



Security Assessment Report



Lido V3 Oracle

December 2025

Prepared for Lido

Table of contents

Project Summary.....	3
Project Scope.....	3
Project Overview.....	3
Protocol Overview.....	5
Security Considerations.....	7
Audit Goals.....	7
Key Focus Areas.....	8
Findings Summary.....	10
Severity Matrix.....	10
Detailed Findings.....	11
High Severity Issues.....	13
H-01 - Negative APR Leads to DOS.....	13
H-02 - Double Counting of end-of-frame block When Calculating Liquidity Fees.....	15
Medium Severity Issues.....	17
M-01 - Missing IPFS CID Verification.....	17
M-02 - Missing Fees Calculation on VaultConnectionUpdated Event.....	19
Low Severity Issues.....	21
L-01 - Merkle Tree Validation Missing Leaf Count.....	21
L-02 - Lack of Validation in Event Handling.....	23
L-03 - Kubo RPC API Bound to 0.0.0.0.....	26
L-04 - Prometheus Server Bound to 0.0.0.0.....	28
L-05 - Missing IPFS Data Validation.....	30
L-06 - Double Counting of end-of-frame block When Calculating Liquidity Fees (Partial Fix).....	33
L-07 - Storacha Upload URL Leak in Oracle Logs.....	35
Informational Severity Issues.....	37
I-01 - oz_merkle_tree Proof Verification on Internal Node (Informational).....	37
I-02 - Division by Zero in get_steth_by_shares Function.....	38
I-03 - Consensus-critical contract methods should use finalized block by default.....	39
I-04 - Incorrect Events Order Leads to Wrong Fees Calculation.....	40
I-05 - Oracle HTTP Provider follows HTTP Redirects (30x).....	42
Disclaimer.....	42
About Certora.....	43

Project Summary

Project Scope

Project Name	Repository Link	Branch/PR/Commit Hash	Platform
Lido Oracle	lidofinance/lido-oracle/	Branch: feat/vaults-pectra → develop PR 815 39a1c5b	Off-chain

Project Overview

This document describes the manual code review of **Lido Oracle**. The work was undertaken from **July 31 2025 to December 10 2025**

Last audited commit: `39a1c5b8a5e1a90debb2f9eebac6e6588767a473`

The following script list is included in our scope:

- `src/constants.py`
- `src/main.py`
- `src/metrics/prometheus/accounting.py`
- `src/modules/accounting/accounting.py`
- `src/modules/accounting/events.py`
- `src/modules/accounting/types.py`
- `src/modules/submodules/consensus.py`
- `src/modules/submodules/types.py`
- `src/providers/consensus/client.py`
- `src/providers/consensus/types.py`
- `src/providers/execution/contracts/accounting.py`
- `src/providers/execution/contracts/lazy_oracle.py`
- `src/providers/execution/contracts/lido.py`
- `src/providers/execution/contracts/lido_locator.py`
- `src/providers/execution/contracts/oracle_daemon_config.py`
- `src/providers/execution/contracts/staking_router.py`
- `src/providers/execution/contracts/vault_hub.py`
- `src/services/bunker.py`



- `src/services/staking_vaults.py`
- `src/utils/apr.py`
- `src/utils/validator_state.py`
- `src/variables.py`
- `src/web3py/extensions/contracts.py`
- `src/web3py/types.py`
- `src/providers/ipfs/storacha.py`
- `src/providers/ipfs/lido_ipfs.py`
- `src/providers/ipfs/types.py`
- `src/providers/http_provider.py`
- `src/modules/checks/suites/ipfs.py`
- `src/providers/ipfs/pinata.py`
- `src/web3py/extensions/ipfs.py`

During the manual audit, the Certora team discovered the bugs listed on the following page.



Protocol Overview

The oracle is the operational backbone of the Lido protocol on Ethereum. It acts as the bridge between the Ethereum consensus layer, execution layer, and the Lido smart contracts, ensuring accurate state reporting, reward distribution, and protocol safety mechanisms.

The oracle operates through a committee-based consensus mechanism where multiple oracle nodes must agree on protocol state before updates are committed to the blockchain. This ensures:

- Fault Tolerance: Protection against malicious or faulty oracle nodes
- Data Integrity: Multiple validations before state changes
- Network Resilience: Redundancy across multiple oracle instances

The oracle collects off-chain data about validator performance, balances in the consensus Layer of Ethereum, and vault states in Execution Layer, then submits periodic reports on-chain (on Execution Layer), the idea is to track and update :

- Total Value Locked (TVL) Calculation: Real-time computation of protocol assets across all vaults and validators
- Share Rate Updates: Maintaining accurate stETH:ETH exchange rates
- Multi-Vault Support: Handles complex vault structures with different fee models
- Slashing Reserve Management: Tracks and manages slashing penalties across the validator set
- Performance Attribution: Detailed tracking of validator performance for reward calculations
- Protocol health and rebasing (stETH supply adjustments)
- Validator exits
- Vault System (V3 Addition) :
 - individual vault performance tracking
 - separate accounting for vault validators
 - performance based fee calculations
 - merkle tree building
- Withdrawal Management (withdrawals work in queue so it manages what to finalize, how much eth it requires, which validators should exit to get access to ETH to finalize etc)



- Node Operator Performance and Rewards Distribution (NO = handling 1 or more validators on consensus layer): the oracle tracks attestation performance, rewards/penalties, limits and keys availability and distributes rewards

Security Considerations

The team considered risks and attack vectors that could potentially compromise the integrity, availability, and financial security of the Lido oracle system, with particular focus on the stVaults feature and its associated vault management operations. The oracle takes a critical role in managing stVaults data, including vault total values, fee calculations, liability shares, and slashing reserves.

The analysis encompasses both technical vulnerabilities in the oracle's stVaults data processing and broader systemic risks arising from the oracle's position as the authoritative source for vault state information. Specific attention was given to the merkle tree validation processes for vault data, the stVaults service integration, and the cross-vault consistency mechanisms, as these represent critical attack surfaces with high potential impact on vault operations and user funds.

Specifically, the team looked for risks regarding:

- Errors and/or malicious actors that can lead to:
 - Incorrect data to be reported by the Oracle
 - Denial of service on the Oracle, by preventing reports from being generated, preventing Oracles of reaching quorum, or crashing the Oracles
 - Possibility to bypass security mitigations
- While auditing, the following types of attackers were considered:
 - External attackers that target the Oracle directly
 - Malicious vault owners.
 - Malicious stakers.
 - Malicious node operators and/or validators.
 - A single malicious Oracle.
 - A quorum of malicious Oracles, with regard to bypassing mitigations.
 - Malicious public IPFS gateways.



Audit Goals

The primary goal of this security audit is to comprehensively assess the security posture of the Lido oracle system, with particular emphasis on the stVaults feature and its associated vault management operations. The audit aims to identify potential vulnerabilities, attack vectors, and security weaknesses that could compromise the integrity, availability, and financial security of the stVaults ecosystem introduced in [PR-665](#).

1. Enumerate the attack surfaces related to newly introduced code.
2. Find specific risks and attack vectors which could realistically lead to incorrect behavior and/or denial of service of the Oracle due to newly introduced code.
3. Suggest limited and accurate fixes for such risks.

Key Focus Areas

The audit's scope was limited to [PR#665](#) (stVaults Oracle), and this was the main focus area of the audit. Additional code (outside of the PR) that is directly related to the stVaults addition was reviewed for context, including other parts of the Oracle, and relevant smart contracts.

Specifically, the team looked into the following areas of interest:

- IPFS usage
- Accounting, calculations and report generation
- Handling of transactions and events originating from the blockchain, with emphasis on events that can be triggered by malicious actors
- stVaults Data Integrity Assessment

Objective: Evaluate the security of vault data processing and validation mechanisms

Scope: Merkle tree validation, vault data submission, and cross-vault consistency checks

Focus Areas:

- Merkle tree structure validation for vault data
 - Cross-vault invariant enforcement
 - Bad debt handling and socialization mechanisms
- stVaults Economic Security Analysis

Objective: Evaluate the security of vault economic operations and fee mechanisms

Scope: Fee calculations, reward distributions, and vault health assessments

Focus Areas:

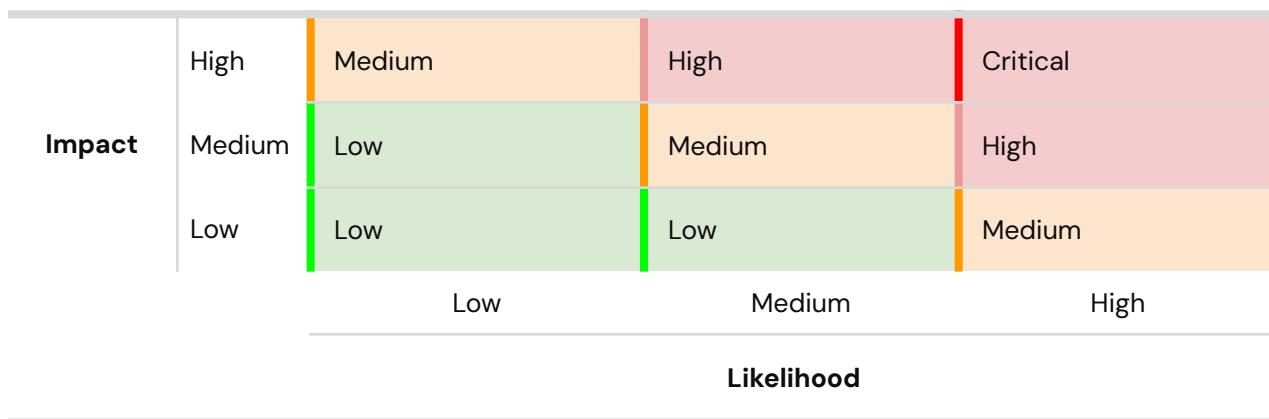
- Vault fee manipulation resistance
- Reward distribution algorithm security
- Vault health assessment accuracy
- Economic parameter manipulation prevention

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	2	2	2
Medium	2	2	2
Low	7	5	2
Informational	5	3	1
Total	16	11	7

Severity Matrix



Detailed Findings

ID	Title	Severity	Status
H-01	Negative APR Leads to DOS	High	Fixed
H-02	Double Counting of end-of-frame block When Calculating Liquidity Fees	High	Fixed
M-01	Missing IPFS CID Verification	Medium	Fixed
M-02	Missing Fees Calculation on VaultConnectionUpdated Event	Medium	Fixed
L-01	Merkle Tree Validation Missing Leaf Count	Low	Acknowledged
L-02	Lack of Validation in Event Handling	Low	Acknowledged
L-03	Kubo RPC API Bound to 0.0.0.0	Low	Acknowledged
L-04	Prometheus Server Bound to 0.0.0.0	Low	Acknowledged
L-05	Missing IPFS Data Validation	Low	Acknowledged
L-06	Double Counting of end-of-frame block When Calculating Liquidity Fees (Partial Fix)	Low	Fixed
L-07	Storacha Upload URL Leak in	Low	Fixed

Oracle Logs			
I-01	oz_merkle_tree Proof Verification on Internal Node (Informational)	Informational	Acknowledged
I-02	Division by Zero in get_steth_by_shares Function	Informational	Acknowledged
I-03	Consensus-critical contract methods should use finalized block by default	Informational	Acknowledged
I-04	Incorrect Events Order Leads to Wrong Fees Calculation	Informational	Fixed
I-05	Oracle HTTP Provider follows HTTP Redirects (30x)	Informational	Acknowledged

High Severity Issues

H-01 – Negative APR Leads to DOS

Severity: High	Impact: Medium	Likelihood: High
Files: src/services/staking_vaults.py, src/utils/apr.py	Status: Fixed	Violated Property: N/A

Description:

`calculate_steth_apr()` can return a negative APR (when `pre_rate > post_rate`). In that case, `get_core_apr_ratio()` would return a negative core apr ratio the negative `core_apr_ratio` is then used in `get_vaults_fees() -> calc_fee_value()`

staking_vaults.py::`get_vaults_fees()`:622

```
Python
infra_fee=int(vault_infrastructure_fee.to_integral_value(ROUND_UP)),
reservation_fee=int(vault_reservation_liquidity_fee.to_integral_value(ROUND_UP)),
liquidity_fee=int(vault_liquidity_fee.to_integral_value(ROUND_UP)),
```

The fees calculated here would be negative, and `ROUND_UP` would round small negatives away from zero (-0.1 rounds to -1).

These fees are later used in `build_tree_data()` to build the Merkle Tree:

staking_vaults.py::`build_tree_data()`:263

```
Python
tree_data.append(
(
    vault_address,
```



```
        Wei(vaults_total_values[vault_address]),
        vaults_fees[vault_address].total(),
        vault.liability_shares,
        vaults_slashing_reserve.get(vault_address, 0),
    )
)
```

The Merkle trees use uint256 values, but in this case a negative value (`vaults_fees[vault_address].total()`) is passed.

Tree building process fails, resulting in failed reporting. These failures can happen ‘naturally’, or be intentionally triggered by malicious actors aiming to DOS the Oracle.

Recommendations

- Round negative apr_ratio values to 0
- Verify calc_fee_value() doesn't return negative values
- Consider adding a regression test that covers this case (current tests check for negative apr_ratios calculation correctness, but nothing else)

Lido's response: Fixed in [7582f00](#)

Fix Review: Fixed. Negative APR is clamped to zero.

`lido-oracle/src/utils/apr.py::calculate_gross_core_apr()`

Python

```
rate_diff: Decimal = post_share_rate_no_fees - pre_share_rate

if rate_diff < 0:
    return Decimal(0)
```

H-02 – Double Counting of end-of-frame block When Calculating Liquidity Fees

Severity: High	Impact: Medium	Likelihood: High
Files: src/services/staking_vaults.py	Status: Fixed	Violated Property: N/A

Description:

In calc_liquidity_fee events are used to calculate prev_liability_shares from the liability_shares of the current block by applying the events to liability_shares in reverse order.

The prev_liability_shares from the last report reflects the EL/CL state after the transactions in the last block of the frame.

staking_vaults.py::get_start_point_for_fee_calculations:486

```
Python
_get_start_point_for_fee_calculations(...)

...
ref_block = get_blockstamp(self.w3.cc, last_processing_ref_slot, ...)
prev_block_number = ref_block.block_number
```

When fetching the events, the range includes the prev_block already used in the last report's calculations.

events.py::get_events_in_range():67

```
Python
def get_events_in_range(event: ContractEvent, l_block: BlockNumber, r_block: BlockNumber) ->
Iterator[EventData]:
...
for e in event.get_logs(from_block=l_block, to_block=to_block):
```



```
if not l_block <= e["blockNumber"] <= to_block:  
    raise InconsistentEvents  
    yield e  
    ...
```

So events are counted twice: once in the report(as seen in the final value in the report) and once again in the reverse event calculation.

This leads to tripping an assertion and failing the report here:

staking_vaults.py::get_vaults_fees():616

Python

```
if prev_liability_shares != liability_shares:  
    raise ValueError(  
        f"Wrong liability shares by vault {vault_address}. Actual  
{liability_shares} != Expected {prev_liability_shares}"  
    )
```

Recommendations

- when fetching the events for calculation, omit the prev_block, i.e. use l_block+1 in get_events_in_range

Lido's response: Fixed in [7582f00](#)

Fix Review: Fixed. _get_start_point_for_fee_calculations now returns ref_block.block_number + 1

Medium Severity Issues

M-01 – Missing IPFS CID Verification

Severity: Medium	Impact: High	Likelihood: Low
Files: src/services/staking_vaults.py	Status: Fixed	Violated Property: N/A

Description:

The code fetches the report based on tree_cid and immediately parses it, without verifying that the fetched report data actually matches the CID.

This is fine when using a trusted IPFS provider. However, the Oracle can be configured to use a public IPFS gateway as a fallback to Kubo and Pinata:

In staking_vaults.py::get_vault_report():369

Python

```
def get_vault_report(self, tree_cid: str) -> StakingVaultIpfsReport:
    bb = self.w3.ipfs.fetch(CID(tree_cid))
    return StakingVaultIpfsReport.parse_merkle_tree_data(bb)
```

main.py::ipfs_providers:142

Python

```
yield PublicIPFS(timeout=variables.HTTP_REQUEST_TIMEOUT_IPFS)
```

Impact:

The core data within the Merkle Tree remains safe. However, values outside of the Merkle Tree - extraValues and others, could be tampered with.



Recommendations:

- The current prioritization is correct: Kubo first, Pinata next, and the Public IPFS as a fallback.
- If Public IPFS remains an option, add a verification of the fetched report data against the provided CID - after fetching the report data and before parsing it.

Lido's response: Fixed in [7582f00](#)

Fix Review: IPFS CID Validation Implemented.

M-02 – Missing Fees Calculation on VaultConnectionUpdated Event

Severity: Medium	Impact: High	Likelihood: Low
Files: VaultHub.sol, src/services/staking_vault s.py	Status: Fixed	Violated Property: N/A

Description:

VaultConnectionUpdated event includes fields related to fee changes (infraFeeBP, liquidityFeeBP, reservationFeeBP)

However, there is no code that handles these types of events in the Oracle (particularly within calc_liquidity_fee())

VaultHub.sol::VaultConnectionUpdated():1577

Python

```
event VaultConnectionUpdated(  
    address indexed vault,  
    uint256 shareLimit,  
    uint256 reserveRatioBP,  
    uint256 forcedRebalanceThresholdBP,  
    uint256 infraFeeBP,  
    uint256 liquidityFeeBP,  
    uint256 reservationFeeBP  
)
```

Impact:

The calculated fees could be incorrect.



Recommendations:

Add handling of such events during fees calculation.

Lido's response: Fixed on-chain by adding VaultFeesUpdated event to `updateConnection` method.

Fix Review: Fixed.

Low Severity Issues

L-01 – Merkle Tree Validation Missing Leaf Count

Severity: Low	Impact: Low	Likelihood: Low
Files: lido-oracle/src/services/staking_vaults.py	Status: Acknowledged	Violated Property: N/A

Description:

The oracle's `is_tree_root_valid()` function rebuilds a merkle tree from vault data but only validates that the root hash matches. It does not verify that the number of leaves in the reconstructed tree matches the number of leaves in the original tree structure. This allows attackers to manipulate vault data while maintaining the same root hash.

`lido-oracle/src/services/staking_vaults.py:373`

Python

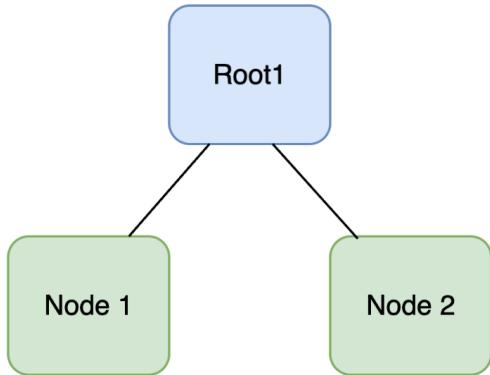
```
def is_tree_root_valid(self, expected_tree_root: str, merkle_tree: StakingVaultIpfsReport) -> bool:
    # .... rebuild the tree ...
    return merkle_tree.tree[0] == root_hex and root_hex == expected_tree_root # Leaf count check not included.
```

Tree 1 and Tree 2 in the diagram below can provide the same hash. To exploit the vulnerability, an attacker has to calculate two vault data that hashes to the same value of the original Tree1 root.

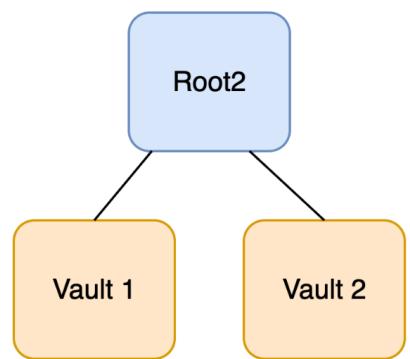
$$\text{Hash}(\text{Node1}, \text{Node2}) = \text{Hash}(\text{Vault1}, \text{Vault2})$$

`is_tree_root_valid()` function returns True for both Tree1 and Tree2.

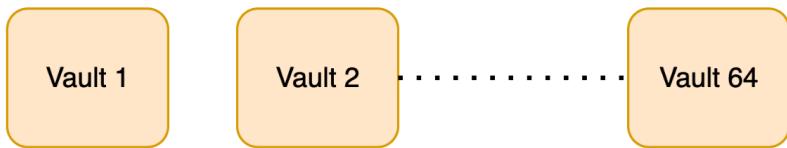
Tree 1 (Original)



Tree 2 (Malicious)



..... Internal Nodes



Recommendation:

Modify the `is_tree_root_valid()` function to validate the number of leaves.

Lido's response: Acknowledged. Creating different leaf values that still lead to the exact same root would require solving a Keccak-256 second-preimage problem.

L-02 - Lack of Validation in Event Handling

Severity: Low	Impact: Low	Likelihood: Low
Files: src/modules/accounting/events.py	Status: Acknowledged	Violated Property: N/A

Description:

Oracle's event processing system lacks proper input validation when parsing blockchain event logs. All event parsing methods directly access dictionary keys without validation, making the Oracle vulnerable to crashes when processing malformed or corrupted event data. This vulnerability can be exploited through malicious RPC node responses or network data corruption, leading to Oracle downtime and potential disruption of the Lido protocol's reporting mechanism.

src/modules/accounting/events.py:38-46

```
Python
@classmethod
def from_log(cls, log: EventData) -> "MintedSharesOnVaultEvent":
    args = log["args"]  # ← NO VALIDATION: KeyError if 'args' missing
    return cls(
        vault=args["vault"],  # ← NO VALIDATION: KeyError if 'vault' missing
        amount_of_shares=args["amountOfShares"],  # ← NO VALIDATION: KeyError/TypeError
        locked_amount=args["lockedAmount"],  # ← NO VALIDATION: KeyError/TypeError
        **cls._extract_common(log),
    )
```

src/modules/accounting/events.py:20-29

```
Python
@classmethod
def from_log(cls, log: EventData) -> "MintedSharesOnVaultEvent":
    args = log["args"]  # ← NO VALIDATION: KeyError if 'args' missing
    return cls(
        vault=args["vault"],  # ← NO VALIDATION: KeyError if 'vault' missing
        amount_of_shares=args["amountOfShares"],  # ← NO VALIDATION: KeyError/TypeError
```

```

        locked_amount=args["lockedAmount"], # ← NO VALIDATION: KeyError/TypeError
        **cls._extract_common(log),
    )

```

All Event Classes Affected

- MintedSharesOnVaultEvent.from_log() - Lines 38-46
- BurnedSharesOnVaultEvent.from_log() - Lines 55-61
- VaultFeesUpdatedEvent.from_log() - Lines 75-85
- VaultRebalancedEvent.from_log() - Lines 97-103
- BadDebtSocializedEvent.from_log() - Lines 115-121
- BadDebtWrittenOffToBeInternalizedEvent.from_log() - Lines 133-139

Exploit Scenario

Malicious RPC Node Response

1. Attacker compromises or controls an RPC node that the Oracle uses for event fetching
2. RPC node returns corrupted event data with missing or malformed fields
3. Oracle processes the corrupted event during fee calculation in calc_liquidity_fee()
4. KeyError exception crashes the Oracle when trying to access log["args"]
5. Oracle stops processing reports, disrupting the Lido protocol

Recommendations

1. Add Validation to Common Field Extraction:

Python

```

@classmethod
def _extract_common(cls, log: EventData) -> Dict[str, Any]:
    """Extract common fields with validation."""
    try:
        required_fields = [
            "event", "logIndex", "transactionIndex",
            "transactionHash", "address", "blockHash", "blockNumber"
        ]

        for field in required_fields:
            if field not in log:
                raise ValueError(f"Missing required field: {field}")

    return {
        "event": log["event"],
        "log_index": log["logIndex"],

```

```
    "transaction_index": log["transactionIndex"],
    "transaction_hash": log["transactionHash"],
    "address": log["address"],
    "block_hash": log["blockHash"],
    "block_number": log["blockNumber"],
}
except Exception as e:
    logger.error(f"Failed to extract common fields: {e}, log: {log}")
    raise ValueError(f"Invalid event log structure: {e}")
```

2. Add Graceful Error Handling in Event Processing
3. Add Event Validation Tests

Lido's response: Acknowledged. Failing on invalid network responses is part of a proper oracle flow. When an invalid payload occurs, the oracle should crash with an error instead of suppressing or masking it. Oracle monitoring is configured to trigger alerts on every crash, ensuring that we are immediately aware of any issues.

L-03 – Kubo RPC API Bound to 0.0.0.0

Severity: Low	Impact: Low	Likelihood: Low
Files: kubo/010-set-conf.sh	Status Acknowledged	Violated Property: N/A

Description:

The Kubo API is bound within the container, and has no authentication. This API allows full control over the IPFS node.

kubo/010-set-conf.sh:11

Python

```
ipfs config Addresses.API /ip4/0.0.0.0/tcp/$API_PORT
```

Impact is limited since this isn't internet exposed, but it is reachable within the same docker network, and could get published by Oracle operators unaware of the risk.

The API allows malicious actors to fully control the IPFS node. As configured now, it can be targeted in a lateral movement (attackers that reached the docker network can target it), and could easily be misconfigured to face the internet.

Recommendations

1. Prefer using 127.0.0.1 instead of 0.0.0.0 to limit exposure if possible

Lido's response: Acknowledged. Using Kubo is an experimental feature and is not intended to be enabled on the mainnet.



L-04 – Prometheus Server Bound to 0.0.0.0

Severity: Low	Impact: Low	Likelihood: Low
Files: src/main.py, Dockerfile	Status: Acknowledged	Violated Property: N/A

Description:

src/main.py:56

```
Python
start_http_server(variables.PROMETHEUS_PORT)
```

Dockerfile:67

```
Python
ENV PROMETHEUS_PORT=9000
..
EXPOSE $PROMETHEUS_PORT
```

The Prometheus HTTP server is bound to all interfaces within the container, and the port is exposed.

This allows for various information to be disclosed to attackers, and is available as a potential DDOS target.

Impact is limited since this isn't internet exposed, but it is reachable within the same docker network, and could get published by Oracle operators unaware of the risk.

The Prometheus HTTP server allows for information disclosure. As configured now, it can be targeted in a lateral movement (attackers that reached the docker network can target it), and could easily be misconfigured to face the internet (which would make this critical).



Recommendations

1. Prefer using 127.0.0.1 instead of 0.0.0.0 to limit exposure if possible

Lido's response: Acknowledged. Using Prometheus is an experimental feature and will not be enabled on the mainnet.

L-05 – Missing IPFS Data Validation

Severity: Low	Impact: Low	Likelihood: Low
Files: src/modules/accounting/types.py, src/services/staking_vaults.py	Status: Acknowledged	Violated Property: N/A

Description:

Oracle lacks proper validation of IPFS data before processing, making it vulnerable to malicious IPFS nodes that can return corrupted or malformed data. This vulnerability can cause the Oracle to crash with JSON parsing errors, IndexError due to empty merkle tree or other exceptions, leading to denial-of-service attacks and potential disruption of the entire Lido staking protocol.

src/modules/accounting/types.py:324

Python

```
def from_ipfs_data(cls, raw_bytes: bytes) -> "StakingVaultIpfsReport":  
    data = json.loads(raw_bytes.decode("utf-8"))  # ← No validation  
    return cls(  
        format=data["format"],  
        leaf_encoding=data["leafEncoding"],  
        tree=data["tree"],  # ← Assumes tree exists and is valid  
        values=data["values"],  
        extra_values=data["extraValues"],  
    )±
```

src/services/staking_vaults.py:388

Python

```
def is_tree_root_valid(self, expected_tree_root: str, merkle_tree: StakingVaultIpfsReport) ->  
    bool:  
        # ... build tree_data ...
```

```
rebuild_merkle_tree = self.get_merkle_tree(tree_data)
root_hex = f'0x{rebuild_merkle_tree.root.hex()}'  
return merkle_tree.tree[0] == root_hex and root_hex == expected_tree_root # ← IndexError
if tree is empty
```

Exploit Scenario

- Malicious IPFS node returns valid JSON but with empty tree data:

Python

```
{  
    "format": "standard-v1",  
    "leafEncoding": ["address", "uint256", "uint256", "uint256", "int256"],  
    "tree": [], // ← Empty array  
    "values": [], // ← Empty array  
    "extraValues": {}  
}
```

IndexError when accessing `merkle_tree.tree[0]`, causing Oracle crash

- Malicious IPFS node returns invalid JSON, causing `json.loads` to raise `json.decoder.JSONDecodeError` exception
- Malicious IPFS node returns non UTF-8 characters causing `raw_bytes.decode('utf-8')` to raise `UnicodeDecodeError`.

Recommendations

In `parse_merkle_tree_data`:

- Require IPFS data size limits

Python

```
if len(raw_bytes) > MAX_IPFS_DATA_SIZE:  
    logger.warning(f"IPFS data too large: {len(raw_bytes)} bytes")  
    return None
```

- Handle UTF-8 decoding errors



Python

```
try:  
    decoded_str = raw_bytes.decode("utf-8")  
except UnicodeDecodeError as e:  
    logger.warning(f"Invalid UTF-8 encoding: {e}")  
    return None
```

- Handle JSON parsing errors

Python

```
try:  
    data = json.loads(decoded_str)  
except json.JSONDecodeError as e:  
    logger.warning(f"Invalid JSON data: {e}")  
    return None  
except Exception as e:  
    logger.warning(f"Unexpected error parsing JSON: {e}")  
    return None
```

- Validate required JSON fields

Python

```
required_fields = ["format", "leafEncoding", "tree", "values", "extraValues"]  
for field in required_fields:  
    if field not in data:  
        logger.warning(f"Missing required field: {field}")  
        return None
```

- Validate Tree Structure

Python

```
if not isinstance(data["tree"], list) or len(data["tree"]) == 0:  
    logger.warning("Invalid tree structure")  
    return None
```



Lido's response: Acknowledged. CID Validation reduces this risk as the oracle processes only validated IPFS report data. While CID Validation is not mandatory, disabling it should be used only for testing.

L-06 – Double Counting of end-of-frame block When Calculating Liquidity Fees (Partial Fix)

Severity: Low	Impact: Low	Likelihood: Low
Files: src/modules/accounting/types.py, src/services/staking_vaults.py	Status: Fixed	Violated Property: N/A

Description:

This is a continuation issue of "H-02: Double Counting of end-of-frame block When Calculating Liquidity Fees", covering only the main code path. Double Counting is still feasible when the ipfs report is unavailable. Fix should be applied to this case as well.

staking_vaults.py:_get_start_point_for_fee_calculations():511

Python

```
# Prevent double-counting of vault events:  
# If any vault-related event occurred in the same block as the previous IPFS  
report,  
# it has already been included in that report. To avoid overlapping calculations,  
# we shift the starting point by one block forward.  
return prev_ipfs_report, ref_block.block_number + 1  
  
## When we do NOT HAVE prev IPFS report => we have to check two branches: for mainnet  
and devnet (genesis vaults support)  
## Mainnet  
## in case when we don't have prev ipfs report - we DO have previous oracle report
```

```
## it means we have to take this point for getting fees at the FIRST time only
last_processing_ref_slot =
accounting_oracle.get_last_processing_ref_slot(blockstamp.block_hash)
if last_processing_ref_slot:
    ref_block = get_blockstamp(
        self.w3.cc, last_processing_ref_slot,
SlotNumber(int(last_processing_ref_slot) + slots_per_frame)
    )
    return None, ref_block.block_number      # <-- Double Counting Still Possible
```

Recommendations

Use `ref_block.block_number + 1` also when the ipfs report is unavailable.

Lido's response: Fixed in [ecb54a9](#)

Fix Review: Fixed.

L-07 – Storacha Upload URL Leak in Oracle Logs

Severity: Low	Impact: Low	Likelihood: Low
lido-oracle/src/providers/ipfs/storacha.py	Status: Fixed	Violated Property: N/A

Description:

Storacha file upload URL used for the PUT operation is included in the exception logging and contain sensitive token. According to the example in the storacha API Bridge documentation, this is an ephemeral url which is used for submitting the file and no other credential is being used:
<https://docs.storacha.network/how-to/http-bridge/>

lido-oracle/src/providers/ipfs/storacha.py:92

Python

```
if store_result['status'] == 'upload':
    upload_url = store_result['url']
    upload_headers = store_result['headers']

    try:
        upload_response = requests.put(upload_url, headers=upload_headers,
data=car_file.car_bytes, timeout=self.timeout)
        upload_response.raise_for_status()
    except requests.RequestException as ex:
        logger.error({"msg": "Upload request failed", "error": str(ex)})    #
upload_url is logged
        raise UploadError(f"Upload request failed: {ex}") from ex
```

Recommendations

Sanitize the error message, filter out the sensitive upload URL.



Lido's response: Fixed in [0095fbe](#)

Fix Review: Fixed.

Informational Severity Issues

I-01 - oz_merkle_tree Proof Verification on Internal Node (Informational)

Description: The `StandardMerkleTree.verify()` function in the `oz_merkle_tree` library accepts any 32-byte value as a "leaf" without validating that it represents actual leaf data. This allows internal tree nodes to be used as "leaves" in merkle proofs, potentially enabling data forgery attacks. The Lido oracle's security is not affected as it only uses this library for tree generation, not proof verification. However, if the `verify()` function is used in future oracle implementations, developers must be aware of this vulnerability.

Lido's response: Acknowledged.

I-02 - Division by Zero in get_steth_by_shares Function

Description:

The `get_steth_by_shares` function lacks validation for zero `total_shares`, which can lead to a `ZeroDivisionError` when `total_shares` is 0.

src/utils/apr.py:31-33

Python

```
def get_steth_by_shares(shares: int, total_ether: int, total_shares: int) -> Decimal:  
    return (Decimal(shares) * Decimal(total_ether)) / Decimal(total_shares) # ← NO  
VALIDATION
```

Lido's response: Acknowledged. Total shares can't go below 10 because of the original protocol design.

I-03 - Consensus-critical contract methods should use finalized block by default

Description:

The Oracle system uses finalized blocks for all consensus-critical operations to ensure consistency and prevent race conditions between different Oracle instances. However, the contract interface methods default to 'latest' instead of 'finalized', creating an inconsistency that could lead to consensus failures if future code changes accidentally use the default parameter. While the current codebase doesn't issue any calls that use the default parameter, this design flaw makes the system prone to errors during future development.

src/providers/execution/contracts/lazy_oracle.py:35

Python

```
def get_latest_report_data(self, block_identifier: BlockIdentifier = 'latest') ->
OnChainIpfsVaultReportData:
```

Recommendation:

Change the default parameter from 'latest' to 'finalized' for consensus-critical methods:

Python

```
def get_latest_report_data(self, block_identifier: BlockIdentifier = 'finalized') ->
OnChainIpfsVaultReportData:
```

Lido's response: Acknowledged. Will be fixed in future versions



I-04 - Incorrect Events Order Leads to Wrong Fees Calculation

Description:

in staking_vaults.py::get_vaults_fees() (#533) events dict is generated by type:

Python

```
for fees_updated_event in fees_updated_events:  
    events[fees_updated_event.vault].append(fees_updated_event)  
  
    for minted_event in minted_events:  
        events[minted_event.vault].append(minted_event)  
  
    for burned_event in burn_events:  
        events[burned_event.vault].append(burned_event)  
...  
...
```

staking_vaults.py::calc_liquidity_fee():449

Python

```
events[vault_address].sort(key=lambda x: x.block_number, reverse=True)
```

- get_vaults_fees() generates the events dict by type.
- calc_liquidity_fee() sorts the event lists by block numbers only.
- If two different event types take place at the same block, and affect fees – the fees calculated within that block can be wrong (yet deterministic for all Oracles).
- For example, if a block contains VaultFeesUpdated events as well as MintedSharesOnVault events, the fees calculated by the Oracle within that block may be incorrect (fees should be calculated by onchain chronological order)

Impact: If a block contains multiple fees related events, this issue leads to incorrect fees calculation. Vault owners and/or stakers may abuse this to minimize fees within blocks.

Recommendation:



Use a stable sort based on block number, tx index and log index.

Lido's response: Fixed in [7582f00](#)

Fix Review: Fixed. Events are now sorted by log_index.



I-05 - Oracle HTTP Provider follows HTTP Redirects (30x)

Description:

The oracle uses the `HTTPProvider` class for HTTP related operation on multiple components – fetching Consensus Layer Data, Validator Keys, IPFS operations, monitoring and alerts.

`HTTPProvider` uses `requests.Session()` which, by default, sets `allow_redirects=True`.

If an attacker manages to compromise a domain or an endpoint used by the oracle and make it redirect to a domain it controls, they would be able to provide malicious responses.

Recommendation:

1. By default, allow following redirects only to the same host.
2. If a specific component or service requires HTTP redirects then enable it specifically for that service.

Lido's response: Acknowledged. Will be fixed in future versions.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.