MixBytes()

Lido CircuitBreaker Security Audit Report

MAY 01, 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

CircuitBreaker is a permanent emergency pause manager for the Lido protocol, introduced by LIP-34 as the successor to GateSeals. It allows the Lido DAO Agent to delegate single-use pause authority to multisig committees (pausers) that can instantly pause designated pausable contracts without waiting for a governance vote, while requiring those committees to periodically prove their liveness through on-chain heartbeats. Unlike GateSeals, CircuitBreaker is a single permanent contract with a stable address, adjustable global parameters, and an on-chain enumerable registry of pausable-pauser assignments, eliminating the annual redeployment cycle and enabling granular per-contract emergency response.

The audit was carried out over a period of 2.5 days by a team of 3 auditors, combining manual review with automated tooling.

The code base was reviewed against the following primary attack vectors:

1. Unauthorized pause / privilege escalation — verifying that only the admin can register pausers and modify global parameters, that only the registered pauser for a given pausable can trigger its pause, that the admin itself cannot invoke a pause, and that there is no path by which an unregistered or expired pauser can act on a pausable contract.
2. Heartbeat liveness bypass and griefing — verifying that an expired pauser cannot send a heartbeat to self-revive, that re-registration by the admin correctly refreshes the heartbeat, that changing the global `heartbeatInterval` does not retroactively affect already-stored expiries, and that the heartbeat mechanism cannot be griefed by a malicious pauser frontrunning admin operations.
3. Registry state integrity — verifying the consistency of the enumerable `pausables` set and `oneBasedIndex` / `pausableCount` mappings across all combinations of register, replace, and unregister operations, including edge cases such as idempotent unregistration, replace-self, and the swap-and-pop removal path, so that no state corruption, duplicate entries, stale indexes, or counter underflows can occur.
4. One-pauser-per-pausable invariant — verifying that the registry enforces a strict one-to-one mapping from each pausable to its pauser (`Registry.pauser[pausable]`), making it architecturally impossible to assign multiple pausers to the same pausable, while a single pauser can be registered for many pausables.
5. Heartbeat interval change does not retroactively affect existing expiries — verifying that when the admin updates the global `heartbeatInterval`, already-stored `heartbeatExpiry` values for active pausers are not recalculated. Each pauser's expiry remains based on

the interval that was in effect at the time of their last heartbeat or registration, and the new interval only applies on subsequent `heartbeat()` calls or re-registrations. This avoids surprising existing pausers with a suddenly shortened or extended deadline mid-cycle.

Out of scope for this audit: the off-chain monitoring infrastructure for heartbeat expiry and pauser liveness; the multisig committee implementations and their key-management practices; the interaction with Dual Governance, Aragon voting, and the ResealManager beyond the documented integration assumptions; the deployment scripts and the on-chain migration vote from GateSeals.

Overall, the CircuitBreaker codebase is of high quality: it is small, single-file (plus one library), free of external dependencies, non-upgradeable, and implemented with explicit bounds validation, an explicit transient reentrancy guard, and strict adherence to Check-Effects-Interactions in the pause flow. The implementation faithfully reflects the LIP-34 specification and preserves all security properties of the GateSeal predecessor. No critical, high or medium severity issues were identified.

It is worth noting that the `ADMIN` address is immutable – set once at deployment with no rotation mechanism. Only `ADMIN` can register or replace pausers via `registerPauser()`. If the admin is set to the Lido DAO Agent, replacing an inactive or compromised pauser requires a full governance vote, which may take days. This creates an inherent tension: the very governance delay that CircuitBreaker is designed to bypass for emergency pauses also applies to pauser management itself. This is a deliberate design tradeoff – immutability eliminates admin takeover risk at the cost of slower pauser rotation under DAO governance.

1.3 Project Overview

Summary

Title	Description
Client	Lido
Category	Liquid Staking
Project	CircuitBreaker
Type	Solidity
Platform	EVM
Timeline	09.04.2026 - 14.04.2026

Scope of Audit

File	Link
src/CircuitBreaker.sol	src/CircuitBreaker.sol
src/Registry.sol	src/Registry.sol

Versions Log

Date	Commit Hash	Note
09.04.2026	30b01f13792e73b4dfc50e4aa093ab4dbf36802a	Initial Commit
14.04.2026	b4b2fbc921b3191560a3fc62d502d4bb98ad99e1	Commit for Re-audit

Mainnet Deployments

File	Address
CircuitBreaker.sol	0x6019cb557978296ba3c08a7b73225c0975dfb2f7

The deployment compiles byte-for-byte from the audit-scope source at commit b4b2fbc921b3191560a3fc62d502d4bb98ad99e1, with the only deviations being the five constructor-immutable slots, which hold the expected values.
The immutables decode to ADMIN = 0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c (Lido Aragon Agent), MIN_PAUSE_DURATION = 5 days, MAX_PAUSE_DURATION = 60 days, MIN_HEARTBEAT_INTERVAL = 30 days, MAX_HEARTBEAT_INTERVAL = ~3 years, and the mutable initial values pauseDuration = 21 days and heartbeatInterval = 365 days read back consistently both from on-chain getters and from the three configuration events emitted in the creation transaction (CircuitBreakerInitialized, PauseDurationUpdated, HeartbeatIntervalUpdated).
No pauser has been registered yet and no further configuration events have been emitted; the on-chain state is the constructor baseline only. Re-validation will be required after the Lido DAO vote to confirm that the registered pauser-to-pausable bindings are correct - that each pausable is paired with the intended pauser multisig and that the CircuitBreaker has been granted the pause role on each of those pausables.

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	1

Findings Statuses

ID	Finding	Severity	Status
L-1	heartbeatExpiry is not cleared on pauser removal	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	heartbeatExpiry is not cleared on pauser removal		
Severity	Low	**Status**	Fixed in b4b2fbc9

Description

In CircuitBreaker.sol, the heartbeatExpiry mapping stores the timestamp after which a pauser is no longer authorized to heartbeat or pause.

This mapping is written in _updateHeartbeat whenever a pauser is registered, sends a heartbeat, or triggers a pause.

However, heartbeatExpiry[pauser] is never cleared when a pauser loses all its registrations. This happens in two scenarios:

1. The admin unregisters the pauser's last pausable by calling registerPauser(pausable, address(0)). The Registry.setPauser function decrements pausableCount[pauser], potentially down to zero, but CircuitBreaker does not touch heartbeatExpiry[pauser].
2. The pauser triggers a pause on its last assigned pausable via pause(pausable). The Registry.setPauser(_pausable, address(0)) call inside pause() decrements pausableCount[pauser] to zero for a pauser that had a single assignment, but heartbeatExpiry[pauser] retains its previously stored, still-in-the-future value.

As a result, the public view function isPauserLive(pauser) continues to return true for addresses that are no longer registered as pausers anywhere in the registry, until the stored expiry naturally passes.

This is not exploitable as the heartbeat() function explicitly gates on registry.isRegistered(msg.sender), and the pause() function gates on msg.sender == registry.getPauser(_pausable). A pauser whose pausableCount is zero cannot heartbeat, cannot pause, and is effectively retired regardless of the stale heartbeatExpiry value.

The impact is limited to off-chain monitoring: any external observer that uses isPauserLive(pauser) as a standalone liveness check, without first confirming that the pauser still has at least one registered pausable, can be misled into believing that a fully retired address is still an active committee. Given that LIP-34 emphasizes on-chain

enumerability and transparency as a first-class goal, a view function that can silently return a misleading answer represents a deviation from the spirit of that design goal.

Recommendation

Clear `heartbeatExpiry[pauser]` whenever the pauser's `pausableCount` transitions to zero, so that `isPauserLive` reflects the actual registration state.

One approach is to wrap the `registry.setPauser` calls in `CircuitBreaker` and explicitly delete the expiry when appropriate. For example, in the admin path:

```
function registerPauser(address _pausable, address _newPauser) external onlyAdmin {
    address previousPauser = registry.getPauser(_pausable);
    registry.setPauser(_pausable, _newPauser);

    if (previousPauser != address(0) && registry.getPausableCount(previousPauser) == 0) {
        delete heartbeatExpiry[previousPauser];
    }

    if (_newPauser != address(0)) _updateHeartbeat(_newPauser, false);
}
```

and, symmetrically, in the `pause` flow after `registry.setPauser(_pausable, address(0))`:

```
if (registry.getPausableCount(msg.sender) == 0) {
    delete heartbeatExpiry[msg.sender];
}
```

Alternatively, strengthen `isPauserLive` to require current registration as well:

```
function isPauserLive(address _pauser) public view returns (bool) {
    return registry.isRegistered(_pauser) && block.timestamp < heartbeatExpiry[_pauser];
}
```

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