PUBLIC

# Code Assessment

of the LIP-23: Rebase Check Smart Contracts

Jun 21, 2024

Produced for

LIDO

by

CHAINSECURITY

# Contents

# 1 Executive Summary

Dear Lido Team,

Thank you for trusting us to help Lido with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of LIP-23: Rebase Check according to Scope to support you in forming an opinion on their security risks.

Lido implements an improvement of the `OracleReportSanityChecker` which aims to mitigate the risk of malicious oracle daemons colluding and reporting excessive negative rebases of stETH.

The most critical subjects covered in our audit are compliance with the specification, correctness of the arithmetic operations, and functional correctness. No major issues were uncovered. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 2 |
| • Code Corrected | 1 |
| • Specification Changed | 1 |

# 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the LIP-23: Rebase Check repository based on the documentation files.

The audit was carried out on the changes introduced through the GitHub pull request "Negative rebase limit #67". More specifically, only the changes from the `develop` branch to Version 2 were taken into consideration. The codebase at `develop` was assumed to be bug-free.

The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 27 May 2024 | 694e7b10f5f8a00e3b7634e4e020539b3a67b20d | Initial Version |
| 2 | 18 Jun 2024 | efeff81c18f85451ebf98e8fd8bb78b8eb0095f6 | Version with fixes |

For the solidity smart contracts, the compiler version `0.8.9` was chosen.

The following files are in scope:

- code/lido-core_code/contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol
- code/lido-core_code/contracts/0.8.9/lib/SafeCastExt.sol

### 2.1.1 *Excluded from scope*

All files not specified in the scope are automatically considered to be out-of-scope. `OracleReportSanityChecker` is assumed to work correctly. Only the changes in its functionality were reviewed. Moreover, the correctness of Oracle Daemons is out of scope. In the current setup, the liveness of the system might be impacted if high negative rebasing is reported. We assume that the Lido Governance will be able to correctly configure the system to make progress. This is an assessment of the correct implementation of the specification of LIP-23. The specification itself was not subject to review. LIP-23 makes assumptions based on the current specification of the consensus layer. Should this change, the sanity check may be revisited.

## 2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Lido offers LIP-23, an improvement of `OracleReportSanityChecker` which aims to mitigate the risk of malicious Oracle daemons colluding and reporting big negative rebases of stETH. For more information about Lido core, please refer to previous reviews.

### 2.2.1 stETH

Lido is a liquid staking protocol. Users can stake their ETH in exchange for stETH. The stETH contract is an ERC20 rebasing token. User balances are internally represented as shares and their actual balances are a product of the total shares and the amount of ETH per share currently held by active validators, and pending validators, and buffered in the Lido contract. As the amount of ETH per share increases due to rewards, the user balances also increase proportionally. Negative rebasing can also occur in case of slashing or other penalties to the Lido validators.

### 2.2.2 AccountingOracle

The rebasing mechanism of the stETH strongly depends on information coming from the consensus layer. This information is contained in an Oracle Report that is transmitted daily to the `AccountingOracle` from off-chain entities named Oracle Daemons. There are currently nine Oracle Daemons, and out of these, it is sufficient for five of them to reach a consensus in order for the report to be finalized.

The Oracle Daemons call the function `submitReportData` of the `AccountingOracle` effectively passing the report on-chain.

### 2.2.3 OracleReportSanityChecker

Before the data of the Oracle report can be used, it is necessary to perform several sanity checks. This is to avert the rare case of the committee of Oracle Daemons is compromised, malfunctions, or colludes.

On a high level, the `OracleReportSanityChecker` performs the following checks on a report submitted on the i-th day:

1. The amount to withdraw on the i-th day cannot be larger than the balance of the `WithdrawalVault`.
2. The amount of execution layer rewards received on the i-th day cannot be larger than the balance of `elRewardsVault`.
3. The shares to burn on the i-th day cannot be more than the ones stored in the `Burner` contract.
4. The total ether staked by Lido cannot decrease more than a certain threshold.
5. The total ether staked by Lido cannot increase more than a maximum APY.
6. The number of new Lido validators that appeared on the i-th day cannot be more than the Ethereum validators' churn limit.

### 2.2.4 Negative Rebase Sanity Check

The focus of this audit is the changes to check number 4, which aims to lower the threshold of a tolerated negative rebase. A negative rebase beyond this threshold could imply malfunctioning or colluding Oracle Daemons. Note, however, that the check cannot detect wrong/false values that still satisfy its conditions. In particular, the check requires that the sum of the absolute values of the negative rebases in the last 18 days does not exceed the sum of:

1. the slashing penalty (1 ETH) of an overapproximation of all the active validators of the last 18 days
2. the inactivity penalty (0.101 ETH) of an overapproximation of all the active validators of the last 54 days

In case the sanity check fails, the `OracleReportSanityChecker` will ask a second opinion from a specific `SecondOpinionOracle`. If the second opinion is consistent with the data received from the rest of the Oracle committee, then the negative rebase will happen. It is important to note that at the current state no `SecondOpinionOracle` is implemented, and therefore the transaction will simply revert in such case. If the address of the second-opinion Oracle is set then the value it reports is sanitized by comparing it to the value reported. Note that the tolerance for negative rebases depends on the frequency of the report as less frequent reports allow for bigger negative rebases.

### 2.2.5 Changes in Version 2

Version 2 of the system introduces the following changes:

- `OracleReportSanityChecker.checkAccountingOracleReport()` can only be called by the Lido contract.
- The second opinion oracle returns an extra value `withdrawalVaultBalanceWei` which is checked against the `oracleWithdrawalVaultBalanceWei`.

### 2.2.6 Roles & Trust Model

We identified the following roles:

- Oracle Daemon: they daily submit an Oracle report. In general, they are not trusted.
- Lido Governance: in case the sanity check fails and a second opinion is needed, the Lido Governance is assumed to deploy a proper second-opinion oracle to allow the system to progress.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| **Likelihood** | **Impact** | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 2 |

- Specification Mismatch Code Corrected
- The Second Opinion Oracle Should Return a Unified Balance Specification Changed

| | |
|---|---|
| Informational Findings | 1 |

- Past Report Data Retrieval Code Corrected

## 6.1 Specification Mismatch

Correctness Low Version 1 Code Corrected

*CS-LIP23-003*

The specification states that in cases where `clRebaseSumNegative_18 > maxClRebaseNegativeSum`, then the check must try to retrieve a second opinion.

However, the check is implemented as follows:

```
if (negativeCLRebaseSum < maxAllowedCLRebaseNegativeSum) {
    // If the diff is less than limit we are finishing check
    emit NegativeCLRebaseAccepted(_refSlot, _unifiedPostCLBalance,
              negativeCLRebaseSum, maxAllowedCLRebaseNegativeSum);
    return;
}

...

_askSecondOpinion(_refSlot, _unifiedPostCLBalance, _limitsList)
```

This implies that if `negativeCLRebaseSum >= maxAllowedCLRebaseNegativeSum` then the check will try to retrieve a second opinion, which is not in line with the specifications. In particular, if `negativeCLRebaseSum = maxAllowedCLRebaseNegativeSum`, according to the specifications, a second opinion should **not** be consulted, whereas according to the code, a second opinion should be consulted.

---

**Core corrected:**

Lido has corrected the code in Version 2. Now the check is implemented as follows (notice the <= operator):

```
if (negativeCLRebaseSum <= maxAllowedCLRebaseNegativeSum) {
    // If the rebase diff is less or equal max allowed sum, we accept the report
    emit NegativeCLRebaseAccepted(_refSlot, _postCLBalance + _withdrawalVaultBalance,
                negativeCLRebaseSum, maxAllowedCLRebaseNegativeSum);
    return;
}

...

_askSecondOpinion(_refSlot, _postCLBalance, _withdrawalVaultBalance, _limitsList);
```

## 6.2 The Second Opinion Oracle Should Return a Unified Balance

**Design** **Low** **Version 1** **Specification Changed**

*CS-LIP23-001*

In `OracleReportSanityCheck._askSecondOpinion()`, the value `clBalanceGwei` returned by the second opinion oracle is converted to WEI and then compared to the unified Consensus Layer (CL) balance originating from the Accounting Oracle. This latter unified balance is equal to the CL balance plus the amount of Ether that was withdrawn from the Beacon Chain to `WithdrawalVault`.

Even though no implementation of the second opinion oracle is available yet, the variable name `clBalanceGwei` suggests that the balance returned by the second opinion will not represent a unified CL balance. In this case, the checks inside `_askSecondOpinion` might not be correct.

---

**Specification changed:**

Lido corrected the code in **Version 2**. Now `_askSecondOpinion` compares the post CL balance (and not the unified post CL balance) to the second opinion CL balance. Moreover, it requires that the balance of the withdrawal vault at the refslot is strictly equal to the balance reported by the second opinion (`oracleWithdrawalVaultBalanceWei == _withdrawalVaultBalance`). Note that according to the plan at the time of review, for the second opinion, some zk-solution or direct Merkle-Patricia proof (e.g., with EIP-2935) is going to be used. Therefore, for an honest oracle, the check should succeed.

## 6.3 Past Report Data Retrieval

**Informational** **Version 1** **Code Corrected**

*CS-LIP23-002*

In the specifications, the `negativeClRebaseSum_18` is computed through a sum over the values of the last 18 reports. However, developers should note that the implementation in `_sumNegativeRebasesNotOlderThan(uint256 _timestamp)` sums all reports having a timestamp greater or equal than `_timestamp`. As as consequences, it might be that the sum is executed over the last 19 reports, instead on 18, depending on the reference slot of the reports.

---

**Code corrected:**

Lido corrected the code in **Version 2** by only considering reports having a timestamp greater (and not equal) than `_timestamp`.

Note that since the updated function sums the negative rebases **newer** than `_timestamp`, the name `_sumNegativeRebasesNotOlderThan` is currently not accurate.