(parSig.Signature())
        if err != nil {
            return errors.Wrap(err, "signature from core")
        }
        blsSigs[parSig.ShareIdx] = sig
    }

    // ...
```

The severity is raised as information as `parsigdb` module should not call `Aggregate()` with duplicate indexes.

Recommendations

To prevent the possibility of this issue occurring, consider enforcing `len(blsSigs) == a.threshold`.

Resolution

The recommended solution has been implemented in PR #2061.

OBOL-17	`sigagg` Does Not Verify the Reconstructed Signature
Asset	`core/sigagg/sigagg.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

After the partials have been used to reconstruct an aggregate signature there is a lack of validation to ensure the reconstructed signature is valid.

Within the function `Aggregate()`, if malformed partial signatures have been supplied, the reconstructed signature will be invalid. The invalid signature will be unnoticed and passed to `sigaggdb`.

Recommendations

Consider using the public key and message to validate the reconstructed signature is valid in `Aggregate()`.

Resolution

PR #2123 updates the code such that it will verify an aggregate signature when it is reconstructed.

OBOL-18	Aggregate Lock Only Collects `t` Signatures Rather Than `n`
Asset	`dkg/dkg.go`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `AggregateLock` signature is intended to be the summation of all signatures of each set. There is a limitation in the code when only aggregates `t` signatures from each validation set rather than the number of nodes.

```
391 peerSigs, err := ex.exchange(ctx, sigLock, lockHashSig)
    if err != nil {
393   return cluster.Lock{}, err
    }
395

397 pubkeyToShares := make(map[core.PubKey]share)
    for _, sh := range shares {
399   pk, err := core.PubKeyFromBytes(sh.PubKey[:])
      if err != nil {
401     return cluster.Lock{}, err
      }
403

405   pubkeyToShares[pk] = sh
    }
407

409 aggSigLockHash, aggPkLockHash, err := aggLockHashSig(peerSigs, pubkeyToShares, lock.LockHash)
```

Since `ex.exchange()` will only fetch `t` signatures, not all signatures and public keys are aggregated.

The result is `len(peerSigns[pk]) == t` when it should equal to the number of nodes.

The severity is rated as informational since with `t` signatures it is possible to reconstruct any other partial value. Hence, signatures from other parties can be calculated from the logs.

Recommendations

Modify the exchanger to fetch `n` signatures if the duty is for the lock signature.

Resolution

The issue is deemed invalid the return value of `len(peerSigns)` is in fact the number of validators rather than the threshold.

This is covered by line [**92-95**] in `exchanger.go::exchange()` which break when `e.numVals` signatures have been received.

OBOL-19	CreateDKG Allows the Threshold to be Larger Than the Number of Operators
Asset	`cmd/createdkg.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

There is a lack of sanity checks in the `createdkg` command to ensure the threshold is less than the number of parties.

It would not be possible to run a valid DKG if the threshold is larger than the number of parties as Lagrange interpolation will fail. However, this issue is not raised during the `createdkg` command which will execute successfully.

The severity is rated as information as this would be a configuration error and occur before the DKG is run. If the DKG is executed it would always fail.

Recommendations

Consider adding a sanity check to ensure threshold is less than the number of operators in the `createdkg` command.

Resolution

The recommendation was implemented in PR #2136 to ensure the threshold is not greater than the number of nodes.

OBOL-20	CreateDKG Does Not Validate the Checksum of the Withdrawal Addresses
Asset	`cmd/createdkg.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `validateWithdrawalAdds()` does not validate the checksum of the address is valid.

The return value of `eth2util.ChecksumAddress()` is the checksum address of the decoded hex string `addr`. Since the return value is dropped there are no checks to ensure `addr` is in checksum format.

The impact is malformed withdrawal addresses could be submitted as the withdrawal address for the validator. If a deposit is processed with an invalid withdrawal address the deposit will be permanently lost.

```
func validateWithdrawalAddrs(addrs []string, network string) error {
    for _, addr := range addrs {
        if _, err := eth2util.ChecksumAddress(addr); err != nil { //@audit consider checking the return value matches addr
            return errors.Wrap(err, "invalid withdrawal address", z.Str("addr", addr))
        }

        // We cannot allow a zero withdrawal address on mainnet or gnosis.
        if isMainNetwork(network) && addr == defaultWithdrawalAddr {
            return errors.New("zero address forbidden on this network", z.Str("network", network))
        }
    }

    return nil
}
```

Recommendations

Consider enforcing withdrawal addresses to be checksum addresses and validating them in `validateWithdrawalAddrs()`.

Resolution

Additional checks have been added in PR #2136 which ensure the supplied address is in checksum format.

OBOL-21	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:**
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Inconsistent use of "relay" vs "bootnode".**
   - relay: cmd/relay.go on line [**33**]
   - bootnode: docs/structure.md on line [**71**]
2. **Invalid comment about function name.**
   `tooxHex` is referenced however the function name is `fromoxHex()` in cluser/helpers.go line [**233**].
3. **Inconsistent variable names `tp` vs `tx` for `kcTransport`.**
   See dkg/keycast.go line [**55-92**].
4. **Spelling and grammar errors.**
   - Grammar cluster/definition.go on line [**247**].
   - *"enode"* in p2p/peer.go on line [**121**].
5. **Open TODOs with minimal security impact.**
   The following *TODOs* have been noted in the code as potential improvements but have minimal security impact.
   - cmd/createdkg.go#L130
   - core/parsigdb/memory.go#L94
   - app/peerinfo/peerinfo.go#L208-L209
   - app/app.go#L783
   - app/app.go#L855
   - app/app.go#L866

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing them where appropriate in PR #2181.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The native Go Lang test framework was used to perform these tests and fuzzers and the output is given below.

```
--- FAIL: FuzzDecodeBytes (0.00s)
    --- FAIL: FuzzDecodeBytes/0c4654b58f6d2016 (0.00s)
panic: runtime error: slice bounds out of range [:-7264253215423582151] [recovered]
  panic: runtime error: slice bounds out of range [:-7264253215423582151]

goroutine 78 [running]:
testing.tRunner.func1.2({0x15c73c0, 0xc00003c390})
  /usr/local/go/src/testing/testing.go:1526 +0x24e
testing.tRunner.func1()
  /usr/local/go/src/testing/testing.go:1529 +0x39f
panic({0x15c73c0, 0xc00003c390})
  /usr/local/go/src/runtime/panic.go:884 +0x213
github.com/obolnetwork/charon/eth2util/rlp.DecodeBytes({0xc000a14d50, 0xa, 0x10})
  /home/knaps/engagements/obol/obol-review/code/charon/eth2util/rlp/rlp.go:88 +0x1fb
github.com/obolnetwork/charon/eth2util/rlp_test.FuzzDecodeBytes.func1(0x0?, {0xc000a14d50?, 0x0?, 0x484719?})
  /home/knaps/engagements/obol/obol-review/code/charon/eth2util/rlp/rlp_test.go:61 +0x27
reflect.Value.call({0x14c6900?, 0x19d6d30?, 0x469756?}, {0x16467f5, 0x4}, {0xc0004cf2f0, 0x2, 0x2?})
  /usr/local/go/src/reflect/value.go:586 +0xb07
reflect.Value.Call({0x14c6900?, 0x19d6d30?, 0x1fcb828?}, {0xc0004cf2f0?, 0x163efe0?, 0xc000a14d90?})
  /usr/local/go/src/reflect/value.go:370 +0xbc
testing.(*F).Fuzz.func1.1(0xc0009031e0?)
  /usr/local/go/src/testing/fuzz.go:335 +0x3f3
testing.tRunner(0xc000903380, 0xc0003a8120)
  /usr/local/go/src/testing/testing.go:1576 +0x10b
created by testing.(*F).Fuzz.func1
  /usr/local/go/src/testing/fuzz.go:322 +0x5b9
exit status 2
FAIL  github.com/obolnetwork/charon/eth2util/rlp  0.020s

------------------------------------------------------------

--- FAIL: FuzzDecodeBytesList (0.00s)
    --- FAIL: FuzzDecodeBytesList/031b8be6dad7ca5f (0.00s)
panic: runtime error: slice bounds out of range [:-58493811630788525] [recovered]
  panic: runtime error: slice bounds out of range [:-58493811630788525]

goroutine 66 [running]:
testing.tRunner.func1.2({0x15c73c0, 0xc000b20048})
  /usr/local/go/src/testing/testing.go:1526 +0x24e
testing.tRunner.func1()
  /usr/local/go/src/testing/testing.go:1529 +0x39f
panic({0x15c73c0, 0xc000b20048})
  /usr/local/go/src/runtime/panic.go:884 +0x213
github.com/obolnetwork/charon/eth2util/rlp.DecodeBytesList({0xc000aa40c0, 0x24, 0x30})
  /home/knaps/engagements/obol/obol-review/code/charon/eth2util/rlp/rlp.go:54 +0x4e6
github.com/obolnetwork/charon/eth2util/rlp_test.FuzzDecodeBytesList.func1(0x0?, {0xc000aa40c0?, 0x0?, 0x484719?})
  /home/knaps/engagements/obol/obol-review/code/charon/eth2util/rlp/rlp_test.go:46 +0x27
reflect.Value.call({0x14c6900?, 0x19d6d10?, 0x469756?}, {0x16467f5, 0x4}, {0xc00086e1e0, 0x2, 0x2?})
  /usr/local/go/src/reflect/value.go:586 +0xb07
reflect.Value.Call({0x14c6900?, 0x19d6d10?, 0x1fcb758?}, {0xc00086e1e0?, 0x163efe0?, 0xc000874188?})
  /usr/local/go/src/reflect/value.go:370 +0xbc
testing.(*F).Fuzz.func1.1(0x0?)
  /usr/local/go/src/testing/fuzz.go:335 +0x3f3
testing.tRunner(0xc000a02680, 0xc000132090)
  /usr/local/go/src/testing/testing.go:1576 +0x10b
created by testing.(*F).Fuzz.func1
  /usr/local/go/src/testing/fuzz.go:322 +0x5b9
exit status 2
FAIL  github.com/obolnetwork/charon/eth2util/rlp  0.022s

------------------------------------------------------------
```

```
--- FAIL: FuzzParse (0.10s)
    --- FAIL: FuzzParse (0.00s)
        testing.go:1485: panic: runtime error: slice bounds out of range [:-8052150190496291]
            goroutine 56 [running]:
            runtime/debug.Stack()
              /usr/local/go/src/runtime/debug/stack.go:24 +0x9e
            testing.tRunner.func1()
              /usr/local/go/src/testing/testing.go:1485 +0x1f6
            panic({0x15c9bc0, 0xc00016e078})
              /usr/local/go/src/runtime/panic.go:884 +0x213
            github.com/obolnetwork/charon/eth2util/rlp.DecodeBytesList({0xc0001677d0, 0x25, 0x25})
              /home/knaps/engagements/obol/obol-review/code/charon/eth2util/rlp/rlp.go:54 +0x4e6
            github.com/obolnetwork/charon/eth2util/enr.Parse({0xc000983ac1, 0x36})
              /home/knaps/engagements/obol/obol-review/code/charon/eth2util/enr/enr.go:65 +0x23b
            github.com/obolnetwork/charon/eth2util/enr_test.FuzzParse.func1(0x0?, {0xc000983ac1?, 0x0?})
              /home/knaps/engagements/obol/obol-review/code/charon/eth2util/enr/enr_test.go:36 +0x25
            reflect.Value.call({0x14c96c0?, 0x19d9ab0?, 0x469756?}, {0x16491f7, 0x4}, {0xc000158ff0, 0x2, 0x2?})
              /usr/local/go/src/reflect/value.go:586 +0xb07
            reflect.Value.Call({0x14c96c0?, 0x19d9ab0?, 0x1fcfe20?}, {0xc000158ff0?, 0x16419e0?, 0xc00017bb60?})
              /usr/local/go/src/reflect/value.go:370 +0xbc
            testing.(*F).Fuzz.func1.1(0x0?)
              /usr/local/go/src/testing/fuzz.go:335 +0x3f3
            testing.tRunner(0xc0000de340, 0xc0001b4510)
              /usr/local/go/src/testing/testing.go:1576 +0x10b
            created by testing.(*F).Fuzz.func1
              /usr/local/go/src/testing/fuzz.go:322 +0x5b9
exit status 1
FAIL  github.com/obolnetwork/charon/eth2util/enr  0.182s

------------------------------------------------------------

--- FAIL: TestSamirShareFromProtoCrash (0.00s)
panic: runtime error: invalid memory address or nil pointer dereference [recovered]
  panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x100e2b1]

goroutine 7 [running]:
testing.tRunner.func1.2({0x11c4900, 0x2091ef0})
  /usr/local/go/src/testing/testing.go:1526 +0x24e
testing.tRunner.func1()
  /usr/local/go/src/testing/testing.go:1529 +0x39f
panic({0x11c4900, 0x2091ef0})
  /usr/local/go/src/runtime/panic.go:884 +0x213
github.com/obolnetwork/charon/dkg.keyFromProto(...)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:395
github.com/obolnetwork/charon/dkg.shamirShareFromProto(...)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:314
github.com/obolnetwork/charon/dkg.TestSamirShareFromProtoCrash(0x0?)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/keycast_internal_test.go:140 +0x71
testing.tRunner(0xc0009831e0, 0x16f6c00)
  /usr/local/go/src/testing/testing.go:1576 +0x10b
created by testing.(*T).Run
  /usr/local/go/src/testing/testing.go:1629 +0x3ea
exit status 2
FAIL  github.com/obolnetwork/charon/dkg  0.014s

------------------------------------------------------------

--- FAIL: TestRound1CastFromProtoCrash (0.00s)
panic: runtime error: invalid memory address or nil pointer dereference [recovered]
  panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x28 pc=0x1006d92]

goroutine 20 [running]:
testing.tRunner.func1.2({0x11c4900, 0x2091ef0})
  /usr/local/go/src/testing/testing.go:1526 +0x24e
testing.tRunner.func1()
  /usr/local/go/src/testing/testing.go:1529 +0x39f
panic({0x11c4900, 0x2091ef0})
  /usr/local/go/src/runtime/panic.go:884 +0x213
```

```
github.com/obolnetwork/charon/dkg.keyFromProto(...)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:395
github.com/obolnetwork/charon/dkg.round1CastFromProto(0xc0000e3580)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:354 +0x3b2
github.com/obolnetwork/charon/dkg.TestRound1CastFromProtoCrash(0x0?)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/keycast_internal_test.go:149 +0xbc
testing.tRunner(0xc0003af520, 0x16f6bf0)
  /usr/local/go/src/testing/testing.go:1576 +0x10b
created by testing.(*T).Run
  /usr/local/go/src/testing/testing.go:1629 +0x3ea
exit status 2
FAIL  github.com/obolnetwork/charon/dkg  0.020s

------------------------------------------------------------

--- FAIL: TestRound2CastFromProtoCrash (0.00s)
panic: runtime error: invalid memory address or nil pointer dereference [recovered]
  panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x28 pc=0x1007133]

goroutine 38 [running]:
testing.tRunner.func1.2({0x11c4900, 0x2091ef0})
  /usr/local/go/src/testing/testing.go:1526 +0x24e
testing.tRunner.func1()
  /usr/local/go/src/testing/testing.go:1529 +0x39f
panic({0x11c4900, 0x2091ef0})
  /usr/local/go/src/runtime/panic.go:884 +0x213
github.com/obolnetwork/charon/dkg.keyFromProto(...)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:395
github.com/obolnetwork/charon/dkg.round2CastFromProto(0xc00056a360)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/frostp2p.go:379 +0xd3
github.com/obolnetwork/charon/dkg.TestRound2CastFromProtoCrash(0x0?)
  /home/knaps/engagements/obol/obol-review/code/charon/dkg/keycast_internal_test.go:158 +0x88
testing.tRunner(0xc0000d7380, 0x16f6bf8)
  /usr/local/go/src/testing/testing.go:1576 +0x10b
created by testing.(*T).Run
  /usr/local/go/src/testing/testing.go:1629 +0x3ea
exit status 2
FAIL  github.com/obolnetwork/charon/dkg  0.019s
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

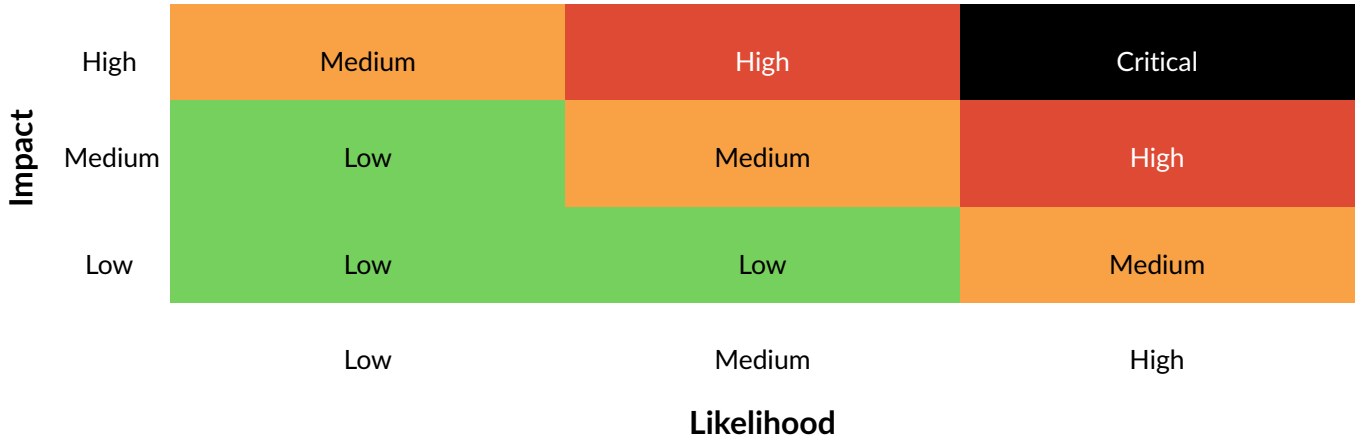

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References