MixBytes()

Lido v3 Security Audit Report

APRIL 21, 2026

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

VaultHub is the core registry managing staking vaults in Lido V3. It tracks each vault's liability shares, enforces minting limits, handles oracle reports, and controls validator withdrawals. LazyOracle is the oracle contract that sanity-checks vault report data before forwarding it to VaultHub for processing.

The codebase was audited in 1 day by 4 auditors using a combination of manual review and automated tooling.

The diff contains two targeted fixes. First, LazyOracle's `_handleSanityChecks` drops the `_maxLiabilityShares > record.maxLiabilityShares` bound check. This condition caused false reverts because `record.maxLiabilityShares` can drift between the oracle's reference slot and report delivery — it increases on stETH mints and decreases on burns/rebalances. The check was redundant: VaultHub's `_applyVaultReport` already guards against oracle manipulation by only updating `maxLiabilityShares` when the on-chain value exactly matches the reported one, skipping the update otherwise. The remaining `_maxLiabilityShares >= _liabilityShares` invariant is preserved as defense-in-depth against malformed reports.

Second, `VaultHub.triggerValidatorWithdrawals` now outright forbids any partial withdrawal when the vault has an obligations shortfall (`obligationsShortfallAmount > 0`). Previously, partial withdrawals were allowed if the requested amount appeared sufficient to cover the shortfall, but this check was flawed: with 0x02 compounding withdrawal credentials, the consensus layer caps partial withdrawals to the validator's actual excess above 32 ETH. A vault owner could front-run a forced exit by requesting a partial withdrawal on a validator with minimal excess (e.g. 32 ETH + 1 gwei), which VaultHub would accept but CL would trim to 1 gwei — not covering the shortfall and setting `pending_balance_to_withdraw != 0`, blocking subsequent forced full exits until finalization. The fix replaces the minimum-amount tracking loop with a simple boolean check for any non-zero amount, breaking early on the first partial withdrawal found.

Third (commit `6e36d1969788097b857ce8d15f28beccc3f8cfdc`), the same `triggerValidatorWithdrawals` function now additionally reverts with `PartialValidatorWithdrawalNotAllowed()` when `_operatorGrid().isVaultInJail(_vault)` returns true. This closes a fee-avoidance griefing vector. The attack works as follows: a vault has accrued unsettled Lido fees (tracked in `VaultRecord.cumulativeLidoFees / settledLidoFees`) which contribute to its obligations shortfall via `_obligationsAmount`. This shortfall

normally blocks partial withdrawals per the check described above. However, the attacker can temporarily fund the vault via VaultHub.fund() to inject ETH into the vault's available balance, zeroing out the shortfall. They then mint stETH shares via mintShares() (increasing liability), trigger partial validator withdrawals (now passing the shortfall check since the available balance covers obligations), and finally call rebalance() to burn the freshly minted shares and withdraw the temporarily deposited ETH back. The key asymmetry is that rebalance() calls the internal _withdraw, which only checks _totalValue and does not enforce fee settlement — unlike the public withdraw() which limits withdrawals to _withdrawableValue, a function that deducts unsettled fees from the available amount. After rebalancing, liability returns to its original level, the temporarily injected ETH is recovered by the attacker, yet the full fee obligation remains unsettled and the vault's available balance returns to near zero. This cycle can be repeated each oracle frame, allowing the operator to trigger partial withdrawals while perpetually avoiding fee payment.

The fix relies on governance detecting this behavior and jailing the offending vault via OperatorGrid.setVaultJailStatus() (requires REGISTRY_ROLE). Once jailed, the new check blocks all partial withdrawals before the shortfall calculation is even reached. The jail status is stored in OperatorGrid's isVaultInJail mapping, is persistent across tier changes and disconnect/reconnect cycles, and can only be cleared by governance. Full validator withdrawals remain permitted for jailed vaults, preserving the ability to fully exit validators and rebalance. The forceValidatorExit path is also unaffected, as it bypasses triggerValidatorWithdrawals entirely.

Note on residual risk: the fix is reactive rather than preventive, so the fund-mint-partialWithdraw-rebalance cycle can execute at least once before governance jails the vault; governance should respond within one oracle frame to prevent repeated exploitation.

The audit verified that: removing the maxLiabilityShares bound check does not open oracle manipulation since report data is Merkle-proven and VaultHub's equality guard prevents unauthorized updates; the partial withdrawal ban cannot be bypassed through batched or mixed withdrawal requests since the loop catches any non-zero amount; healthy vaults retain unrestricted partial withdrawal capability; unhealthy vaults can still perform full withdrawals (all amounts == 0); forceValidatorExit remains unaffected as it bypasses triggerValidatorWithdrawals entirely; and _requireFreshReport is still enforced, preventing stale-report-based attacks; and the jail check queries OperatorGrid.isVaultInJail, a simple storage read that cannot be manipulated by the vault operator since only REGISTRY_ROLE holders can modify jail status.

Contracts outside the two audited files — including the broader VaultHub logic, stVault implementation, and the rest of LazyOracle — were not in scope of this audit.

The changes are minimal, well-motivated, and correct. No issues were found.

1.3 Project Overview

Summary

Title	Description
Client	Lido
Category	Liquid Staking
Project	Lido v3
Type	Solidity
Platform	EVM
Timeline	19.03.2026 - 31.03.2026

Scope of Audit

File	Link
contracts/0.8.25/vaults/LazyOracle.sol	contracts/0.8.25/vaults/LazyOracle.sol
contracts/0.8.25/vaults/VaultHub.sol	contracts/0.8.25/vaults/VaultHub.sol

Versions Log

Date	Commit Hash	Note
19.03.2026	ffd6c57fc1ae2d4193dd6e03757e1f95cd9b616d	Initial Commit
19.03.2026	b0be4a06d238376842d624c73b4650caba1d4979	Commit with Updates
25.03.2026	6e36d1969788097b857ce8d15f28beccc3f8cfdc	Commit with Updates
31.03.2026	702ef9da1bc067a996614de1e3c6a53b797fa4c2	Commit with Updates

Mainnet Deployments

We verified that the bytecode of the LazyOracle and VaultHub implementation contracts audited in this report matches the contracts bytecode from the commit referenced in the audit scope.
The initialization parameters of these new implementations are consistent with those currently used by the production mainnet proxies (LazyOracle proxy at 0x5DB427080200c235F2Ae8Cd17A7be87921f7AD6c and VaultHub proxy at 0x1d201BE093d847f6446530Efb0E8Fb426d176709), whose current implementations are 0x47f3a6b1E70F7Ec7dBC3CB510B1fdB948C863a5B and 0x7c7d957D0752AB732E73400624C4a1eb1cb6CF50 respectively.
The upgrade is executed via Aragon DAO vote (https://dao.lido.fi/vote/199). The vote bundles 25 forward calls; two of them correspond to this audit's scope:
Item 4: LazyOracle(0x5DB427080200c235F2Ae8Cd17A7be87921f7AD6c).proxy__upgradeTo(0x96c9a897D116ef6600 - new implementation matches commit cbcc6d159a6e0f5099d1606096bdc8150ffe7654 in deployed-mainnet.json and was compiled from audited commit 702ef9da1bc067a996614de1e3c6a53b797fa4c2.
Item 5: VaultHub(0x1d201BE093d847f6446530Efb0E8Fb426d176709).proxy__upgradeTo(0x6330fE7756FBE8649adf - new implementation matches commit cbcc6d159a6e0f5099d1606096bdc8150ffe7654 in deployed-mainnet.json; its constructor arguments (Locator, stETH, AccountingOracle, relative share limit bps 1000) match the currently active VaultHub implementation 0x7c7d957D0752AB732E73400624C4a1eb1cb6CF50.
Both upgrades use proxy__upgradeTo (no initializer call), which is safe given the diffs are purely logic changes with no storage-layout additions or reordering.
The upgrade is also not exposed to the CPIMP attack class, in which an attacker frontruns the initialize() call on a freshly deployed proxy to seize admin/owner roles. In this case: (i) both proxies (LazyOracle at 0x5DB427080200c235F2Ae8Cd17A7be87921f7AD6c, VaultHub at 0x1d201BE093d847f6446530Efb0E8Fb426d176709) are already live and were initialized during the v3.0.0 and v3.0.1 deployments — their InitializableStorage._initialized slot is non-zero, so any reinvocation of initialize(...) after the upgrade will revert with InvalidInitialization(); (ii) neither of the new implementations introduces a reinitializer(version) or any alternative post-upgrade init hook, and no unused-role grant path or __init-style function exists that could be invoked post-swap.
An earlier v3.0.2 VaultHub implementation was deployed to mainnet on 2026-03-27 at 0x77c017EBB74037E593F3552596dEb75565F01294, corresponding to commit ead66e8f76720e2ba70908b1de6fc10c8b0524de in the core repository. This build predates the safecast hardening introduced in commit 702ef9da1bc067a996614de1e3c6a53b797fa4c2 and was therefore replaced on 2026-03-31 by a new implementation at 0x6330fE7756FBE8649adfb9A541d61C5edB8B4D70, recorded in deployed-mainnet.json under commit cbcc6d159a6e0f5099d1606096bdc8150ffe7654. Vote #199 correctly targets the superseding implementation at 0x6330fE7756FBE8649adfb9A541d61C5edB8B4D70, and the earlier 0x77c017EBB74037E593F3552596dEb75565F01294 contract is no longer referenced by any proxy.

File	Address
LazyOracle	0x96c9a897D116ef660086d3aA67b3af653324aB37
VaultHub	0x6330fE7756FBE8649adfb9A541d61C5edB8B4D70

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	0

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

Not Found

3. About MixBytes

MixBytes is a leading provider of smart contract auditing and blockchain security research, helping Web3 projects enhance their security and reliability.

Since its inception, MixBytes has been dedicated to safeguarding innovation in **DeFi** through rigorous audits and cutting-edge technical research.

Our team brings together experienced engineers, security auditors, and blockchain researchers with deep expertise in smart contract security and protocol design.

With years of proven experience in Web3, MixBytes combines in-depth technical excellence with a proactive, security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by leading blockchain protocols such as **Lido**, **Aave**, **Curve**, **1inch**, **Fluid**, **Gearbox**,**Resolv**, and others. MixBytes has successfully secured billions of dollars in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems, backed by years of hands-on experience.
- **Innovative Research:** MixBytes actively contributes to blockchain security research and open-source tools, sharing insights with the global community.

Our Services

- **Smart Contract Audits:** Comprehensive security assessments of DeFi protocols to detect and mitigate vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical studies, economic and protocol-level modeling for Web3 projects.
- **Custom Security Solutions:** Tailored frameworks and advisory for complex decentralized systems and blockchain ecosystems.

Contact Information

 https://mixbytes.io/

 https://github.com/mixbytes/audits_public

 https://x.com/MixBytes

 hello@mixbytes.io