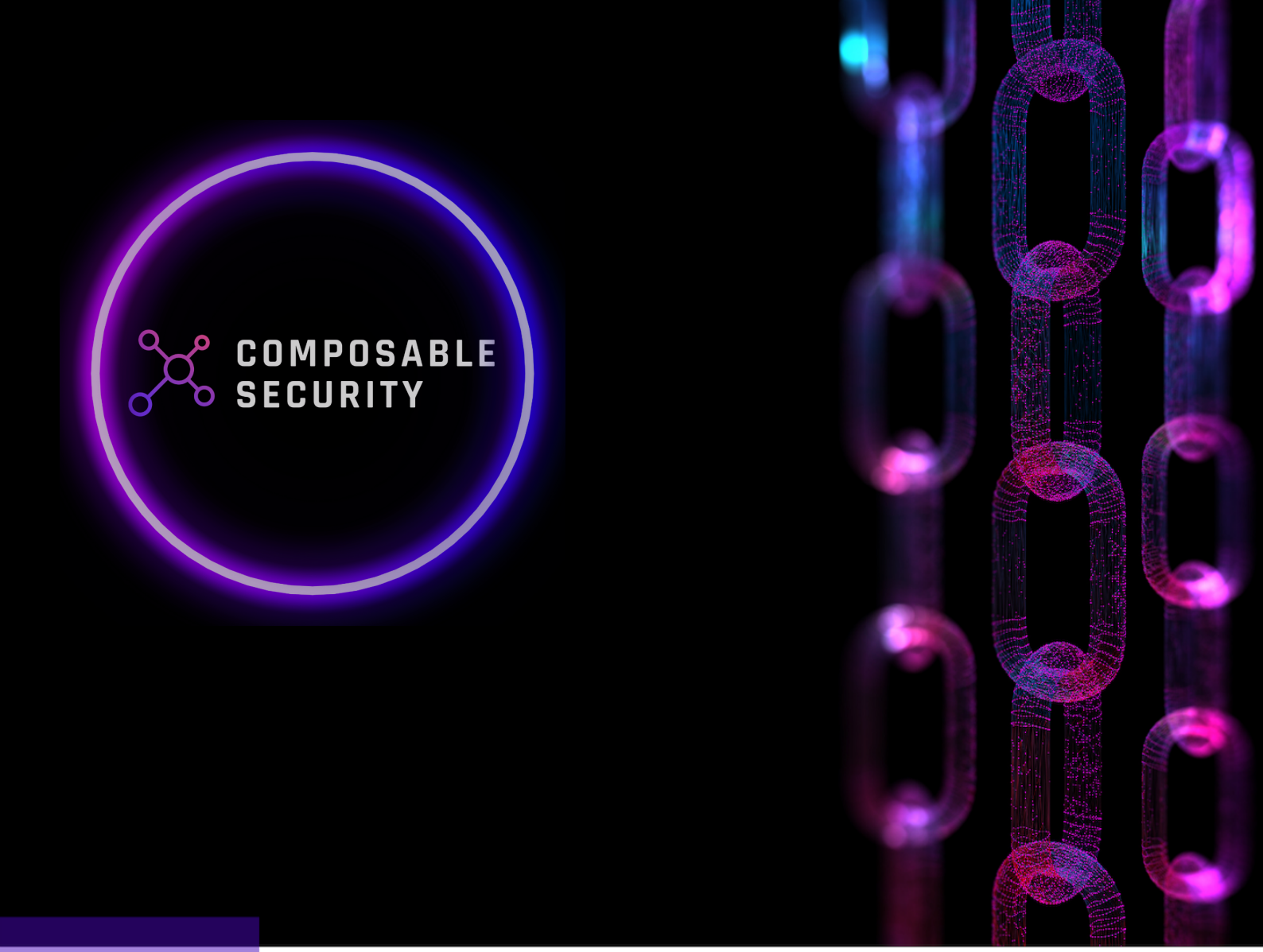


COMPOSABLE SECURITY

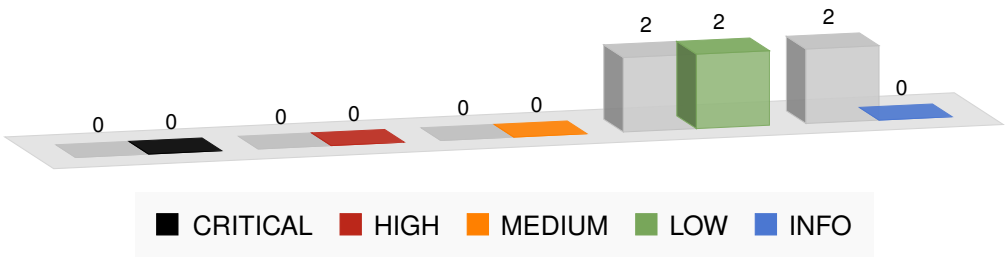
REPORT

Threat-focused security review for Lido

Prepared by: Composable Security
Report ID: LDO-6fda87f2
Test time period: 2025-06-09 - 2025-07-10
Retest time period: 2025-08-20 - 2025-08-28
Report date: 2025-08-21
Version: 1.2
Visit: composable-security.com

Contents

1. Retest summary (2025-08-28)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team participated in a one-time iteration to verify if the vulnerabilities detected during the tests (between 2025-06-09 and 2025-07-10) were correctly removed and no longer appear in the code.

The current status of detected issues is as follows:

- all **low** vulnerabilities have been acknowledged. The Lido team responses have been added in their retest descriptions.
- all security **recommendations** were implemented.

1.2. Scope

The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/lidofinance/lido-oracle
CommitID: ec19c76b5511ec48f0cfa51b9f216240e9373471

1.3. Deployments

After the security review conducted, we verified that the Dockerfile used to build an image uses the source code reviewed during the retest.

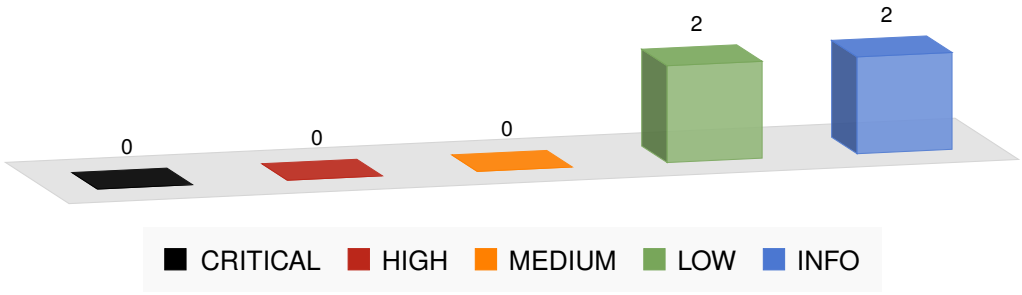
The published image (6.0.1):
sha256:f7d5f06c0b5774f09d85578a42a155cbcacdc305e0c5572ed7eaa8a6769d4077
corresponds to the commitID: **54434f46ab6681785451db88d03677036e4dd07b** which includes changes compared to the retested commit, but those changes do not influence the

security of the Oracle source code.

2. Current findings status

ID	Severity	Vulnerability	Status
LDO-6fda87f2-L01	LOW	Unfair validator performance check	ACKNOWLEDGED
LDO-6fda87f2-L02	LOW	Avoiding strikes penalty	ACKNOWLEDGED
ID	**Severity**	**Recommendation**	**Status**
LDO-6fda87f2-R01	INFO	Remove deprecated variable and constant names	IMPLEMENTED
LDO-6fda87f2-R02	INFO	Use zero-length proofs for rewards and strikes trees	IMPLEMENTED

3. Security review summary (2025-07-10)



3.1. Client project

The Oracle project is a core component of the Lido staking ecosystem, responsible for aggregating, verifying, and relaying critical operational data that supports the protocol's decentralized staking mechanism.

Below is an overview of its key components:

- **Accounting Oracle:** Focuses on tracking staking rewards, fees, and other financial metrics. This module is crucial for ensuring that the economic aspects of the protocol remain consistent and transparent.
- **Validators Exit Bus Oracle:** VEBO is an oracle that ejects Lido validators when the protocol requires additional funds to process user withdrawals. This component ensures that any changes in validator status are accurately reflected by generating requests to exit validators to fulfill the withdrawals demand.
- **CSM Oracle:** Integrates with Community Staking Module to report module-specific metrics, that is the rewards generated by CSM validators.

The changes introduced in **Oracle V6 upgrade** are following:

- Chore
  - Clean up pre-electra code
  - Update deps
- Validator Exit Bus Oracle:
  - Update validator exit order predicates
  - Remove usage last requested indexes
- Accounting Oracle
  - Remove stuck keys reporting
- CSM Oracle
  - Modified CSOracle module to support new CSM v2 contract architecture:

    - Introduced CSParametersRegistryContract with values used to calculate performance and strikes
    - Introduced CSStrikesContract for delivering reported strikes
  - Modified distribution mechanism
    - Using reward share
    - Added rebate calculation
    - Removed stucks from calculation
  - Modified performance calculation mechanisms (handling validator duties (attestation, proposal, sync))
  - Refactored state management in CSM module to support multi-frame epoch processing
  - Implemented strike tracking based on validator perfomance

3.2. Results

The **Lido** engaged Composable Security to review security of **Oracle V6 upgrade**. Composable Security conducted this assessment over 4 person-week with 2 engineers.

The summary of findings is as follows:

- There were **2** vulnerabilities with a **low** impact on risk were identified. Their potential consequences are:
  - Potential non-penalization of the validator (submission of a strike) or forfeiture of rewards. (LDO-6fda87f2-L01).
  - Avoiding incurring a strikes penalty (LDO-6fda87f2-L02).
- 2 **recommendations** have been proposed that can improve overall security and help implement best practice.
- The team was very involved in the audit and helped analyze potential attack scenarios. They took care of preparing the detailed documentation and provided answers to all the questions during the security review.

Composable Security recommends that **Lido** complete the following:

- Address all reported issues.
- Extend peer reviews with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.

3.3. Scope

The scope of the tests included selected contracts from the following repository.

GitHub repository: https://github.com/lidofinance/lido-oracle

CommitID: 6fda87f2cea606e05ca3c80a76f0b852bd244af4

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Lido** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **Lido** repository.

GitHub repository:
https://github.com/lidofinance/lido-oracle
CommitID: 6fda87f2cea606e05ca3c80a76f0b852bd244af4

Files in scope:

```
src
├── __init__.py
├── constants.py
├── main.py
├── metrics/*.py
├── modules/*.py
├── providers/*.py
├── services/*.py
├── types.py
├── utils/*.py
├── variables.py
└── web3py/*.py
```

Documentation:

- Audit scope
- Triggerable Withdrawals Technical Specification (on-chain)
- Oracle V6 Technical Specification (off-chain)
- CSM V2

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Report's submitted data, including:
  - stETH/ETH ratio components,
  - Withdrawal requests queue border,
  - Ejected validators,
- Ether managed by the protocol,
- Bonds,
- Protocols availability.

Threats and potential attackers goals:

- Avoiding strikes.
- Avoid penalty from strikes.
- Getting a higher reward.
- Manipulation of validator's performance.
- Manipulation of the stETH/ETH rate to steal ETH.
- Bonds theft.
- Running validators without required bond.
- Bypassing business logic limits (e.g. the withdrawal request delay).
- Denial of Service (e.g. due to Oracle malfunction).

Potential vulnerable and attack scenarios to achieve the indicated attacker's goals:

- Excluding validators from the total ETH balance calculation.
- Overestimation of performance.
- Including the same action multiple times in the performance calculation.
- Front-running deposits to preset withdrawal credentials.
- Retrieving on-chain data from incorrect block via archive node.
- Intentional exit of validator by its operator to avoid penalties.
- Take advantage of arithmetic errors.
- Incorrect selection of validators to be ejected (e.g. re-ejecting validators).
- Keeping ETH on validators forced to exit.
- Missing a report for one or more frames.
- Missing ETH balances of validators not yet added to registry but already submitted via Deposit contract.
- Influence or bypass the business logic of the system.
- Inconsistency with documentation.

- Design issues.
- Uncaught exceptions.

4.4. Testing methodology

Security review was performed using the following methods:

- Q&A sessions with the **Lido** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests.
- **Manual review of the code.**

4.5. Disclaimer

Security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[LDO-6fda87f2-L01] Unfair validator performance check

LOW ACKNOWLEDGED

Retest (2025-08-20)

The Lido team acknowledged the issue with the following comment:
One of the fundamental goals of the CSM parameterization system is to allow different types of Node Operators—such as ICS (Identified Community Staker)—to have customized performance weight configurations. We cannot use the same weights to compute both the network average and the validator performance, as that would imply the entire network is composed solely of ICS-type operators. This assumption directly conflicts with the business logic of CSM and the smoothing pool design.
Therefore, using distinct weight profiles for the network and the validator is both intentional and necessary to maintain accurate and fair performance evaluations across different types.

Affected files

- distribution.py#L130
- distribution.py#L185-L190
- distribution.py#L228

Description

In order for CSM validators to participate in the reward distribution process and avoid long-term penalties, they must meet specific performance criteria. The performance is calculated as a weighted average of three factors:

- attestations,
- block proposals,
- sync committee participation.

The average is then compared with the collective performance of all Ethereum validators.

Node Operators may have varying weights. However, **their performance is always compared to the average performance of the network, which is determined using standard weights**. Notably, a validator's performance can fall below the network's average performance by a certain threshold defined for each validator. For instance, if the network's average performance is 99%, the threshold is 3%, and a validator's performance is 97%, that

validator will not be penalized, as `99%-3% < 97%`.

This issue can allow some validators to **escape penalties** despite low performance or to **be excluded from reward distribution**.

Vulnerable scenario

Consider a scenario in which two factors are used to calculate performance, with default weights of 90% and 10%. The following steps illustrate how a validator might miss out on rewards:

① The network performance for both factors stands at 95%, resulting in an average weighted performance of `90% * 95% + 10% * 95% = 95%`.
② The validator has a threshold of 2%, with weights of 80% and 20% yielding performances of 90% and 100% respectively, leading to a combined performance of `92%`.
③ In this case, the validator is penalized because `92% < 95%-2%`. However, had the same weights been applied to compute the network performance, it would have been 91%, and the validator would have been eligible for a reward.

Result: Potential non-penalization of the validator (submission of a strike) or loss of rewards.

Recommendation

The weights used for calculating average network performance must be consistent with those used for assessing the validator's performance.

References

1. SCSVS G4: Business logic

[LDO-6fda87f2-L02] Avoiding strikes penalty

LOW ACKNOWLEDGED

Retest (2025-08-20)

The Lido team acknowledged the issue with the following comment:
If a Node Operator independently initiates a timely exit of an underperforming validator before reaching the penalty threshold, that outcome is not only acceptable but preferable, as it aligns with the goal of maintaining validator quality without incurring unnecessary overhead from protocol-side enforcement. There are some details in the CSM v2 tech desing document.

Affected files

- CSStrikes.sol#L221-L245

Description

The strikes mechanism is implemented to penalize validators who fail to meet required performance standards over an extended period. When a validator's performance falls below a predetermined threshold (based on the average performance of the network) a strike is assigned to that validator.

If a validator accumulates a certain number of strikes within a specified timeframe, it may be ejected, resulting in a penalty for the associated Node Operator.

An issue arises from the timing of the CSM report, which is published every 28 days. A Node Operator can withdraw from the Consensus Layer roughly one day after initiating a voluntary exit, assuming there are no delays due to an exit queue or churn limits. This discrepancy allows a Node Operator to exit the validator before the updated penalty information is reflected on-chain.

Vulnerable scenario

Consider a situation where the penalty for strikes requires a total of 3 strikes in a 6 months. The following series of events may occur:

① The validator performs poorly for 2 consecutive months.
② In the third month, the Node Operator realizes that the validator's performance continues to decline and anticipates receiving a penalty in the upcoming report as the next strike will trigger the threshold.
③ The Node Operator then voluntarily exits the validator, with the withdrawal epoch scheduled to occur in approximately one day.
④ The Node Operator submits the withdrawal proof and successfully withdraws the entire amount, avoiding the strike penalty.

Result: The Node Operator avoids incurring a strikes penalty.

Recommendation

Implement measures to block withdrawals from a validator until the next report is published if it is likely to reach a strike penalty.

References

1. SCSVS G1: Architecture, design and threat modeling

6. Recommendations

[LDO-6fda87f2-R01] Remove deprecated variable and constant names

INFO IMPLEMENTED

Retest (2025-08-28)

The recommendation has been implemented as recommended.

Description

The changes introduced in V6 upgrade have deprecated stuck validators, but some comments, constants, and variables have not been updated. This may cause confusion.

Recommendation

Add the following changes:

- Change `STUCK_OR_EXITED_VALS_COUNT` to `EXITED_VALS_COUNT`.
- Change `stuckOrExitedValsCount` to `exitedValsCount`.
- Update comments: 1, 2.
- Remove fields related to stuck validators: lido_validators.py#L64-L103.

References

1. SCSVS G11: Code clarity

[LDO-6fda87f2-R02] Use zero-length proofs for rewards and strikes trees

INFO IMPLEMENTED

Retest (2025-08-28)

The recommendation has been implemented as recommended.

Description

The `make_strikes_tree` function adds a stone (empty item) to the tree if there is 1 element to make sure that this element is not the root.

That's because - as the comment says - *CSModule contract skips pulling rewards if the proof's length is zero*. However, the strikes tree is consumed by `CSStrikes` contract, which accepts zero-length proofs.

Recommendation

Do not add a stone to the strikes tree when there is only one item.

References

1. SCSVS G1: Architecture, design and threat modeling
2. SCSVS G11: Code clarity

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com