Stader Labs

ETHx — Stader Permissioned CLI

Permissioned Operator Software Security Assessment Report

Version: 2.0

July, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Stader Labs off-chain software for permissionless node operators. The review focused solely on the security aspects of the codebase, 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 codebase. 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 Stader Labs 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*.

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

Overview

ETHx staking is a liquid ETH staking protocol that is permissionless and non-custodial. It enables users to participate in proof-of-stake validation with amounts smaller than the 32 ETH validator requirement by staking alongside other users.

This review covers the additional features added to the `stader-permissioned-cli` repository from `stader-node`. The core difference is in the `stader-cli validator register` command and `Web3Signer` interactions. This command prepares the pre-deposit signature and deposit signature for a permissioned Node Operator (NO).

A notable difference between the permissioned and permissionless setup is that permissioned NOs use a `Web3Signer` HTTP end point to request signatures for specific validtor keys. This differs to permissionless NOs who store the validator keys locally.

Security Assessment Summary

This review was conducted on the files hosted on the Stader Permissioned CLI which were assessed at commit 4305991. Additionally, a separate review of the permissionless CLI was also performed prior to this engagement and results were shared in a separate report.

Note: *native Go and Go Ethereum libraries and dependencies were excluded from the primary focus of this assessment. As were the various execution and consensus layer software implementations.*

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 codebase. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Go runtime and Ethereum protocol.

The manual code review focused on critical and sensitive features.

Testing methodology included:

- Identifying all areas consuming user supplied input;
- Manually attempting to exploit vulnerabilities through bypassing validations or manipulation of application business logic;
- Using semi-automated assessment and fault injection tools as appropriate;
- Reviewing private key access permissions;
- Ensuring appropriate logging is enforced;
- Reviewing authentication (user/service accounts), authorisation (access control), and transport encryption (SSL/TLS implementation);

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`.

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 5 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Stader Labs permissioned CLI codebase. 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 software that do not constitute a known security risk 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
SPC-01	Lack Of Verification Of Deposit Signatures	Low	Resolved
SPC-02	`Web3Signer` Does Not Use Authentication Over HTTP	Informational	Closed
SPC-03	`registerValidators()` Does Not Check For Invalid Amount	Informational	Resolved
SPC-04	Duplicate Packages In Stader Node & Stader Permissioned CLI	Informational	Closed
SPC-05	Miscellaneous General Comments	Informational	Closed

SPC-01	Lack Of Verification Of Deposit Signatures		
Asset	`stader/api/validator/register.go`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

There is no verification on the deposit signatures returned by the `Web3Signer`.

Signing on behalf of a validator is outsourced to the `Web3Signer` through an HTTP connection. There are no checks to ensure the deposit signatures returned by the `Web3Signer` are valid signatures.

If the `Web3Signer` returns invalid signatures without an error then `addValidatorKeys()` will be called with an invalid signature for `_preDepositSignature` or `_depositSignature`. If the deposit contract is called with an invalid signature the attached ETH will be unrecoverable.

Note if `_preDepositSignature` is invalid then `transferETHOfDefectiveKeysToSSPM()` is called and the remaining deposit is cancelled. Similarly, if `_depositSignature` is invalid then `transferETHOfDefectiveKeysToSSPM()` should be called before the final deposit is made.

Recommendations

Verify the signatures returned by `Web3Signer` to ensure they are valid.

Resolution

The issue has been resolved by verifying signatures received from the `Web3Signer` before propagating the transactions on-chain. Commit b8aaf83 contains the changes.

SPC-02	`Web3Signer` Does Not Use Authentication Over HTTP
Asset	`shared/services/web3signer-manager.go`
Status	**Closed:** See Resolution
Rating	Informational

Description

This issue is rated as informational as it was deemed out of scope, however it is an integral part of the reviewed components and thus included in this report.

There is no authentication used over the HTTP connection between the `Web3Signer` and permissioned Node Operator (NO).

The `Web3Signer` is responsible for signing messages using the validator private key. Permissioned NOs must use the validator key and therefore the `Web3Signer` to sign a deposit including the withdrawal address and amount.

For security reasons permissioned NOs must perform two deposits. First, a pre-deposit, which is a deposit including the required withdrawal credentials, using 1 ETH from the Stader on-chain pool. Second, the remaining deposit, which is 31 ETH from the Stader on-chain pool. The second deposit is executed after a delay, to ensure the first deposit has not been front-run with malicious withdrawal credentials.

A lack of authentication over the `Web3Signer` may allow a malicious actor to call the endpoint `/api/v1/eth2/sign/` with a deposit that contains malicious withdrawal credentials. The attacker would be able to exploit Stader permissioned pool by front running the pre-deposit with their own deposit.

Withdrawal credentials are taken from the first deposit with a valid signature, which in this case would be the malicious user's deposit. Therefore, the malicious user will be able to claim the 1 ETH amount from the pre-deposit.

Recommendations

Authentication must be used for HTTP connections in the `Web3Signer` to prevent a malicious actor from signing deposits. When using HTTP authentication, the HTTP messages must also be encrypted to prevent eavesdropping and replay attacks, this may be achieved by using SSL / TLS.

Resolution

The issue is partially resolved in commit dfea738 which enforces HTTPS to be used rather than HTTP.

Authentication has not been added to the signer. The onus of access control and authentication of the `Web3Signer` falls outside the scope of the Stader Permissioned CLI. It is left to Node Operators to add their own access control for the `Web3Signer`.

SPC-03	`registerValidators()` Does Not Check For Invalid Amount
Asset	`stader-cli/validator/register.go`
Status	**Resolved:** See Resolution
Rating	Informational

Description

There are insufficient checks in the function `registerValidators()` to cover all edge cases for the struct `CanRegisterValidatorsResponse`. Specifically, the field `InvalidAmount` is not checked for in `registerValidators()`.

The following is an excerpt from `registerValidators()`.

```
canRegisterValidatorsResponse, err := staderClient.CanRegisterValidators(validatorPubKeyList)
if err != nil {
  return err
}
if canRegisterValidatorsResponse.InsufficientBalance {
  fmt.Printf("Account does not have enough balance!")
  return nil
}
if canRegisterValidatorsResponse.DepositPaused {
  fmt.Printf("Deposits are currently paused!")
  return nil
}
if canRegisterValidatorsResponse.MaxValidatorLimitReached {
  fmt.Printf("Max validator limit reached")
  return nil
}
if canRegisterValidatorsResponse.OperatorNotRegistered {
  fmt.Printf("Operator not registered")
  return nil
}
if canRegisterValidatorsResponse.OperatorNotActive {
  fmt.Printf("Operator not active")
  return nil
}
if canRegisterValidatorsResponse.InputKeyLimitReached {
  fmt.Printf("You cannot add more than %d keys at a time", canRegisterValidatorsResponse.InputKeyLimit)
  return nil
}
```

Note this issue is raised as informational as the `InvalidAmount` error case should never arise, since the amounts are fixed for permissioned node operators.

Recommendations

To resolve this issue consider removing the field `InvalidAmount` from `CanRegisterValidatorsResponse`. Alternatively, add a check for `InvalidAmount` amount in `registerValidators()`.

Resolution

`InvalidAmount` has been removed in commit c6ee5eb.

SPC-04	Duplicate Packages In Stader Node & Stader Permissioned CLI
Asset	`stader-cli/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

Many packages are duplicated with identical code in the repositories stader-node and stader-permissioned-cli.

These include many of the files in the directories `stader-lib` and `shared`. Maintaining duplicate code bases adds significant overheads and may result in bug patches being implemented in one repository but not the other.

Recommendations

One solution is to merge the two repositories and have separate binaries for the permissioned and permissionless CLI tools.

An alternate solution is to extract the duplicate code and put them in their own repository which can be called on by the two CLI tools.

Resolution

The development team intend to abstract out common libraries. However, this will not be achieved in the short term and is marked as future work.

SPC-05	Miscellaneous General Comments
Asset	`stader-node/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

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

1. **Use constants over hard coded values.**

   In `stader/api/validator/register.go`, line [173] and line [298] the parameter of `poolType` is hardcoded to `2`.

   ```
   rewardWithdrawVault, err := node.ComputeWithdrawVaultAddress(vfc, 2, operatorId, newValidatorKey, nil)
   ```

   In the same file on line [183], line [187], line [308] and line [312] use constants rather than hardcoded amounts for pre-deposit amount and deposit amount.

   Consider replacing these values with constants.

2. **Lack of check before indexing an array.**

   This should not be reachable unless `Web3Signer` is sending us malformed data.

   In `stader/api/validator/register.go`, on line [81] if `len(val)` is less than two it will panic.

   ```
   types.HexToValidatorPubkey(val[2:])
   ```

   Similarly, in the same file on line [192], line [197], line [202], line [317], line [322] and line [327]. The same issue is also present in a range of files.

   Consider updating the function `types.HexToValidatorPubkey()` to take a string of arbitrary length and strip the first two characters if they are `0x`. Then decode the remaining hex string.

3. **Typos, spelling and grammar.**

   In `stader-cli/node/commands.go` on line [92] *"Send ETH or SD, EthX tokens from the node account to an address."* should say *"Send ETH, SD or ETHx ..."*.

4. **ETHx vs Ethx**

   There are multiple occurrences of ETHx and Ethx used in the code base. For consistency just use ETHx.

5. **Return error messages should be updated.**

   The error messages for `can-send-presigned-msg` and `send-presigned-msg` should be changed from `exit validator`.

   In `shared/services/stader/node.go`

```go
func (c *Client) CanSendPresignedMessage(validatorPubKey types.ValidatorPubkey) (api.CanSendPresignedMsgResponse, error) {
  responseBytes, err := c.callAPI(fmt.Sprintf("node can-send-presigned-msg %s", validatorPubKey))
  if err != nil {
    return api.CanSendPresignedMsgResponse{}, fmt.Errorf("could not get exit validator status: %w", err)
  }
  var response api.CanSendPresignedMsgResponse
  if err := json.Unmarshal(responseBytes, &response); err != nil {
    return api.CanSendPresignedMsgResponse{}, fmt.Errorf("could not decode exit validator response: %w", err)
  }
  if response.Error != "" {
    return api.CanSendPresignedMsgResponse{}, fmt.Errorf("could not get exit validator status: %s", response.Error)
  }
  return response, nil
}

func (c *Client) SendPresignedMessage(validatorPubKey types.ValidatorPubkey) (api.SendPresignedMsgResponse, error) {
  responseBytes, err := c.callAPI(fmt.Sprintf("node send-presigned-msg %s", validatorPubKey))
  if err != nil {
    return api.SendPresignedMsgResponse{}, fmt.Errorf("could not get exit validator status: %w", err)
  }
  var response api.SendPresignedMsgResponse
  if err := json.Unmarshal(responseBytes, &response); err != nil {
    return api.SendPresignedMsgResponse{}, fmt.Errorf("could not decode exit validator response: %w", err)
  }
  if response.Error != "" {
    return api.SendPresignedMsgResponse{}, fmt.Errorf("could not get exit validator status: %s", response.Error)
  }
  return response, nil
}
```

6. **Dangling Comments**

   In `shared/services/web3signer/web3signer-client/std-http-client.go` line [163] there is a commented out partially written function, which should be removed.

   ```go
   //func (c *StandardHttpClient)
   ```

   Similarly, remove the following debug print lines.

   ```
   % I know this is not got but it was having issues formatting otherwise
     stader/api/validator/send-cl-rewards.go
     61:  //fmt.Println("getting validator eth balance")

     stader/api/node/status.go
     139:  //fmt.Printf("Getting operator info...\n")

     stader/api/validator/exit-validator.go
     115:  //fmt.Printf("hex version of pub key is %s\n", validatorPubKey.Hex())

     shared/utils/stader/pre-signed-flows.go
     92:  //fmt.Printf("Debug: bulk presign check response is %s\n", string(body))
     94:  //fmt.Printf("res.body is %s\n", res)

     shared/utils/eth2/eth2.go
     32:  //fmt.Printf("Validator status: %v", validatorStatus)
   ```

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 commit 940320f.

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.

Impact			
High	Medium	High	Critical
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.