LIDO ORACLE SECURITY AUDIT REPORT

April 07, 2025

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.

Stage goal

Eliminate typical vulnerabilities.

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The Lido Oracle is a daemon for the Lido decentralized staking service, consisting of three primary modules:

1. Accounting Module

   - Updates protocol TVL and distributes node-operator rewards
   - Manages exited/stuck validator information and user withdrawal requests
   - Controls bunker mode activation
   - Operates in 24-hour frames with stages: waiting, data collection, hash consensus, core update report, and extra data report

2. Ejector Module

   - Manages Lido validator exits based on withdrawal needs
   - Predicts ETH income and schedules validator withdrawals
   - Can force eject validators from Node Operators with boosted exits

3. CSM (Community Staking Module) Module

   - Collects CSM node operator performance data
   - Calculates validator performance based on attestations
   - Creates a Merkle tree of operator share distribution
   - Submits the Merkle root to the module contract

These modules maintain the Lido staking service's integrity, ensuring accurate reward distribution, validator management, and performance tracking.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Lido
Project name	Lido Oracle
Timeline	03.02.2025 - 17.03.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
03.02.2025	0dbc1d8a2acf069075118e94770c20de2c1de3d7	Commit for the audit
13.02.2025	1e01a750e39ec50bef304b73947693e7b9e7ee9b	Commit with updates
21.02.2025	0a7f910cffae8437e02a480aaa270a1aa1a1990a	Commit for the re-audit
03.03.2025	79f714cea60ef725bc0662db2138ddafe9f4c11c	Commit with updates
17.03.2025	5380490c3511f20570ce2424454a2eea21226145	Commit with updates
03.04.2025	41f3f9671ea7e349e048c3ac47264a118c7983a8	Commit with updates

Project Scope

The audit covered the following files:

File name	Link

File name	Link
src/__init__.py	__init__.py
src/constants.py	constants.py
src/main.py	main.py
src/metrics/__init__.py	__init__.py
src/metrics/healthcheck_server.py	healthcheck_server.py
src/metrics/logging.py	logging.py
src/metrics/prometheus/__init__.py	__init__.py
src/metrics/prometheus/accounting.py	accounting.py
src/metrics/prometheus/basic.py	basic.py
src/metrics/prometheus/business.py	business.py
src/metrics/prometheus/csm.py	csm.py
src/metrics/prometheus/duration_meter.py	duration_meter.py
src/metrics/prometheus/ejector.py	ejector.py
src/metrics/prometheus/validators.py	validators.py
src/modules/__init__.py	__init__.py
src/modules/accounting/__init__.py	__init__.py
src/modules/accounting/accounting.py	accounting.py
src/modules/accounting/third_phase/__init__.py	__init__.py
src/modules/accounting/third_phase/extra_data.py	extra_data.py
src/modules/accounting/third_phase/types.py	types.py

File name	Link
src/modules/accounting/types.py	types.py
src/modules/checks/__init__.py	__init__.py
src/modules/checks/checks_module.py	checks_module.py
src/modules/checks/pytest.ini	pytest.ini
src/modules/checks/suites/__init__.py	__init__.py
src/modules/checks/suites/common.py	common.py
src/modules/checks/suites/conftest.py	conftest.py
src/modules/checks/suites/consensus_node.py	consensus_node.py
src/modules/checks/suites/execution_node.py	execution_node.py
src/modules/checks/suites/ipfs.py	ipfs.py
src/modules/checks/suites/keys_api.py	keys_api.py
src/modules/csm/__init__.py	__init__.py
src/modules/csm/checkpoint.py	checkpoint.py
src/modules/csm/csm.py	csm.py
src/modules/csm/log.py	log.py
src/modules/csm/state.py	state.py
src/modules/csm/tree.py	tree.py
src/modules/csm/types.py	types.py
src/modules/ejector/__init__.py	__init__.py
src/modules/ejector/data_encode.py	data_encode.py

File name	Link
src/modules/ejector/ejector.py	ejector.py
src/modules/ejector/sweep.py	sweep.py
src/modules/ejector/types.py	types.py
src/modules/submodules/__init__.py	__init__.py
src/modules/submodules/consensus.py	consensus.py
src/modules/submodules/exceptions.py	exceptions.py
src/modules/submodules/oracle_module.py	oracle_module.py
src/modules/submodules/types.py	types.py
src/providers/__init__.py	__init__.py
src/providers/consensus/__init__.py	__init__.py
src/providers/consensus/client.py	client.py
src/providers/consensus/types.py	types.py
src/providers/consistency.py	consistency.py
src/providers/execution/__init__.py	__init__.py
src/providers/execution/base_interface.py	base_interface.py
src/providers/execution/contracts/accounting_oracle.py	accounting_oracle.py
src/providers/execution/contracts/base_oracle.py	base_oracle.py
src/providers/execution/contracts/burner.py	burner.py
src/providers/execution/contracts/cs_accounting.py	cs_accounting.py

File name	Link
src/providers/execution/contracts/cs_fee_distributor.py	cs_fee_distributor.py
src/providers/execution/contracts/cs_fee_oracle.py	cs_fee_oracle.py
src/providers/execution/contracts/cs_module.py	cs_module.py
src/providers/execution/contracts/deposit_contract.py	deposit_contract.py
src/providers/execution/contracts/exit_bus_oracle.py	exit_bus_oracle.py
src/providers/execution/contracts/hash_consensus.py	hash_consensus.py
src/providers/execution/contracts/lido.py	lido.py
src/providers/execution/contracts/lido_locator.py	lido_locator.py
src/providers/execution/contracts/oracle_daemon_config.py	oracle_daemon_config.py
src/providers/execution/contracts/oracle_report_sanity_checker.py	oracle_report_sanity_checker.py
src/providers/execution/contracts/staking_router.py	staking_router.py
src/providers/execution/contracts/withdrawal_queue_nft.py	withdrawal_queue_nft.py
src/providers/execution/exceptions.py	exceptions.py
src/providers/http_provider.py	http_provider.py
src/providers/ipfs/__init__.py	__init__.py
src/providers/ipfs/cid.py	cid.py
src/providers/ipfs/dummy.py	dummy.py

File name	Link
src/providers/ipfs/gw3.py	gw3.py
src/providers/ipfs/multi.py	multi.py
src/providers/ipfs/pinata.py	pinata.py
src/providers/ipfs/public.py	public.py
src/providers/ipfs/types.py	types.py
src/providers/keys/__init__.py	__init__.py
src/providers/keys/client.py	client.py
src/providers/keys/types.py	types.py
src/services/__init__.py	__init__.py
src/services/bunker.py	bunker.py
src/services/bunker_cases/__init__.py	__init__.py
src/services/bunker_cases/abnormal_cl_rebase.py	abnormal_cl_rebase.py
src/services/bunker_cases/midterm_slashing_penalty.py	midterm_slashing_penalty.py
src/services/bunker_cases/types.py	types.py
src/services/exit_order_iterator.py	exit_order_iterator.py
src/services/prediction.py	prediction.py
src/services/safe_border.py	safe_border.py
src/services/validator_state.py	validator_state.py
src/services/withdrawal.py	withdrawal.py
src/types.py	types.py

File name	Link
src/utils/__init__.py	__init__.py
src/utils/abi.py	abi.py
src/utils/blockstamp.py	blockstamp.py
src/utils/build.py	build.py
src/utils/cache.py	cache.py
src/utils/dataclass.py	dataclass.py
src/utils/env.py	env.py
src/utils/events.py	events.py
src/utils/exception.py	exception.py
src/utils/input.py	input.py
src/utils/range.py	range.py
src/utils/slot.py	slot.py
src/utils/timeit.py	timeit.py
src/utils/types.py	types.py
src/utils/units.py	units.py
src/utils/validator_state.py	validator_state.py
src/utils/web3converter.py	web3converter.py
src/variables.py	variables.py
src/web3py/__init__.py	__init__.py
src/web3py/contract_tweak.py	contract_tweak.py

File name	Link
src/web3py/extensions/__init__.py	__init__.py
src/web3py/extensions/consensus.py	consensus.py
src/web3py/extensions/contracts.py	contracts.py
src/web3py/extensions/csm.py	csm.py
src/web3py/extensions/fallback.py	fallback.py
src/web3py/extensions/keys_api.py	keys_api.py
src/web3py/extensions/lido_validators.py	lido_validators.py
src/web3py/extensions/tx_utils.py	tx_utils.py
src/web3py/middleware.py	middleware.py
src/web3py/types.py	types.py

Docker Image Hash Validation

After conducting the audit, the team that reviewed the scope verified the published image by building the Docker container locally, following the instructions provided by the Lido team. It was confirmed that the local and published manifest digests match and are equal to `sha256:b15bb7fcd19b6368cf8f8498b52bb527b1da1f36227a44b47b3a5709db8b4e46` (Image IDs are equal to `sha256:916d140bd8eb0123460e3b479c94f23437d87deaecb71bf7354e8612e6b5c32b`).

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	1
Low	5

ID	Name	Severity	Status
M-1	Incorrect CL Rebase Calculation	Medium	Fixed
L-1	Missed Validators in `_get_consensus_lido_state` Function	Low	Fixed
L-2	Unaccounted Assumption on `MAX_PENDING_PARTIALS_PER_WITHDRAWALS_SWEEP` in `predict_withdrawals_number_in_sweep_cycle`	Low	Fixed
L-3	Potentially Incorrect Accounting for the Validators' Effective Balance	Low	Fixed
L-4	Incorrect Variable Name in the Comment	Low	Fixed
L-5	Missing Accounting for Pending Validator Deposits with Zero Effective Balance	Low	Acknowledged

1.6 Conclusion

During the audit, in addition to examining well-known attack vectors, we carefully investigated the following areas:

- **Hard Fork Epoch Activation** - The specifics of the new hard fork will only be used when the corresponding epoch takes place and when the consensus version is switched. This ensures that protocol-level changes are applied precisely at the intended network transition point. All functions smoothly accommodate the upgrade, which requires upgrading the consensus version in the contracts first.
- **Special logic for unprocessed deposits** – The logic triggered by the Pectra upgrade to handle **non-processed deposits** was thoroughly reviewed to ensure correct implementation.
- **Source code differences** – A **detailed diff analysis** was conducted to compare the latest audited version of the service's source code with the current version.
- **Electra upgrade impact** – All **specification changes introduced by the Electra upgrade** to the consensus layer were researched and verified to ensure proper handling in the new service version.
- **EIP-7549 Committee Index Calculation** - The implementation requires attestations to have a zero index when using committee bits, allowing multiple committee indices to be extracted from a single attestation. Changes made to the `process_attestations` function were carefully examined.
- **Bunker Mode Effective Balance Handling** - The implementation in `abnormal_cl_rebase.py` and `midterm_slashing_penalty.py` meticulously tracks validator balances across epochs, accounting for newly appeared validators, withdrawal vault balance changes, and potential slashing penalties.
- **Ejector Module Exit Epoch Prediction** - Exit epoch is predicted correctly in the ejector module, enabling precise validator exit scheduling and maintaining the protocol's validator lifecycle management.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Incorrect CL Rebase Calculation
Severity	Medium
Status	Fixed in 1e01a750

Description

There is an issue in the abnormal_cl_rebase.py#L206 method of `abnormal_cl_rebase.py`. There is an incorrect calculation of the Consensus Layer (CL) rebase. The current implementation incorrectly adds the `validators_count_diff_in_gwei` to the raw CL rebase, which leads to an inaccurate representation of validator effective balances.

When new validators appear between epochs, they start with a full 32 ETH balance. These new validator balances should be subtracted from the `raw_cl_rebase` to prevent overestimating the CL rewards.

The issue is classified as **Medium** severity because that miscalculation may lead to an incorrect CL rebase calculation, which then may affect the Bunker Mode switching.

Recommendation

We recommend changing the mentioned `cl_rebase` calculation formula to `cl_rebase = Gwei(raw_cl_rebase - validators_count_diff_in_gwei + withdrawn_from_vault)`.

Client's Commentary

> Fixed in commit 1e01a750e39ec50bef304b73947693e7b9e7ee9b

2.4 Low

L-1	Missed Validators in `_get_consensus_lido_state` Function
Severity	Low
Status	Fixed in 0a7f910c

Description

This issue has been identified in the accounting.py#L211 of the `accounting` module.
The implementation does not account for validators that have been deposited but are not immediately reflected in the consensus registry. This means their balance is temporarily lost in calculations, which can lead to inaccurate state reporting. Although the Lido contract compensates for this using a `deposited_vals - CL_vals` diff approach, it relies on an assumption that validator balance can be only 32 ETH, which may become invalid in future protocol upgrades. The same issue is present in the exit_order_iterator.py#L152 of the `exit_order_iterator.py` as the constant `LIDO_DEPOSIT_AMOUNT` is used to calculate the transient validators' balance.
The issue is classified as **Low** severity because it does not immediately break functionality but can lead to inaccurate reporting and inconsistencies in future updates.

Recommendation

We recommend adding a comment to the method to ensure that this behavior will be accounted for in future updates.

Client's Commentary

> A description was added for the LIDO_DEPOSIT_AMOUNT variable, which warns about the need to revise the working logic if the deposit algorithm changes.
> PR-630

L-2	Unaccounted Assumption on `MAX_PENDING_PARTIALS_PER_WITHDRAWALS_SWEEP` in `predict_withdrawals_number_in_sweep_cycle`
Severity	Low
Status	Fixed in 0a7f910c

Description

This issue has been identified in the sweep.py#L43 of the `sweep` submodule.
The function assumes that `MAX_PENDING_PARTIALS_PER_WITHDRAWALS_SWEEP` will never be reached, which might not always hold true.
The issue is classified as **Low** severity because it does not cause an immediate failure but may lead to miscalculations that could affect staking performance analysis.

Recommendation

We recommend explicitly documenting this assumption in the function comments to ensure awareness among developers.

Client's Commentary

> Commented in commit c6a4a3b7

L-3	Potentially Incorrect Accounting for the Validators' Effective Balance
Severity	Low
Status	Fixed in 0a7f910c

Description

There is an issue in the abnormal_cl_rebase.py#L306 in `abnormal_cl_rebase.py`. The `MIN_ACTIVATION_BALANCE`, equal to 32 ETH, is used to calculate the effective balance of new validators instead of their actual balances (assuming that there is a possibility of processing larger deposits in the future).

The issue is classified as **Low** severity because for now it is impossible to have any validator with a deposit larger than 32 ETH.

Recommendation

We recommend adding a comment to the method to ensure that this behavior is accounted for in future updates.

Client's Commentary

> Variable name updated in commit
> PR-628
>
> Additionally, a description was added for the LIDO_DEPOSIT_AMOUNT variable, which warns about the need to revise the working logic if the deposit algorithm changes.
> PR-630

L-4	Incorrect Variable Name in the Comment
Severity	Low
Status	Fixed in 0a7f910c

Description

There is a validator_state.py#L132 of the `validator_state.py`. It accepts the `ref_epoch` parameter, but the comment to this method describes the `epoch` variable.

Recommendation

We recommend changing the variable name used in the comment to `ref_epoch`.

Client's Commentary

> Fixed in commit
> PR-629

L-5	Missing Accounting for Pending Validator Deposits with Zero Effective Balance
Severity	Low
Status	Acknowledged

Description

There is an issue in the exit_order_iterator.py#L178 of the `exit_order_iterator.py`. This function calculates the total validators' effective balance, but doesn't account for the validators, which are not yet processed after the Pectra upgrade and have effective balance equal to 0. Those validators need to be processed separately using the `pending_deposits` list.

The issue is classified as **Low** severity because there may be a slight underestimation of the validators' effective balance, which may affect the validators' exit order when there are still some of the not yet processed after the transition (to the Pectra) period.

Recommendation

We recommend calculating the mentioned validators' effective balance using the `calculate_total_eth1_bridge_deposits_amount` function in the same way as it is done in other parts of the code.

Client's Commentary

> Acknowledged. Deposits made shortly before the Pectra hard fork but processed after its activation can indeed introduce some inaccuracies in calculating validator exit priorities.
>
> However, given the current share of Lido node operators in the network, the activation of this predicate seems practically unattainable in the foreseeable future. Considering the even stake distribution among operators and validators' network lifetime, the impact of this predicate on exit priorities will be minimal or nonexistent.
>
> If any inaccuracy does occur, it will be corrected by the nearest reports after such transient deposits pass the `pending_deposits` queue or even earlier by new deposits into the protocol. Since the likelihood of such an event is extremely low, limited to a narrow window, and the impact is negligible, the dev team believes that introducing a fix would add unnecessary complexity to the code and can therefore be omitted.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes