Lido Finance

DC4BC Batch BLSToExecutionChange Signing

Security Assessment Report

Version: 2.0

March, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Lido Finance **dc4bc** (**D**istributed **C**ustody **f**or the **B**eacon **C**hain) software, with a particular focus on recent modifications that enable bulk signing `BLSToExecutionChange` messages. The review focused solely on the security aspects of the implementation, though general recommendations and informational comments are also provided.

Sigma Prime performed an initial security review of the Lido platform in a previous engagement (Q4 2020). An earlier version of dc4bc was included in the scope of review.

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 contract. 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 Lido Finance smart contracts 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*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Lido Finance smart contracts.

Overview

Lido is a liquid staking solution for the Ethereum proof-of-stake (PoS) consensus layer. It is built to improve the liquidity, composability and accessibility experience for those participating in consensus layer staking. Users can deposit *ETH* into a smart contract in order to mint a corresponding amount of *stETH*[1] in return. The ETH is deposited into the consensus layer in batches of 32 ether, to create validators that earn staking rewards. As an ERC20, stETH is easily integrated into the DeFi ecosystem, and can be redeemed for ETH in the Lido deposit pool. When validator withdrawals become active as part of the Capella hard fork[2], it is intended to be able to redeem stETH for any liquid ETH withdrawn by Lido validators.

The **dc4bc** (**D**istributed **C**ustody **for** the **B**eacon **C**hain) project allows trusted network participants (i.e. initial Lido DAO members) to perform a distributed key generation (DKG) ceremony using threshold signatures and sign infrequent messages. This software was used to generate the BLS `0x00` withdrawal credentials[3] used by the roughly $18,000$ Lido validators created prior to July 2021[2].

In order to enable withdrawals for each of these validators, a correctly signed `BLSToExecutionChange` message must be published that nominates an execution layer address as the validator's withdrawal recipient. For Lido,

[1] A derivative ERC20[1] token

[2] Refer to https://notes.ethereum.org/@launchpad/withdrawals-faq#Q-What-are-withdrawals.

[3] For explanation of type `0x00` credentials, refer to https://notes.ethereum.org/@launchpad/withdrawals-faq#Q-What-are-0x00-and-0x01-withdrawal-credentials-prefixes

using the original dc4bc to create $18,000$ signatures would involve a laborious manual procedure for every signature. At this scale, the process becomes intractable and prone to user-error. This review focuses on upgrades to dc4bc that enable signing batches of these signatures in bulk.

Security Assessment Summary

This review was conducted on the files hosted on the Lido Finance dc4bc repository and were assessed at commit 7a391fa. Retesting activities targeted commit 6e233ce.

The manual code review section of the report is focused on identifying any and all 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` meta-linter: `https://golangci-lint.run/` Including linters
  - errcheck: `https://github.com/kisielk/errcheck`
  - staticcheck: `https://staticcheck.io/`
  - gocritic: `https://github.com/go-critic/go-critic`
  - gosec: `https://github.com/securego/gosec`
  - revive: `https://github.com/mgechev/revive`
  - govet: `https://golang.org/cmd/vet`
- semgrep: `https://semgrep.dev/`, using ruleset from `https://github.com/dgryski/semgrep-go.git`.
- 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 8 issues during this assessment. Categorised by their severity:

- Medium: 3 issues.
- Low: 2 issues.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Lido Finance smart contracts. 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.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

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
LDC-01	Potential Index Out of Bounds Panic	Medium	Resolved
LDC-02	Kafka Does Not Handle JSON Errors Gracefully	Medium	Resolved
LDC-03	`reinitDKG` Can Be Called Multiple Times With The Same ID	Medium	Resolved
LDC-04	Outdated Prysm Dependency	Low	Resolved
LDC-05	Verification Script Fragile to Whitespace	Low	Resolved
LDC-06	Lock Not Released in Error Scenario	Informational	Resolved
LDC-07	Messages May Be Sent to a Single Node	Informational	Closed
LDC-08	Miscellaneous General Comments	Informational	Resolved

LDC-01	Potential Index Out of Bounds Panic		
Asset	`fsm/types/requests/signing_proposal.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

An index out of bounds panic may occur in the function `ReconstructBakedMessage()` if the `id` parameter is larger than the length of `validatorIDS`.

```go
func ReconstructBakedMessage(id int) (MessageToSign, error) {
  validatorsIDS := strings.Split(wc_rotation.ValidatorsIndexes, "\n")
  vID, err := strconv.ParseInt(validatorsIDS[id], 10, 64) //@audit this will panic if: id >= len(validatorIDS)
```

`wc_rotation.ValidatorIndexes` is predetermined in the file `payload.csv`.

To exploit this panic an attacker may send a `EventSigningStart` with a range that extends past $18,632$. The impact is that each of the node services will have an index out of bounds panic when they process the message in `processSignatureProposal()`.

Recommendations

To mitigate this issue ensure that `id` is less than `len(validatorIDS)`.

Resolution

A length check has been added in commit 44ddc81 which will prevent an index out of bounds panic.

LDC-02	Kafka Does Not Handle JSON Errors Gracefully		
Asset	`storage/kafka_storage/kafka_storage.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

If a Kafka message is created which has invalid JSON encoding the function `GetMessages()` will error without returning any messages.

```go
139 func (ks *KafkaStorage) GetMessages(_ uint64) ([]storage.Message, error) {
      ctx, cancel := context.WithDeadline(ks.readerCtx, time.Now().Add(ks.readDuration))
141   defer cancel()

143   var (
        message  storage.Message
145     messages []storage.Message
      )
147   for {
        kafkaMessage, err := ks.reader.ReadMessage(ctx)
149     if err != nil {
          if errors.Is(err, context.DeadlineExceeded) || errors.Is(err, context.Canceled) {
151         break
          } else {
153         return nil, fmt.Errorf("failed to ReadMessage: %w", err)
          }
155     }

157     if err = json.Unmarshal(kafkaMessage.Value, &message); err != nil {
          return nil, fmt.Errorf("failed to unmarshal a message %s: %w",
159         string(kafkaMessage.Value), err) //@audit returning here does not process previous / future messages
        }
161
        message.Offset = uint64(kafkaMessage.Offset)
163
        _, idOk := ks.idIgnoreList[message.ID]
165     _, offsetOk := ks.offsetIgnoreList[message.Offset]
        if !idOk && !offsetOk {
167       messages = append(messages, message)
        }
169   }

171   return messages, nil
    }
```

When a JSON encoding error occurs on line [157] the function immediately returns. Previously read `messages` are not included in the return value.

There is a censorship attack where a malicious node may send a message with invalid JSON encoding directly after the message they wish to censor. Since the previous messages will not be processed when there is an error the previous message is essentially censored. An attacker could repeatedly perform this attack to prevent any messages from being processed.

Recommendations

To resolve this issue, messages with invalid JSON encoding can be skipped rather returning an error.

Resolution

The issue is resolved in commit d50d7b4 by continuing past the error case if one arises.

LDC-03	`reinitDKG` Can Be Called Multiple Times With The Same ID		
Asset	`client/services/node/node_service.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

It is possible to send a message of type `ReinitDKG` which has the same DKG ID as the current round.

The most promiment issue that arises from sending a second Reinit DKG message is that it will update the communications key for each of the participants.

The following code snippet is taked from `reinitDKG()` which will be called if a malicious node submits a `ReinitDKG` message to the message storage. The malicious node may set `DKGID` to an arbitrary value, such as the current round ID.

```go
// save new comm keys into FSM to verify future messages
fsmInstance, err := s.fsmService.GetFSMInstance(req.DKGID, true)
if err != nil {
    return fmt.Errorf("failed to get FSM instance: %w", err)
}
for _, reqParticipant := range req.Participants {
    fsmInstance.FSMDump().Payload.SetPubKeyUsername(reqParticipant.Name, reqParticipant.NewCommPubKey)
}
fsmDump, err := fsmInstance.Dump()
if err != nil {
    return fmt.Errorf("failed to get FSM dump")
}

if err := s.fsmService.SaveFSM(message.DkgRoundID, fsmDump); err != nil {
    return fmt.Errorf("failed to SaveFSM: %w", err)
}
```

The impact of this issue is that all node will update the communication public key to the value set by the attacker. The attacker can then control all messages sent via the protocol.

There is a second issue that is related. The verification of signatures is disabled during `reinitDKG()` then operations are executed. Therefore, it is possible for a node to send malicious message acting as another party and have them processed. This is done similarly to the previous attack by calling `reinitDKG()` setting `req.DKGID` as the current round.

```
// temporarily fix cause we can't verify patch messages
// TODO: remove later
if !s.GetSkipCommKeysVerification() {
    s.SetSkipCommKeysVerification(true)
    defer s.SetSkipCommKeysVerification(false)
}

operations := make([]*types.Operation, 0)
for _, msg := range req.Messages {
    if fsm.Event(msg.Event) == sif.EventSigningStart {
        break
    }
    if msg.RecipientAddr == "" || msg.RecipientAddr == s.GetUsername() {
        operation, err := s.processMessage(msg)
        if err != nil {
            s.Logger.Log("failed to process operation:  %w", err)
        }
        if operation != nil {
            operations = append(operations, operation)
        }
    }
}
```

Recommendations

To resolve this issue prevent `reinitDKG()` from being called if `req.DKGID` is already in use.

Resolution

Commit 4e106cc adds an additional check to `reinitDKG()` to ensure the ID has not already been used.

LDC-04	Outdated Prysm Dependency		
Asset	`go.mod`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

dc4bc currently depends on the module `github.com/prysmaticlabs/prysm v1.4.2`. This was released on 23 July 2021, and is severely out of date, with the most recent release being `v3.2.1`.

As this module is currently only used for compatibility testing and the relevant code is relatively stable, it poses no known *direct* risk to the operation of dc4bc. However, as it may allow for unnoticed bugs or outdated transitive dependencies, the risk is still deemed *low* (due to the crucial nature of the software).

Recommendations

The testing team recommends replacing the existing `go.mod` entry for `github.com/prysmaticlabs/prysm v1.4.2-0.2022062813032` with `github.com/prysmaticlabs/prysm/v3`. This would involve updating the corresponding Go files where the module is used, such as `pkg/prysm/prysm.go`.

The latest release can be added as a dependency via

```
> go get github.com/prysmaticlabs/prysm/v3@latest
```

Resolution

A resolution to this issue can be seen in commit c1a2f33, where the dependency has been updated to version v3.2.1.

LDC-05	Verification Script Fragile to Whitespace		
Asset	`pkg/wc_rotation/payload_csv_test.sh`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The script, `payload_csv_test.sh`, fails to adequately handle whitespace present in the beacon node's JSON response. There are also several bash best practices that are advised.

Although the JSON fields returned by the API are specified to contain no whitespace,[4] valid JSON may contain arbitrary whitespace *between* fields. The Beacon API specification makes no requirement that JSON responses must be *compact*, so a compliant node's response may contain whitespace.

The `payload_csv_test.sh` script is purposed to help signing members verify the content of `payload.csv` file. If the script were to fail, the primary impact to hinder users' ability and confidence in verifying the payload to be signed.

Detail

Consider the following excerpt of `payload_csv_test.sh`:

```
16  RESPONSE=($(curl --request GET --url "$URL" --header 'Content-Type: application/json'))

18  echo "Validating payloads.csv"

20  JQ_FILTER=".data[] | select(.validator.withdrawal_credentials == \"$WITHDRAWAL_CREDENTIALS\") | .index | tonumber"
    SORTED_VALIDATORS=$(jq -r "$JQ_FILTER" <<< $RESPONSE | jq -s | jq '.|sort')
22
    jq -r '.[]' <<< "$SORTED_VALIDATORS" > actual.csv
```

At line [16], the syntax `VAR=(value1 ... valuen)` is actually a compound array assignment in bash script, with whitespace separated *words* assigned to each index. It is likely that `RESPONSE` is not intended as an array variable, as it is never referenced using explicit array syntax (like `${RESPONSE[0]}`).

According to the bash manual[3], *"Referencing an array variable without a subscript is equivalent to referencing the array with a subscript of 0."* Because of this, the data actually passed to `jq` at line [21] is only the first *word* of the curl output.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions below:

1. Remove the array assignment at line [16] like so:

   ```
   RESPONSE=$(curl --request GET --url "$URL" --header 'Content-Type: application/json')
   ```

[4] Refer to https://ethereum.github.io/beacon-APIs/#/Beacon/getStateValidators

2. Recommend enclosing `$RESPONSE` at line [**21**] in double quotes, as the *herestring* syntax is only documented to accept a single *word* input. While the current syntax is reportedly ok for Bash versions starting from 4, the default Bash version for Mac OSX is still 3.2.
3. Consider handling a failure retrieving the data from the beacon API, with a curl option `--fail` to set a nonzero exit code in the event of an > 400 http status
4. At line [**29**] (shown below) prefer the more robust bash `[[` conditional to the posixcompatible `[` test.

   ```
   if [ "$DIFF" != "" ]
   ```

5. The testing team recommends using the shellcheck tool to identify common shell script pitfalls.
6. Test the modified script to confirm that it operates correctly with responses containing whitespace.

Resolution

Commits 43a3a2c and 4c851ce resolve the issue.

LDC-06	Lock Not Released in Error Scenario
Asset	`cmd/airgapped/main.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

In the `(*prompt).run()` method, it is possible to return at line [464] without releasing the `p.airgapped` Mutex.

In this case, an error result will cause the program to exit so there is no problems with halting due to lock contention. As such, this issue poses no known security risk to the current codebase, and is deemed *informational*.

Recommendations

Ensure the `p.airgapped` Mutex is unlocked in all scenarios.

It is generally preferable and less prone to error to unlock via the `defer` functionality. This can be done by splitting lines [452-468] into their own function.

Resolution

This issue is resolved in commit 961f0e2 by releasing the lock for all error cases.

LDC-07	Messages May Be Sent to a Single Node
Asset	`client/services/node/node_service.go`
Status	**Closed:** See Resolution
Rating	Informational

Description

Messages can be sent directly to a single node. A malicious node may send different messages to each node in order to set the FSM of each node to a different state.

The following code snippet shows if `msg.ReceipientAddress` is the username of the current or is empty, then the node will process the message.

```go
for _, msg := range req.Messages {
  if fsm.Event(msg.Event) == sif.EventSigningStart {
    break
  }
  if msg.RecipientAddr == "" || msg.RecipientAddr == s.GetUsername() {
    operation, err := s.processMessage(msg)
    if err != nil {
      s.Logger.Log("failed to process operation:  %w", err)
    }
    if operation != nil {
      operations = append(operations, operation)
    }
  }
}
```

A possible exploit of this situation is to send different signing proposal messages to each node. The nodes will each be in a state where they are awaiting partial signatures from other nodes. However, these will not occur as the other nodes have not processed the initial message for the required proposal.

Recommendations

Consider remove the ability to send individual message and instead enforcing the message server to act as broadcast only.

This will have the drawback that it is more challenging to patch a single node if it falls out of sync.

Resolution

The development team have opted not to fix this issue. Removing the ability to send messages to individual users will break backwards compatibility with `v0.1.4`. As it is possible to recover quickly from this kind of attack by restarting each node service, the risk is deemed acceptable.

For more details see commit 6e233ce.

LDC-08	Miscellaneous General Comments
Asset	All files in scope of review
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **Inconsistent error wrapping:**
   There is inconsistent use of the preferred `%w` format string when rendering errors in `fmt.Errorf()`, and often the non-wrapping verb is instead used `%v`. This is of minor importance in the current use-case, which intends for the errors to be displayed to a user and would not need to be handled programmatically. However, using `%w` to wrap errors is a general best practice for which it is recommended to abide.

   For example, `cmd/dc4bc_cli/main.go` contains a combination of `%w` and `%w` in its `fmt.Errorf()` calls:

   - `%v` is used at lines:
     129, 142, 253, 288, 313, 318, 323, 395, 408, 435, 465, 478, 529, 561, 573, 597, 629, 663, 690, 703, 734, 754, 777, 784, 868, 884, 908, 967, 1021, 1034, 1126, 1153
   - `%w` is used at lines:
     119, 124, 147, 230, 235, 240, 258, 278, 283, 329, 356, 363, 368, 373, 386, 391, 412, 439, 455, 460, 483, 489, 499, 506, 511, 536, 541, 551, 556, 578, 602, 607, 611, 634, 648, 668, 680, 685, 708, 714, 739, 743, 759, 788, 807, 811, 848, 873, 890, 913, 923, 941, 947, 972, 977, 981, 992, 998, 1011, 1016, 1039, 1071, 1104, 1109, 1131, 1159, 1181, 1187

   This can be identified automatically with the errorlint tool.

2. **Identified Miscellaneous Optimisations:**

   - At `pkg/wc_rotation/payload_csv_test.sh`, consider lines [**20-21**]

     ```
     JQ_FILTER=".data[] | select(.validator.withdrawal_credentials == \"$WITHDRAWAL_CREDENTIALS\") | .index | tonumber"
     SORTED_VALIDATORS=$(jq -r "$JQ_FILTER" <<< $RESPONSE | jq -s | jq '.|sort')
     ```

     The additional `jq` commands can be condensed as follows:

     ```
     JQ_FILTER="[.data[] | select(.validator.withdrawal_credentials == \"$WITHDRAWAL_CREDENTIALS\") | .index | tonumber] |
         ↪  sort"
     SORTED_VALIDATORS=$(jq -r "$JQ_FILTER" <<< $RESPONSE)
     ```

3. **Identified Typographical Errors:**

   - The file named `HowToSignBacked.md` should be `HowToSignBaked.md`.
   - At `storage/kafka_storage/kafka_storage.go:76` — "tsl" should be "TLS".
   - At `client/services/node/node_service.go:561` "calculat" should be "calculate".

4. **Unnecessary if statements:** In `pkg/wc_rotation/rotation.go` the function parameters are either `[4]byte` or `[32]byte`. The length of these arrays are fixed to 4 and 32 respectively, therefore they will never have length zero.

   Hence, the following two checks can never be triggered.

```
func computeDomain(domainType [4]byte, forkVersion [4]byte, genesisValidatorsRoot [32]byte) ([32]byte, error) {
  if len(forkVersion[:]) == 0 { // @audit caanot be triggered
    forkVersion = GenesisForkVersion
  }

  if len(genesisValidatorsRoot[:]) == 0 { // @audit canot be triggered
    genesisValidatorsRoot = [32]byte{}
  }
```

Recommendations

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

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in commit 665af4b, where appropriate.

Appendix A 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.

	High	Medium	High	Critical
Impact	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

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

References

[1] ERC-20 Token Standard. Github, Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[2] Lido. Withdrawal Credentials in Lido. Blog, July 2021, Available: https://blog.lido.fi/withdrawal-credentials-in-lido/. [Accessed 2023].

[3] Brian Fox and Chet Ramey. BASH(1) Manual Page, September 2022, Available: https://tiswww.case.edu/php/chet/bash/bash.html. [Accessed 2023].

σ'