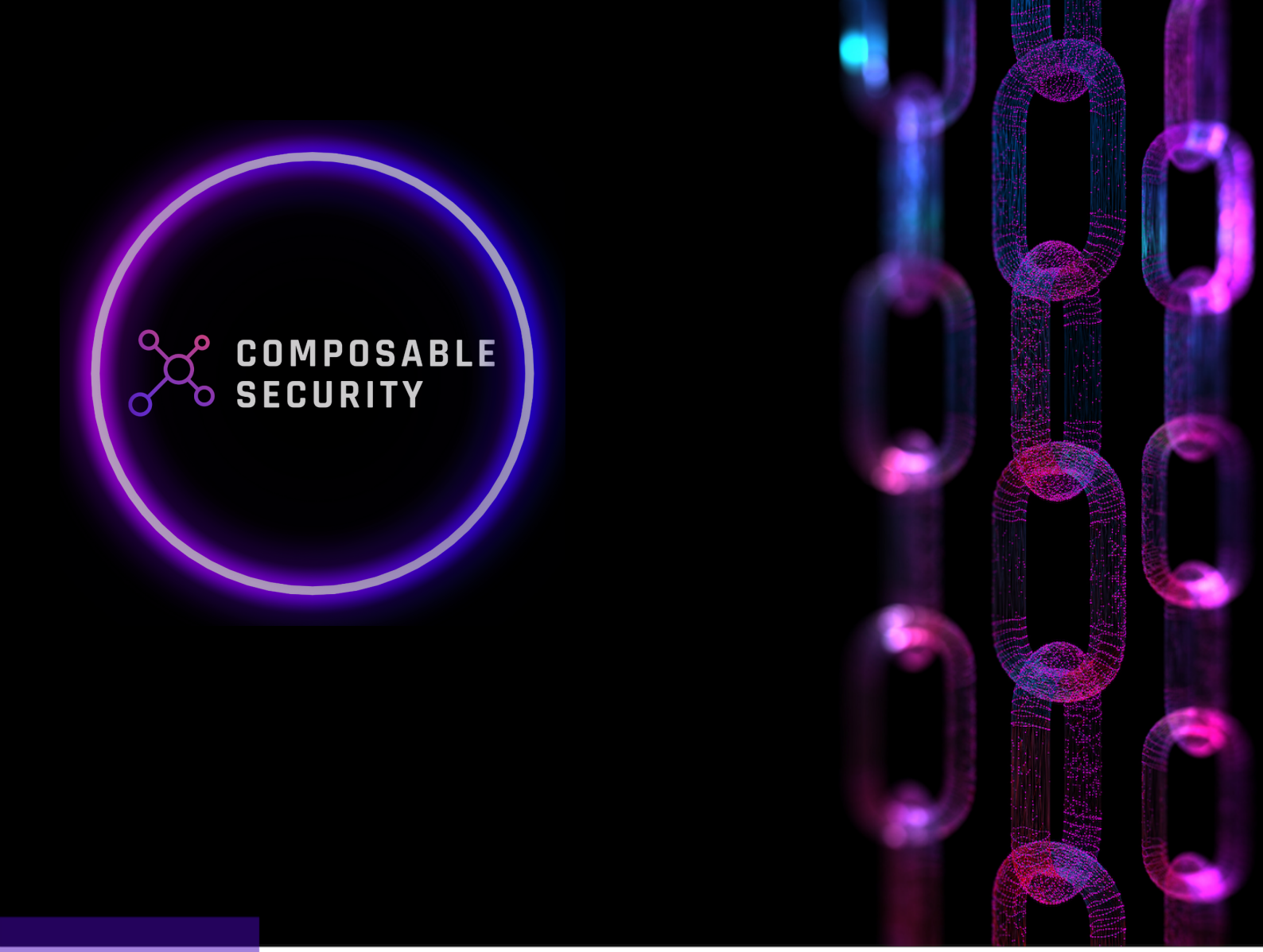


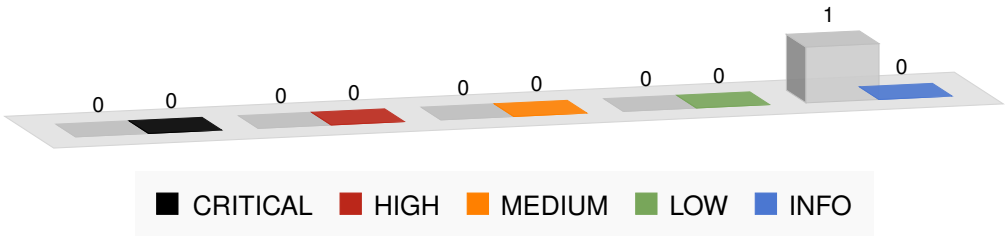
REPORT

Security review for Lido

Prepared by: Composable Security
Report ID: LDO-33bd3e47
Test time period: 2026-02-04 - 2026-02-06
Retest time period: 2026-03-04 - 2025-03-04
Report date: 2026-03-04
Version: 1.1
Visit: composable-security.com

Contents

1. Retest summary (2025-03-04)



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 2026-02-04 and 2026-02-06) were correctly removed and no longer appear in the code.

The current status of detected issues is as follows:

- The security **recommendation** has been implemented.

1.2. Scope

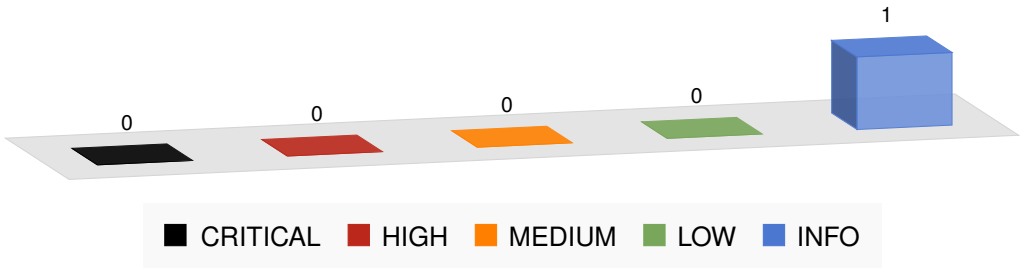
The retest scope included the files, on a different commit in the same repository.

GitHub repository: https://github.com/lidofinance/lido-oracle/
CommitID: b2e92969fd35b2a1838667a290ded649ca33fbf5

2. Current findings status

ID	Severity	Recommendation	Status
LDO-33bd3e47-R01	INFO	Validate negative slot value	IMPLEMENTED

3. Security review summary (2026-02-06)



3.1. Client project

Lido is the largest decentralized staking protocol that allows users to stake ETH and receive liquid staking derivative in return. The project successfully manages over $20B TVL supporting Ethereum growth and security since 2020.

The **Lido Oracle** project is a daemon that monitors the Lido staking protocol across the Execution and Consensus layers and submits regular update reports to Lido smart contracts.

It is responsible for aggregating, verifying, and relaying critical operational data that support the protocol's decentralized staking mechanism.

It has three modules:

- **Accounting**: Handles TVL updates, node operator rewards, validator status, withdrawal requests, and bunker mode (24-hour frames)
- **Ejector**: Initiates validator ejection requests to fund withdrawals (8-hour frames)
- **CSM**: Collects and reports validator attestation rates for node operators (28-day frames)

The oracle runs in Docker containers, requires archive nodes for both Execution and Consensus layers, integrates with a Keys API service, and can operate in dry-run or production mode (with member private keys).

In this engagement, the Composable Security team focused on **changes introduced in version 7.1.0** of Lido Oracle. The central theme is migrating vault fee calculations from block-number-based intervals to time-based (seconds/timestamps) intervals. This is a significant rearchitecture of how staking vault fees are computed.

3.2. Results

The **Lido** engaged Composable Security to review the security of **Lido Oracle V7.1.0**. Composable Security conducted this assessment over 3 person-days with 2 engineers.

The summary of findings is as follows:

- No vulnerabilities were identified.
- 1 **recommendation** has been proposed that can improve overall security and help implement best practice.
- The team was very engaged and communication was excellent, including ongoing vulnerability triaging and solution consultations.

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

- Address all reported issues.
- Extend unit tests with scenarios that cover detected issues where possible.
- Consider whether the detected issues may exist in other places (or ongoing projects) that have not been detected during engagement.

3.3. Scope

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

GitHub repository: https://github.com/lidofinance/lido-oracle/
CommitID: 33bd3e47c0f709d1c6524f41bf4a9a0874a29feb

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 security review on the following:

- Perform a tailored threat analysis.
- 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 scope includes review of the fix that was implemented in this PR: #838.

GitHub repository:
https://github.com/lidofinance/lido-oracle/
CommitID: 33bd3e47c0f709d1c6524f41bf4a9a0874a29feb

Files in scope:

```
.
└── ./src
    ├── ./src/constants.py
    ├── ./src/modules
    │   └── ./src/modules/accounting
    │       └── ./src/modules/accounting/events.py
    ├── ./src/services
    │   └── ./src/services/staking_vaults.py
    ├── ./src/utils
    │   └── ./src/utils/block.py
    └── ./src/variables.py
```

Documentation:

- Issue description
- Lido Staking Vaults (stVaults): Technical Design and Architecture

4.3. Threat analysis

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

Key assets that require protection:

- Vault fee integrity.
- Report's submitted data.
- Protocols availability.

Threats and potential attackers goals:

- Manipulation of fees between vaults.
- Block timestamps manipulation.
- Invalid retrieval and manipulation of vaults' balances.
- Denial of Service (e.g. due to Oracle malfunction).

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

- RPC timestamp fabrication.
- Applying vault data different that submitted in the report.
- Fee manipulation via disconnection.
- Using deprecated or too fresh vault data when building the report.
- Retrieving on-chain data from incorrect block via archive node.
- Intentional exit of validator by its operator to avoid fees.
- Missing events when calculating liquidity fees.
- Invalid chain configuration.
- Take advantage of arithmetic errors.
- 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.
- **Manual review of the code.**

4.5. Deployments

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

The published image with the manifest digest **sha256:3dffe7e885a01961777d3cdebe1d8d0bdb988e90a14e44d18ae33b6ccb230993** corresponds to the commitID: **b2e92969fd35b2a1838667a290ded649ca33fbf5**.

4.6. 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 complex 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. Recommendations

[LDO-33bd3e47-R01] Validate negative slot value

INFO IMPLEMENTED

Retest (2026-03-04)

The recommendation has been implemented as recommended.
Additionally, the slot starting from which the events are collected has been shifted back one frame because `initial_epoch` is the epoch when the oracle starts producing reports.
There is still an edge case where `initial_epoch` is set to 0 and the `from_block` parameter is set to 1, missing events from the epoch 0. However, the protocol needs to be deployed on the devnet, which will take at least one block. Therefore, it is impossible to have any meaningful events in the very first block.

Description

When `frame_config.initial_epoch` is 0 and there's no previous Oracle report, the calculation `prev_ref_slot = SlotNumber(frame_config.initial_epoch * chain_config.slots_per_epoch - 1)` results in `SlotNumber(-1)`. This negative slot value causes multiple issues:

1. **RPC call failure**: The negative slot is passed to `get_blockstamp`, which calls `get_prev_non_missed_slot` and then `get_non_missed_slot_header`. The Beacon API defines slots as `uint64`, so slot `-1` is invalid. Consensus layer clients will likely return a 400 Bad Request, which may crash the oracle if not properly handled. Otherwise, the client would return 404 Not Found and the slot would be marked as missed. Then, when slot 0 is retrieved, the Oracle would try to get its parent root which does. not exist.
1. **Incorrect timestamp calculation**: Even if the error is caught, `_get_report_timestamp` with slot `-1` returns `genesis_time - 12` (a timestamp before genesis), inflating `report_interval_seconds` by 12 seconds and leading to incorrect fee calculations.

This primarily affects devnet/testnet environments where `initial_epoch` may be 0, but it represents a liveness issue that could prevent the oracle from functioning correctly on fresh deployments.

Recommendation

Add validation to ensure `prev_ref_slot` is never negative. When `initial_epoch` is 0 and there's no previous report, use slot 0 as the minimum value instead of `-1`.

References

1. SCSVS G7: Arithmetic

6. 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 blockchain projects. 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

7. Long-term best practices

7.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

7.2. Perform threat modeling

Before implementing or introducing changes, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

7.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

7.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

7.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com