# StaderLabs - Oracle

## Golang Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 29th, 2023 - June 1st, 2023**

Visit: **Halborn.com**

## DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Document Creation | 05/30/2023 | Alejandro Taibo |
| 0.2 | Document Updates | 06/01/2023 | Alejandro Taibo |
| 0.3 | Document Updates | 06/01/2023 | Gokberk Gulgun |
| 0.4 | Draft Review | 06/02/2023 | Gokberk Gulgun |
| 0.5 | Draft Review | 06/05/2023 | Gabi Urrutia |
| 1.0 | Remediation Plan | 06/20/2023 | Alejandro Taibo |
| 1.1 | Remediation Plan Review | 06/20/2023 | Gokberk Gulgun |
| 1.2 | Remediation Plan Review | 06/20/2023 | Gabi Urrutia |

## CONTACTS

| CONTACT | COMPANY | EMAIL |
|---|---|---|
| Rob Behnke | Halborn | Rob.Behnke@halborn.com |
| Steven Walbroehl | Halborn | Steven.Walbroehl@halborn.com |
| Gabi Urrutia | Halborn | Gabi.Urrutia@halborn.com |
| Gokberk Gulgun | Halborn | Gokberk.Gulgun@halborn.com |
| Alejandro Taibo | Halborn | Alejandro.Taibo@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

StaderLabs engaged Halborn to conduct a security audit on their oracle repositories beginning on May 29th, 2023 and ending on June 1st, 2023. The security audit was scoped to the repositories provided to the Halborn team.

## 1.2 AUDIT SUMMARY

The team at Halborn was provided four days for the engagement and assigned two full-time security engineers to audit the security of the **oracle** implementation. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that oracle Implementation functions as intended.
- Identify potential security issues with the oracle.

In summary, Halborn identified some security risks that were mostly addressed by the StaderLabs team.

## 1.3 SCOPE

**IN-SCOPE CODE & COMMIT:**

- Repository: stader-guardian
  - Commit ID: b02bb9cf4a2863b00c16e629535c6ab96777af49

**REMEDIATION COMMIT:**

- Repository: stader-guardian
  - Commit ID: d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the custom modules. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of structures and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Static Analysis of security for scoped repository, and imported functions. (e.g., staticcheck, gosec, unconvert, codeql, ineffassign and semgrep)
- Manual Assessment for discovering security vulnerabilities on codebase.
- Ensuring correctness of the codebase.
- Dynamic Analysis on files and modules related to the project.
- Custom fuzz testing using Go's built-in fuzzing tools.

## 2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

## 2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

| Exploitability Metric ($m_E$) | Metric Value | Numerical Value |
|---|---|---|
| Attack Origin (AO) | Arbitrary (AO:A) | 1 |
| | Specific (AO:S) | 0.2 |
| Attack Cost (AC) | Low (AC:L) | 1 |
| | Medium (AC:M) | 0.67 |
| | High (AC:H) | 0.33 |
| Attack Complexity (AX) | Low (AX:L) | 1 |
| | Medium (AX:M) | 0.67 |
| | High (AX:H) | 0.33 |

Exploitability $E$ is calculated using the following formula:

$$E = \prod m_e$$

## 2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

| Impact Metric $(m_I)$ | Metric Value | Numerical Value |
|---|---|---|
| Confidentiality (C) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Integrity (I) | None (I:N) | 0 |
| | Low (I:L) | 0.25 |
| | Medium (I:M) | 0.5 |
| | High (I:H) | 0.75 |
| | Critical (I:C) | 1 |
| Availability (A) | None (A:N) | 0 |
| | Low (A:L) | 0.25 |
| | Medium (A:M) | 0.5 |
| | High (A:H) | 0.75 |
| | Critical | 1 |
| Deposit (D) | None (D:N) | 0 |
| | Low (D:L) | 0.25 |
| | Medium (D:M) | 0.5 |
| | High (D:H) | 0.75 |
| | Critical (D:C) | 1 |
| Yield (Y) | None (Y:N) | 0 |
| | Low (Y:L) | 0.25 |
| | Medium: (Y:M) | 0.5 |
| | High: (Y:H) | 0.75 |
| | Critical (Y:H) | 1 |

Impact $I$ is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

## 2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

| Coefficient ($C$) | Coefficient Value | Numerical Value |
|---|---|---|
| Reversibility ($r$) | None (R:N) | 1 |
| | Partial (R:P) | 0.5 |
| | Full (R:F) | 0.25 |
| Scope ($s$) | Changed (S:C) | 1.25 |
| | Unchanged (S:U) | 1 |

Severity Coefficient $C$ is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score $S$ is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

| Severity | Score Value Range |
|---|---|
| Critical | 9 - 10 |
| High | 7 - 8.9 |
| Medium | 4.5 - 6.9 |
| Low | 2 - 4.4 |
| Informational | 0 - 1.9 |

## 3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 1 | 0 | 3 | 8 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| UNHANDLED COINGECKO RESPONSES CAN SET UNDESIRED PRICES ON CHAIN | High | SOLVED - 06/20/2023 |
| ORACLE PRICE FEEDER IS VULNERABLE TO MANIPULATION BY A SINGLE MALICIOUS FEED | Low (4.2) | RISK ACCEPTED |
| PRIVATE KEY IN THE ENVIRONMENT VARIABLES | Low (2.5) | RISK ACCEPTED |
| DOCKER IMAGE RUNNING AS ROOT | Low (3.8) | SOLVED - 06/20/2023 |
| MAINNET CONFIGURATION ADDRESS IS UNDEFINED IN STADERCONFIGADDRESSSTR | Informational (0.0) | SOLVED - 06/20/2023 |
| LACK OF EXTENSIVE TEST COVERAGE | Informational (0.0) | PARTIALLY SOLVED - 06/20/2023 |
| POTENTIAL HTTP REQUEST TIMEOUT MISSING IN HTTP CLIENT CONFIGURATION | Informational (0.0) | ACKNOWLEDGED |
| ABSENCE OF CHECK FOR whenNotPaused CONDITION IN GOLANG CODEBASE | Informational (0.0) | ACKNOWLEDGED |
| ABI INCOMPATIBILITY WITH RECENT CODEBASE IN GO FILES | Informational (0.0) | SOLVED - 06/20/2023 |
| HARDCODED URL IN THE FUNCTIONS | Informational (0.0) | ACKNOWLEDGED |
| LACK OF HTTP RESPONSE STATUS CODE CHECK IN THE FUNCTIONS | Informational (0.0) | SOLVED - 06/20/2023 |
| LACK OF ERROR HANDLING FOR NON-EXISTENT CHAINID IN ConvertTimestampToSlotAndEpoch FUNCTION | Informational (0.0) | ACKNOWLEDGED |

# FINDINGS & TECH DETAILS

## 4.1 (HAL-01) UNHANDLED COINGECKO RESPONSES CAN SET UNDESIRED PRICES ON CHAIN - HIGH

Description:

The program makes use of Coingecko API in order to fetch prices, sending this data to StaderOracle smart contract to perform some operations on-chain over these values. Once prices are fetched in a specific range of time, the program calculates the average of all fetched prices. This average will act as a simple TWAP since all received prices are supposed to be gathered during a period of time specified by parameters in the endpoint request. These prices come wrapped in the following structure:

**Listing 1**

```
type CoingeckoResponse struct {
    Prices       [][]float64 `json:"prices"`
    MarketCaps   [][]float64 `json:"market_caps"`
    TotalVolumes [][]float64 `json:"total_volumes"`
}
```

However, this endpoint under some circumstances can return an empty struct whose Prices array is also empty. Then, during the average calculation, the program iterates over the Prices array in order to add each value into a single variable, since there are no prices in the struct, the averagePrice variable will be 0.0. Just after this loop, this variable is divided by then length of Prices array, which is 0 due this array is empty. Naturally, it's reasonable to think this operation will raise an error, but Golang returns NaN value in these kinds of operations. So, this function will end up returning NaN as averagePrice that will impact directly to the on-chain prices.

This returned NaN value won't raise any errors after its function call returns, but rather it will be converted to a huge number since this value is used as an argument in Float64ToUint256() function. This function always returns -9223372036854775808 number for NaN arguments, which is later submitted to the smart contract, setting a price of 115792089237316195...876274864128 in the StaderOracle contract.

Code Location:

Listing 2: external/coingecko.go (Lines 31,37,40)

```go
17 func FetchPriceFromCoingecko(from int64, to int64) (float64, error) {
18     url := fmt.Sprintf("https://api.coingecko.com/api/v3/coins/stader/market_chart/range?vs_currency=eth&from=%d&to=%d", from, to)
19     response, err := http.Get(url)
20     if err != nil {
21         return 0, fmt.Errorf("error fetching data from Coingecko: %w", err)
22     }
23     defer response.Body.Close()
24 
25     body, err := io.ReadAll(response.Body)
26     if err != nil {
27         return 0, fmt.Errorf("error reading response body: %w", err)
28     }
29 
30     var coingeckoResponse CoingeckoResponse
31     err = json.Unmarshal(body, &coingeckoResponse)
32     if err != nil {
33         return 0, fmt.Errorf("error unmarshalling response: %w", err)
34     }
35 
36     averagePrice := 0.0
37     for _, price := range coingeckoResponse.Prices {
38         averagePrice += price[1]
39     }
40     averagePrice /= float64(len(coingeckoResponse.Prices))
41 
42     return averagePrice, nil
43 }
```

Listing 3: cmd/sdPrice/task.go (Lines 42,48,61)

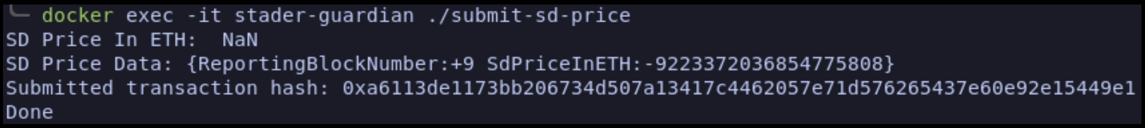
```go
42      sdPrice, err := external.FetchPriceFromCoingecko(
 ↳ blockTimestamp.Add(-24*time.Hour).Unix(), blockTimestamp.Unix())
43      if err != nil {
44          return fmt.Errorf("failed to get SD price: %w", err)
45      }
46      fmt.Println("SD Price In ETH: ", sdPrice)
47
48      if sdPrice == 0 {
49          fmt.Println("SD Price is 0, not submitting")
50          return nil
51      }
52
53      auth, err := bind.NewKeyedTransactorWithChainID(context.Config
 ↳ .PrivateKey, new(big.Int).SetUint64(uint64(context.Config.ChainId)
 ↳ ))
54      if err != nil {
55          return fmt.Errorf("failed to create transactor: %w", err)
56      }
57
58      sdPriceData := contracts.SDPriceData{ReportingBlockNumber:
 ↳ blockNumber, SdPriceInETH: Float64ToUint256(sdPrice)}
59      fmt.Printf("SD Price Data: %+v\n", sdPriceData)
60
61      tx, err := staderOracle.SubmitSDPrice(auth, sdPriceData)
```

Risk Level:

**Likelihood - 3**
**Impact - 5**

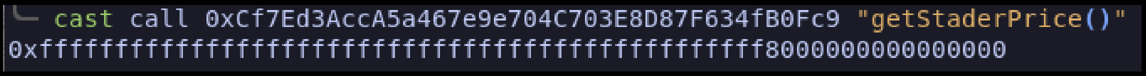
Code Location:

After executing the program and receiving an empty struct as response from CoinGecko, the execution prints the following logs:



```
└ docker exec -it stader-guardian ./submit-sd-price
SD Price In ETH:  NaN
SD Price Data: {ReportingBlockNumber:+9 SdPriceInETH:-9223372036854775808}
Submitted transaction hash: 0xa6113de1173bb206734d507a13417c4462057e71d576265437e60e92e15449e1
Done
```

The aforementioned program execution results in the oracle price being set to a huge value in the mocked smart contract:



```
└─ cast call 0xCf7Ed3AccA5a467e9e704C703E8D87F634fB0Fc9 "getStaderPrice()"
0xffffffffffffffffffffffffffffffffffffffffffffffff8000000000000000
```

Recommendation:

The program should handle this kind of responses since, as it was explained above, it can lead to unexpected and devastating results that could affect the several smart contracts.

Remediation Plan:

**SOLVED:** The StaderLabs team solved the issue by checking if the problematic division results in a NaN value in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.2 (HAL-02) ORACLE PRICE FEEDER IS VULNERABLE TO MANIPULATION BY A SINGLE MALICIOUS FEED - LOW (4.2)

Description:

The system currently relies on a price-feeder component, which fetches average prices from **CoinGecko** and sends this data to the smart contract for on-chain commitment. We have identified a vulnerability within this system, where the price of an asset can be manipulated by a single compromised or malfunctioning third-party feed. This poses a risk to the accuracy of our data and the reliability of transactions dependent on this data.

The vulnerability stems from the FetchPriceFromCoingecko function within the system, which fetches and computes average price data from CoinGecko for a given time range. If the data source is compromised, it can lead to erroneous price data being committed on-chain, with potential adverse effects on the system.

Code Location:

Listing 4: external/coingecko.go

```go
func FetchPriceFromCoingecko(from int64, to int64) (float64, error
↳ ) {
    url := fmt.Sprintf("https://api.coingecko.com/api/v3/coins/
↳ stader/market_chart/range?vs_currency=eth&from=%d&to=%d", from, to
↳ )
    response, err := http.Get(url)
    if err != nil {
        return 0, fmt.Errorf("error fetching data from Coingecko:
↳ %w", err)
    }
    defer response.Body.Close()

    body, err := io.ReadAll(response.Body)
    if err != nil {
        return 0, fmt.Errorf("error reading response body: %w",
↳ err)
    }

    var coingeckoResponse CoingeckoResponse
    err = json.Unmarshal(body, &coingeckoResponse)
    if err != nil {
        return 0, fmt.Errorf("error unmarshalling response: %w",
↳ err)
    }

    averagePrice := 0.0
    for _, price := range coingeckoResponse.Prices {
        averagePrice += price[1]
    }
    averagePrice /= float64(len(coingeckoResponse.Prices))

    return averagePrice, nil
}
```

BVSS:

**AO:A/AC:M/AX:M/C:L/I:H/A:L/D:L/Y:N/R:N/S:U (4.2)**

Recommendation:

It is recommended to consider the robustness and redundancy of the price feed sources. To mitigate the risk of price manipulation, we might consider incorporating multiple price feeds and aggregating the data to establish a more reliable and resilient pricing model.

**Reference :** Synthetix Response to Oracle Incident

Remediation Plan:

**RISK ACCEPTED:** The StaderLabs team states that SD prices are updated multiple times a day. As a result, different Oracles read the API values at different points in time. In order to manage these potential discrepancies, the system has been designed to take the median of the

SD prices from submitted values as the true value, thereby mitigating any issues arising from outliers. The impact from lack an of consensus affects only the secondary security collateral and has no effect on Staked ETH. The StaderLabs team will bolster this logic with data pulled from other API providers and on chain **TWAP** in the subsequent upgrades.

## 4.3 (HAL-03) PRIVATE KEY IN THE ENVIRONMENT VARIABLES - LOW (2.5)

Description:

The provided configuration contains sensitive data, such as the private key. Storing these values in a plain-text configuration file or environment variables can make it easier for attackers to gain unauthorized access to the oracle.

The insecure storage and handling of sensitive configuration data on a single VPS server can lead to various negative consequences, including but not limited to:

- Sensitive data leakage: Exposure of sensitive information, such as API keys, private keys, and login credentials, can enable attackers to gain unauthorized access to the service and its associated resources. This can result in further unauthorized actions, such as data manipulation or theft, disruption of services, and reputational damage to the organization.
- Unauthorized access to the API: With access to the API keys and credentials, attackers can make unauthorized API calls and potentially gain access to sensitive data, manipulate data, or perform other malicious activities.
- Potential loss of assets: Exposure of private keys for blockchain contracts can lead to unauthorized transactions or manipulation of the contract, resulting in potential loss of assets or funds.
- Increased risk of server compromise: Storing sensitive data in plaintext on a VPS server increases the attack surface, making it more attractive for attackers to target the server. A successful compromise of the server can lead to further damage, such as the installation of malware, lateral movement within the infrastructure, or complete takeover of the server.

Code Location:

Listing 5: .env.sample

```
ExecutionHost=
ConsensusHost=
PrivateKey=
ChainId=
```

BVSS:

**AO:A/AC:M/AX:M/C:H/I:H/A:M/D:L/Y:N/R:P/S:U (2.5)**

Recommendation:

To mitigate the potential risks and secure the sensitive configuration data on the VPS server, the following steps are suggested:

- Store sensitive data securely using a secret manager like HashiCorp Vault or AWS Secrets Manager. Avoid using environment variables for sensitive data, and instead, retrieve them directly from the secret manager in the service code.
- Harden the VPS server by implementing security best practices, such as keeping the server up-to-date, disabling unnecessary services, and restricting access using firewall rules and strong authentication mechanisms.
- Segregate responsibilities by deploying separate servers or containers for different components of the service. For example, use a dedicated server or container for the API, another for the database, and another for the secret manager.
- Regularly monitor and audit the VPS server for signs of intrusion or other security issues. Configure intrusion detection and prevention systems (IDPS) and implement centralized logging for better visibility and faster incident response.

- Enable encryption for data in transit and at rest to protect sensitive data from being intercepted or accessed by unauthorized users.

Remediation Plan:

**RISK ACCEPTED:** The StaderLabs team assures that all Oracle operators are recognized and reputable members within the Ethereum community. A comprehensive list of these operators can be found here. Furthermore, for the insertion of erroneous data into contracts, collusion among a majority of Oracle nodes would be necessary, an occurrence that is highly unlikely.

In addition to this, The StaderLabs team states that stringent safeguards have been put in place for critical Oracle updates, such as exchange rate adjustments. Also, Oracle operators are technically adept and thus fully equipped to ensure the security of on-premises data.

The StaderLabs team firmly believes that the probability of a successful attack is virtually non-existent under the current framework. Nonetheless, they continue to strive for excellence and improvement, with plans underway to enhance data storage systems in future upgrades.

## 4.4 (HAL-04) DOCKER IMAGE RUNNING AS ROOT - LOW (3.8)

Description:

Docker containers generally run with root privileges by default. This allows for unrestricted container management, meaning a user could install system packages, edit configuration files, bind privileged ports, etc. During static analysis, it was observed that the docker image is maintained through the root user.

Code Location:

Listing 6: Dockerfile

```
FROM golang:1.20

# Set destination for COPY
WORKDIR /app

# Download Go modules
COPY go.mod go.sum ./
RUN go mod download

COPY . ./

# Build
RUN CGO_ENABLED=0 GOOS=linux go build -C cmd/sdPrice -o ../../
↳ submit-sd-price
RUN CGO_ENABLED=0 GOOS=linux go build -C cmd/validatorStats -o
↳ ../../submit-validator-stats
RUN CGO_ENABLED=0 GOOS=linux go build -C cmd/withdrawnValidators -
↳ o ../../submit-withdrawn-validators
RUN CGO_ENABLED=0 GOOS=linux go build -C cmd/rewardsMerkle -o
↳ ../../submit-rewards-merkle
RUN CGO_ENABLED=0 GOOS=linux go build -C cmd/exchangeRate -o
↳ ../../submit-exchange-rate
```

BVSS:

**AO:A/AC:L/AX:M/C:M/I:L/A:N/D:N/Y:N/R:N/S:U (3.8)**

Recommendation:

It is recommended to build the Dockerfile and run the container as a non-root user.

Listing 7: Reference

```
USER 1001: this is a non-root user UID, and here it is assigned to
↳  the image to run the current container as an unprivileged user.
↳ By doing so, the added security and other restrictions mentioned
↳ above are applied to the container.
```

Remediation Plan:

**SOLVED:** The StaderLabs team solved the issue by adding a new non-root user in the Dockerfile and executing commands through it in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.5 (HAL-05) MAINNET CONFIGURATION ADDRESS IS UNDEFINED IN STADERCONFIGADDRESSSTR - INFORMATIONAL (0.0)

Description:

The StaderConfigAddressStr variable is expected to store the contract addresses for different Ethereum networks. In the present configuration, the goerli test network is mapped correctly to an address. However, the address for the mainnet is missing, and this could lead to failure in operations that require interaction with the Stader contract on the mainnet.

Potential implications of this issue include:

- Failure in contract deployments to the mainnet.
- Breakdown in interaction between the Stader contract and other on-chain services.
- Disruptions in services relying on the Stader contract.

Code Location:

Listing 8: internal/common.go

```go
var StaderConfigAddressStr = map[uint]string{
    1:     "",                                          // mainnet
    5:     "0x8eF9036E524ce6340eF71844C29508C26Fbbe478", // goerli
}
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

To resolve this issue, it is strongly recommended to define the mainnet address in the StaderConfigAddressStr map, following the proper deployment of the Stader contract to the mainnet.

Remediation Plan:

**SOLVED:** The StaderLabs team solved the issue by including the mainnet address in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.6 (HAL-06) LACK OF EXTENSIVE TEST COVERAGE - INFORMATIONAL (0.0)

Description:

Adequate test coverage and regular reporting is an essential process to ensure the codebase works as intended. Insufficient code coverage can lead to unexpected issues and regressions due to changes in service implementation.

Code Location:

Code Location

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

Make sure that the coverage report produced via **go test -cover** covers all functions.

Remediation Plan:

**PARTIALLY SOLVED:** The StaderLabs team partially solved the issue by including some tests to cover some of the project's functions in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.7 (HAL-07) POTENTIAL HTTP REQUEST TIMEOUT MISSING IN HTTP CLIENT CONFIGURATION - INFORMATIONAL (0.0)

Description:

In the existing code, the http.Get function is used for making HTTP requests. This approach doesn't specify a timeout duration, meaning that if the server does not respond, the request could hang indefinitely. This situation could lead to resources being consumed unnecessarily, affecting the overall performance and reliability of the application.

Consider a scenario where the requested server is down, unresponsive, or the network is congested. In such a case, the HTTP request will hang indefinitely as there is no timeout defined.

Code Location:

Listing 9: external/beacon.go

```go
func GetValidatorDetails(consensusHost string, pubKey string,
↳ slotNumber uint64) (ValidatorBeaconResponse, error) {
    url := fmt.Sprintf("%s/eth/v1/beacon/states/%d/validators/%s",
↳  consensusHost, slotNumber, pubKey)
    resp, err := http.Get(url)
    if err != nil {
        return ValidatorBeaconResponse{}, fmt.Errorf("failed to
↳ fetch validator data: %w", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return ValidatorBeaconResponse{
            Balance: big.NewInt(0),
            Status:  string(NotFound),
            Validator: ValidatorBeacon{
                Slashed: false,
                Pubkey:  pubKey,
            },
        }, nil
    }

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return ValidatorBeaconResponse{}, fmt.Errorf("failed to
↳ read response body: %w", err)
    }
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

To mitigate this issue, it is suggested to use a custom HTTP client with a defined timeout. This change ensures that the HTTP request will be terminated if it exceeds the defined timeout period, preventing resource wastage and maintaining application performance.

**Listing 10**

```go
var netClient = &http.Client{
  Timeout: time.Second * 10,  // Define a suitable timeout
}

url := "consensusHost"
resp, err := netClient.Get(url) /
```

Remediation Plan:

**ACKNOWLEDGED:** The StaderLabs team acknowledged this issue.

## 4.8 (HAL-08) ABSENCE OF CHECK FOR whenNotPaused CONDITION IN GOLANG CODEBASE - INFORMATIONAL (0.0)

Description:

The provided Golang script is communicating with a smart contract that has a whenNotPaused modifier on some functions, but the script does not check whether the contract is paused before making these calls. This discrepancy could cause the calls to fail unexpectedly.

The issue occurs when the Golang script attempts to call the SubmitSocializingRewardsMerkleRoot function, which has the whenNotPaused modifier in the smart contract.

If the contract is paused, the transaction will fail but the Golang script will not understand why, leading to potential confusion and the disruption of normal operations.

Code Location:

**Listing 11: rewardsMerkle/task.go**

```go
func fetchAndSubmit(staderOracle *contracts.StaderOracle, poolId
↳ uint8, auth *bind.TransactOpts) error {
    blockNumber, err := staderOracle.
↳ GetMerkleRootReportableBlockByPoolId(&bind.CallOpts{}, poolId)
    if err != nil {
        return fmt.Errorf("failed to get latest reportable block:
↳ %w", err)
    }
    fmt.Printf("block number: %d\n", blockNumber)

    lastIndex, err := cmd.ReadBlockNumber(fileName + string(poolId
↳ ))
    if err != nil {
        lastIndex = big.NewInt(0)
    }

    index, err := staderOracle.GetCurrentRewardsIndexByPoolId(&bind.CallOpts{}, poolId)
    if err != nil {
        return err
    }
    fmt.Printf("rewards index: %d\n", index)

    if big.NewInt(index.Int64()).Cmp(lastIndex) == 0 {
        fmt.Printf("No new index to process. Last processed index: %d\n", lastIndex)
        return nil
    }

    response, err := external.FetchRewardsMerkle(int(index.Uint64()), int(poolId))
    if err != nil {
        return err
    }
    transformedData, err := response.Into(index, poolId, blockNumber)
    if err != nil {
        return err
    }
    fmt.Printf("transformed data: %+v\n", transformedData)

    tx, err := staderOracle.SubmitSocializingRewardsMerkleRoot(auth, transformedData)
    if err != nil {
        return fmt.Errorf("failed to submit Merkle Root: %w", err)
    }
    fmt.Printf("Submitted transaction hash: %s\n", tx.Hash().Hex())

    err = cmd.WriteBlockNumber(fileName+string(poolId), big.NewInt(index.Int64()))
    if err != nil {
        return fmt.Errorf("error writing counter file: %v", err)
    }

    return nil
}
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

To address this issue, add a check for the contract's pause status before making function calls that include the whenNotPaused modifier. If the contract is paused, either handle this condition gracefully or display an appropriate error message. For instance:

Listing 12

```go
isPaused, err := staderOracle.GetPaused(&bind.CallOpts{})
if err != nil {
    return fmt.Errorf("failed to get contract pause status: %w",
    ↳ err)
}
if isPaused {
    return fmt.Errorf("operation failed: contract is paused")
}

tx, err := staderOracle.SubmitSocializingRewardsMerkleRoot(auth,
↳ transformedData)
// ...
```

Remediation Plan:

**ACKNOWLEDGED:** The StaderLabs team acknowledged this issue.

## 4.9 (HAL-09) ABI INCOMPATIBILITY WITH RECENT CODEBASE IN GO FILES - INFORMATIONAL (0.0)

Description:

After the recent updates in the codebase, it appears that the ABI (Application Binary Interface) defined in go files may no longer be compatible with the current smart contract. ABI incompatibility can lead to issues with communication between the Ethereum blockchain and the application. This can cause problems with executing transactions or fetching data from smart contracts.

Code Location:

Code Location

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

It is recommended to revise the ABI specifications in go files to ensure compatibility with the current smart contract code.
### Remediation Plan

**SOLVED:** The StaderLabs team solved the issue by applying the most recent changes in ABI specification in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.10 (HAL-10) HARDCODED URL IN THE FUNCTIONS - INFORMATIONAL (0.0)

Description:

The URL in the functions are hard-coded. This could potentially cause maintenance and scalability issues in the future, especially if the URL endpoint changes or if the function needs to be adapted for different environments, such as testing, development, and production. Currently, the function is rigidly tied to a specific URL, and any change to this URL would require a modification in the code itself.

Code Location:

Listing 13: external/stader.go (Line 60)

```
func FetchRewardsMerkle(cycle_id int, pool_id int) (*
↳ RewardsMerkleResponse, error) {
    url := "https://stage-ethx-offchain.staderlabs.click/
↳ merklesForElRewards/root/" + strconv.Itoa(cycle_id) + "/" +
↳ strconv.Itoa(pool_id)

    resp, err := http.Get(url)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }

    var apiResponse RewardsMerkleResponse
    err = json.Unmarshal(body, &apiResponse)
    if err != nil {
        return nil, err
    }

    return &apiResponse, nil
}
```

Listing 14: external/coingecko.go (Line 18)

```go
func FetchPriceFromCoingecko(from int64, to int64) (float64, error
↳ ) {
    url := fmt.Sprintf("https://api.coingecko.com/api/v3/coins/
↳ stader/market_chart/range?vs_currency=eth&from=%d&to=%d", from, to
↳ )
    response, err := http.Get(url)
    if err != nil {
        return 0, fmt.Errorf("error fetching data from Coingecko:
↳ %w", err)
    }
    defer response.Body.Close()

    body, err := io.ReadAll(response.Body)
    if err != nil {
        return 0, fmt.Errorf("error reading response body: %w",
↳ err)
    }

    var coingeckoResponse CoingeckoResponse
    err = json.Unmarshal(body, &coingeckoResponse)
    if err != nil {
        return 0, fmt.Errorf("error unmarshalling response: %w",
↳ err)
    }

    averagePrice := 0.0
    for _, price := range coingeckoResponse.Prices {
        averagePrice += price[1]
    }
    averagePrice /= float64(len(coingeckoResponse.Prices))

    return averagePrice, nil
}
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

Instead of hardcoding the URL directly within the function, it's recommended to:

- Use configuration files or environment variables: This approach is safer and makes your application more flexible. In production environments, you may have different URLs, credentials, or other data, and using configuration files or environment variables allows you to easily switch between different setups without changing the code.

Remediation Plan:

**ACKNOWLEDGED:** The StaderLabs team acknowledged this issue.

## 4.11 (HAL-11) LACK OF HTTP RESPONSE STATUS CODE CHECK IN THE FUNCTIONS - INFORMATIONAL (0.0)

Description:

In the provided the functions, while errors during the HTTP request and the reading of the response body are properly handled, there is no validation that the HTTP response status code indicates success. This means that the function may proceed to process an unexpected response as if it were valid data, potentially leading to obscure errors later in the application's execution.

Code Location:

**Listing 15: external/stader.go**

```go
func FetchRewardsMerkle(cycle_id int, pool_id int) (*RewardsMerkleResponse, error) {
    url := "https://stage-ethx-offchain.staderlabs.click/merklesForElRewards/root/" + strconv.Itoa(cycle_id) + "/" + strconv.Itoa(pool_id)

    resp, err := http.Get(url)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }

    var apiResponse RewardsMerkleResponse
    err = json.Unmarshal(body, &apiResponse)
    if err != nil {
        return nil, err
    }

    return &apiResponse, nil
}
```

Listing 16: external/coingecko.go

```go
func FetchPriceFromCoingecko(from int64, to int64) (float64, error
↳ ) {
    url := fmt.Sprintf("https://api.coingecko.com/api/v3/coins/
↳ stader/market_chart/range?vs_currency=eth&from=%d&to=%d", from, to
↳ )
    response, err := http.Get(url)
    if err != nil {
        return 0, fmt.Errorf("error fetching data from Coingecko:
↳ %w", err)
    }
    defer response.Body.Close()

    body, err := io.ReadAll(response.Body)
    if err != nil {
        return 0, fmt.Errorf("error reading response body: %w",
↳ err)
    }

    var coingeckoResponse CoingeckoResponse
    err = json.Unmarshal(body, &coingeckoResponse)
    if err != nil {
        return 0, fmt.Errorf("error unmarshalling response: %w",
↳ err)
    }

    averagePrice := 0.0
    for _, price := range coingeckoResponse.Prices {
        averagePrice += price[1]
    }
    averagePrice /= float64(len(coingeckoResponse.Prices))

    return averagePrice, nil
}
```

BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

In order to improve error handling, it's recommended to add a check for the HTTP response status code after making the request. If the status code does not indicate success (usually a 200 status code), the function should return an error.

Here's an example of how you could implement this:

Listing 17

```go
resp, err := http.Get(url)
if err != nil {
    return nil, err
}
if resp.StatusCode != http.StatusOK {
    return nil, fmt.Errorf("unexpected HTTP status: %s", resp.
↳ Status)
}
```

By checking the HTTP response status code, you can ensure that you only proceed with the rest of the function when you have successfully received the expected data from the server. This can help to prevent obscure errors from occurring later in the function's execution.

Remediation Plan:

**SOLVED:** The StaderLabs team solved the issue by checking HTTP response status codes in the following commit ID:

- d9288491ab6d0de57ff8d8b6cb6a0ee4361bd3dc.

## 4.12 (HAL-12) LACK OF ERROR HANDLING FOR NON-EXISTENT CHAINID IN ConvertTimestampToSlotAndEpoch FUNCTION - INFORMATIONAL (0.0)

### Description:

In the provided ConvertTimestampToSlotAndEpoch function, there is an implicit assumption that the chainId provided as a function argument always exists within the genesisTime map. However, in cases where a chainId that is not present in the genesisTime map is passed to the function, the time.Unix() call would be made with a zero value, potentially leading to unexpected results.

### Code Location:

**Listing 18: internal/utils.go**

```go
func ConvertTimestampToSlotAndEpoch(timestamp time.Time, chainId
↳ uint) (uint64, uint64) {
    genesisTime := time.Unix(int64(genesisTime[chainId]), 0)
    timeSinceGenesis := timestamp.Sub(genesisTime)
    slotNumber := uint64(timeSinceGenesis.Seconds()) /
↳ SECONDS_PER_SLOT
    epochNumber := slotNumber / SLOTS_PER_EPOCH

    return slotNumber, epochNumber
}
```

### BVSS:

**AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)**

Recommendation:

It is recommended to add an explicit check for the existence of the chainId within the genesisTime map before using its value. If the chainId does not exist within the map, the function should return an error.

Remediation Plan:

**ACKNOWLEDGED:** The StaderLabs team acknowledged this issue.

# RECOMMENDATIONS OVERVIEW

- The code should be modified to handle the case when the Coingecko API returns an empty response. If the Prices array is empty, the function should raise an exception or return a default value, rather than proceeding with the calculation of the average. This will prevent the NaN issue and avoid the setting of undesired prices on the chain.
- Consider implementing a mechanism to cross-check prices from multiple sources to ensure accuracy and reduce the risk of manipulation. It's also worth considering implementing sanity checks or limits on price changes to prevent large, unexpected fluctuations.
- Storing private keys in the environment variables in plain text are a significant security risk. It would be more secure to store them encrypted and decrypt them only when needed. Alternatively, consider using a secure secret management solution like AWS Secrets Manager or Hashicorp's Vault.
- Running Docker containers as root provides a potential attack surface. You should consider creating a dedicated, non-root user for running the Docker containers. This user should have the least possible privileges required to perform the necessary tasks.
- It is crucial to specify a timeout for HTTP requests to prevent the server from hanging indefinitely. You can use the Client struct in the net/http package to set a timeout for requests.
- Increasing test coverage and regularly running these tests is a critical part of maintaining a robust codebase. Also, consider using tools that provide test coverage metrics to ensure that the tests cover a sufficient part of the code.
- Implement a check in the Golang script for the whenNotPaused condition in the smart contract. If the contract is paused, the script should not attempt to make calls that require the contract to be unpaused. This will prevent unnecessary errors and make the system more robust.

# AUTOMATED TESTING

Description:

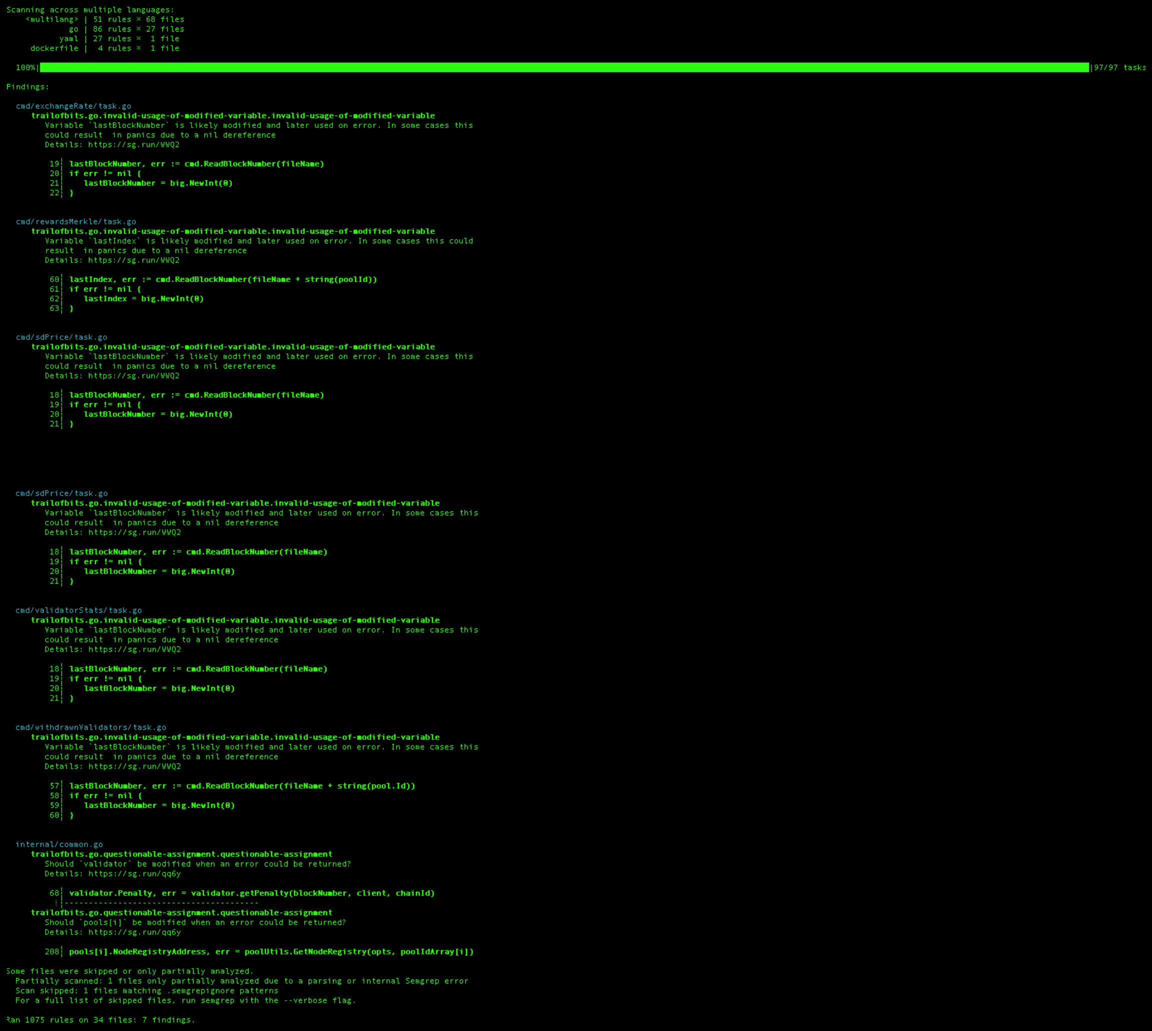
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were staticcheck, gosec, semgrep, unconvert, LGTM and Nancy. After Halborn verified all the contracts and scoped structures in the repository and was able to compile them correctly, these tools were leveraged on scoped structures. With these tools, Halborn can statically verify security related issues across the entire codebase.

Semgrep - Security Analysis Output Sample:

**Listing 19: Rule Set**

```
semgrep --config "p/dgryski.semgrep-go" x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o dgryski.semgrep
semgrep --config "p/owasp-top-ten"       x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o owasp-top-ten.
↳ semgrep
semgrep --config "p/r2c-security-audit" x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o r2c-security-audit.
↳ semgrep
semgrep --config "p/r2c-ci"              x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o r2c-ci.semgrep
semgrep --config "p/ci"                  x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o ci.semgrep
semgrep --config "p/golang"              x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o golang.semgrep
semgrep --config "p/trailofbits"         x --exclude='*_test.go' --
↳ max-lines-per-finding 1000 --no-git-ignore -o trailofbits.semgrep
```

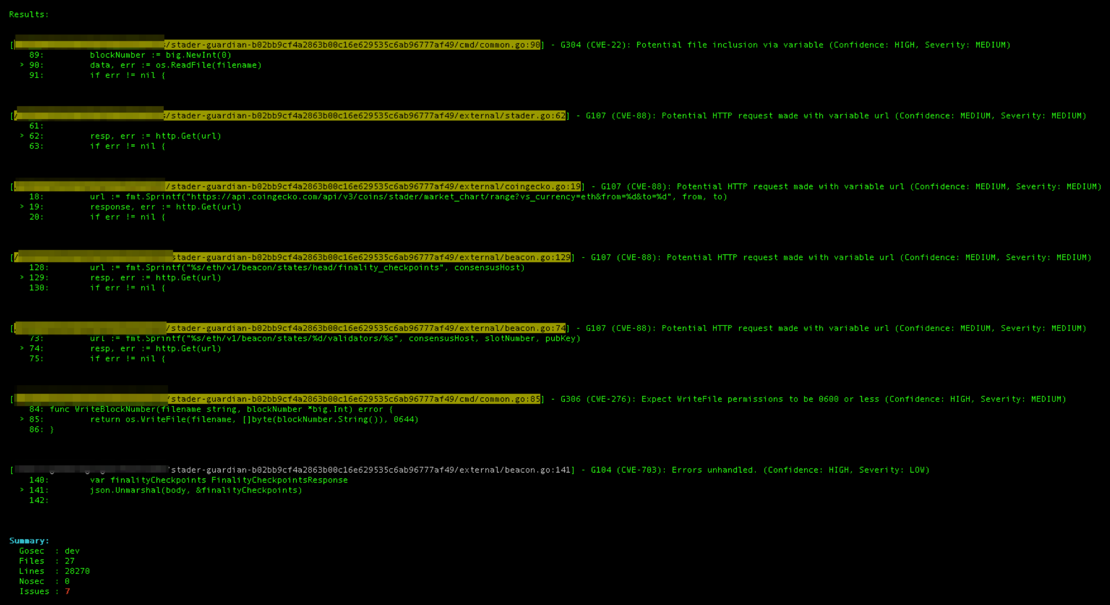
## Semgrep Results:



```
Scanning across multiple languages:
    <multilang> | 51 rules × 68 files
             go | 86 rules × 27 files
           yaml | 27 rules ×  1 file
     dockerfile |  4 rules ×  1 file

  100%|████████████████████████████████████████████████████████████|97/97 tasks

Findings:

  cmd/exchangeRate/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastBlockNumber` is likely modified and later used on error. In some cases this
        could result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         19┆ lastBlockNumber, err := cmd.ReadBlockNumber(fileName)
         20┆ if err != nil {
         21┆    lastBlockNumber = big.NewInt(0)
         22┆ }


  cmd/rewardsMerkle/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastIndex` is likely modified and later used on error. In some cases this could
        result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         60┆ lastIndex, err := cmd.ReadBlockNumber(fileName + string(poolId))
         61┆ if err != nil {
         62┆    lastIndex = big.NewInt(0)
         63┆ }


  cmd/sdPrice/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastBlockNumber` is likely modified and later used on error. In some cases this
        could result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         18┆ lastBlockNumber, err := cmd.ReadBlockNumber(fileName)
         19┆ if err != nil {
         20┆    lastBlockNumber = big.NewInt(0)
         21┆ }




  cmd/sdPrice/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastBlockNumber` is likely modified and later used on error. In some cases this
        could result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         18┆ lastBlockNumber, err := cmd.ReadBlockNumber(fileName)
         19┆ if err != nil {
         20┆    lastBlockNumber = big.NewInt(0)
         21┆ }


  cmd/validatorStats/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastBlockNumber` is likely modified and later used on error. In some cases this
        could result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         18┆ lastBlockNumber, err := cmd.ReadBlockNumber(fileName)
         19┆ if err != nil {
         20┆    lastBlockNumber = big.NewInt(0)
         21┆ }


  cmd/withdrawnValidators/task.go
     trailofbits.go.invalid-usage-of-modified-variable.invalid-usage-of-modified-variable
        Variable `lastBlockNumber` is likely modified and later used on error. In some cases this
        could result  in panics due to a nil dereference
        Details: https://sg.run/WWQ2

         57┆ lastBlockNumber, err := cmd.ReadBlockNumber(fileName + string(pool.Id))
         58┆ if err != nil {
         59┆    lastBlockNumber = big.NewInt(0)
         60┆ }


  internal/common.go
     trailofbits.go.questionable-assignment.questionable-assignment
        Should `validator` be modified when an error could be returned?
        Details: https://sg.run/qq6y

         68┆ validator.Penalty, err = validator.getPenalty(blockNumber, client, chainId)
          ⋮┆----------------------------------------
     trailofbits.go.questionable-assignment.questionable-assignment
        Should `pools[i]` be modified when an error could be returned?
        Details: https://sg.run/qq6y

        208┆ pools[i].NodeRegistryAddress, err = poolUtils.GetNodeRegistry(opts, poolIdArray[i])

Some files were skipped or only partially analyzed.
  Partially scanned: 1 files only partially analyzed due to a parsing or internal Semgrep error
  Scan skipped: 1 files matching .semgrepignore patterns
  For a full list of skipped files, run semgrep with the --verbose flag.

Ran 1075 rules on 34 files: 7 findings.
```

- No major issues found by Semgrep.

## Gosec - Security Analysis Output Sample:



```
Results:

[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/cmd/common.go:90] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
    89:         blockNumber := big.NewInt(0)
  > 90:         data, err := os.ReadFile(filename)
    91:         if err != nil {


[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/external/stader.go:62] - G107 (CWE-88): Potential HTTP request made with variable url (Confidence: MEDIUM, Severity: MEDIUM)
    61:
  > 62:         resp, err := http.Get(url)
    63:         if err != nil {


[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/external/coingecko.go:19] - G107 (CWE-88): Potential HTTP request made with variable url (Confidence: MEDIUM, Severity: MEDIUM)
    18:         url := fmt.Sprintf("https://api.coingecko.com/api/v3/coins/stader/market_chart/range?vs_currency=eth&from=%d&to=%d", from, to)
  > 19:         response, err := http.Get(url)
    20:         if err != nil {


[[redacted]stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/external/beacon.go:129] - G107 (CWE-88): Potential HTTP request made with variable url (Confidence: MEDIUM, Severity: MEDIUM)
    128:        url := fmt.Sprintf("%s/eth/v1/beacon/states/head/finality_checkpoints", consensusHost)
  > 129:        resp, err := http.Get(url)
    130:        if err != nil {


[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/external/beacon.go:74] - G107 (CWE-88): Potential HTTP request made with variable url (Confidence: MEDIUM, Severity: MEDIUM)
    73:         url := fmt.Sprintf("%s/eth/v1/beacon/states/%d/validators/%s", consensusHost, slotNumber, pubKey)
  > 74:         resp, err := http.Get(url)
    75:         if err != nil {


[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/cmd/common.go:85] - G306 (CWE-276): Expect WriteFile permissions to be 0600 or less (Confidence: HIGH, Severity: MEDIUM)
    84: func WriteBlockNumber(filename string, blockNumber *big.Int) error {
  > 85:         return os.WriteFile(filename, []byte(blockNumber.String()), 0644)
    86: }


[[redacted]/stader-guardian-b02bb9cf4a2863b00c16e629535c6ab96777af49/external/beacon.go:141] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
    140:        var finalityCheckpoints FinalityCheckpointsResponse
  > 141:        json.Unmarshal(body, &finalityCheckpoints)
    142:


Summary:
  Gosec  : dev
  Files  : 27
  Lines  : 28270
  Nosec  : 0
  Issues : 7
```

- No major issues found by Gosec.

THANK YOU FOR CHOOSING

// HALBORN