MixBytes()

DeFi Wrapper Mellow Strategy Security Audit Report

MARCH 19, 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

MellowStrategy is a strategy adapter within the Lido Earn architecture that routes user deposits through a Mellow Vault via its deposit and redeem queues, enabling users to earn additional DeFi yield on top of Ethereum staking rewards.

During supply, the strategy accepts ETH (and/or a specified wstETH amount), deposits the ETH into the StvStETHPool to mint stv tokens to the user's StrategyCallForwarder, then mints wstETH against the stv collateral and deposits it into the Mellow deposit queue (sync or async). In return, the user receives Mellow Vault shares representing their position.

To exit, users submit a redeem request through the Mellow async redeem queue, and once the request is settled, they claim wstETH back to their forwarder. The strategy also exposes methods to burn wstETH (reducing the minting obligation in the Pool) and to request withdrawal of stv from the Pool's WithdrawalQueue.

The audit was carried out over a period of 3 days by a team of 4 auditors, combining manual review with automated tooling.

Attack Vectors Analyzed

The codebase was reviewed for the following attack vectors:

- **Reentrancy attacks:** verification of reentrancy protection on all public functions interacting with external contracts
- **Access control bypass:** potential bypass of access control via safeTransferERC20 and direct interaction with Mellow queues
- **Oracle manipulation:** correctness of oracle report usage and handling of suspicious/invalid states
- **State corruption:** potential state corruption during concurrent calls or edge cases
- **Fund extraction:** unauthorized fund withdrawal or bad debt creation
- **Mellow integration consistency:** verification that Mellow protocol infrastructure behind Lido earnETH is integrated correctly including fees and shares calculations
- **Upgrade compatibility:** the behavior if earnETH vault is upgraded

Conclusion

The code is well-written with clear structure and good documentation. No critical vulnerabilities were found. Identified findings fall into Medium and Low severity categories. However, there are several recommendations for improvement.

MellowStrategy does not inherit ReentrancyGuard, and public functions make external calls to Mellow queues via forwarder without `nonReentrant` modifier. However, reentrancy exploitation is blocked by multiple layers: Mellow queues themselves carry `nonReentrant`, the forwarder's `doCall()` is restricted to `onlyOwner`, and MellowStrategy holds no mutable state that could be corrupted mid-call.

Users can transfer Mellow shares from their forwarder via `safeTransferERC20` and redeem directly through Mellow, bypassing the strategy's `REDEEM_FEATURE` pause and event tracking. This does not create bad debt — the user's `mintedStethShares` obligation remains unchanged, STV tokens stay locked proportionally, and the user only complicates their own exit flow. This behavior is not considered a vulnerability, but this flow should be considered as possible and documented.

When oracle report is valid but flagged as `isSuspicious`, the strategy blocks supply/exit requests while direct Mellow queue access remains available. Additionally, there is no wrapper for `DepositQueue.cancelDepositRequest()` — consider adding for feature parity with direct Mellow usage.

The Mellow Protocol contracts (DepositQueue, RedeemQueue, ShareManager, Oracle, FeeManager, Vault) were out of scope of this audit; they were only reviewed and their documentation was studied.

Overall, the MellowStrategy implementation is production-ready with appropriate security measures in place. The identified issues do not pose significant risks to user funds or protocol integrity. The main recommendations focus on edge cases handling, documentation improvement and improving UX parity with direct Mellow usage bypassing this adapter.

1.3 Project Overview

Summary

Title	Description
Client	Lido
Category	Liquid Staking
Project	DeFi Wrapper Mellow Strategy
Type	Solidity
Platform	EVM

Title	Description
Timeline	23.02.2026 – 11.03.2026

Scope of Audit

File	Link
src/factories/MellowStrategyFactory.sol	src/factories/MellowStrategyFactory.sol
src/strategy/MellowStrategy.sol	src/strategy/MellowStrategy.sol

Versions Log

Date	Commit Hash	Note
23.02.2026	2f474d2c61f0b904bba7782d68d9df8e8cdd8ec4	Initial Commit
10.03.2026	37b09995b8695becbaa6bcdbcfcdbc3838b8c926	Commit for Re-audit

Mainnet Deployments

The deployed contracts were verified on Ethereum mainnet. The deployed bytecode of both contracts matches the audited source code, and the on-chain configuration parameters were reviewed and confirmed to be correctly set.

File	Address	Blockchain
MellowStrategyFactory	0x8Fac09FD82F031D390B94622E2E4baBf16Fd2236	Ethereum
StrategyCallForwarder	0x7305bB45aF91893B7BCaF0Ad8Eae37cb16820Bb8	Ethereum

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	2
Low	7

Findings Statuses

ID	Finding	Severity	Status
M-1	Preview functions compute fees and shares differently from Mellow's actual queues	Medium	Fixed
M-2	`finalizeRequestExit()` can be called with a crafted requestId that maps to a valid timestamp	Medium	Fixed
L-1	Immutable references to Mellow managers become stale if Mellow vault is upgraded	Low	Fixed
L-2	`finalizeRequestExit` succeeds with zero claimed assets	Low	Fixed
L-3	Empty `supply()` succeeds without action	Low	Fixed
L-4	ETH can get stuck on StrategyCallForwarder	Low	Fixed
L-5	Missing `__AccessControlEnumerable_init()` call	Low	Fixed
L-6	`claimShares`, `requestWithdrawalFromPool`, `burnWsteth`, and `safeTransferERC20` do not emit events	Low	Acknowledged
L-7	`_requestExit()` balance check includes claimable shares, but `redeem()` only uses active shares	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Preview functions compute fees and shares differently from Mellow's actual queues		
Severity	Medium	**Status**	Fixed in 37b09995

Description

The preview functions in MellowStrategy.sol diverge from the actual Mellow queue implementations in fee and share calculations.

previewSupply applies fee and penalty as sequential multiplications on the shares amount:

```
shares = assets * priceD18 / 1e18
shares = shares * (1e6 - depositFeeD6) / 1e6  // fee applied first
shares = shares * (1e6 - penaltyD6) / 1e6  // penalty applied second (sync only)
```

The actual SyncDepositQueue.deposit() applies penalty to the price first, then computes shares, then subtracts the fee:

```
priceD18 = report.priceD18 * (1e6 - penaltyD6) / 1e6  // penalty on price first
shares = assets * priceD18 / 1e18
feeShares = shares * depositFeeD6 / 1e6  // fee subtracted from shares
shares = shares - feeShares
```

The mismatch is twofold. First, the order of operations is swapped: the strategy applies fee-then-penalty, while Mellow applies penalty-then-fee, and these are not algebraically equivalent when both are non-zero. Second, the fee formula itself differs — the strategy uses shares * (1e6 - feeD6) / 1e6 (single truncation), while Mellow uses shares - floor(shares * feeD6 / 1e6) (subtraction after a separate truncation). The latter is always greater or equal because floor(a * (1-f)) <= a - floor(a * f).

previewWithdraw computes redeem fees using a gross-up formula shares = ceil(shares * 1e6 / (1e6 - redeemFeeD6)), while RedeemQueue.redeem() applies fees = floor(shares * redeemFeeD6 / 1e6); shares -= fees to the input shares. Due to rounding differences between these two approaches,

the net shares after fee subtraction may differ from what previewWithdraw predicted by up to 1 wei of shares. Since assets in requestExitByWsteth is inherently an estimate — the actual wstETH amount depends on the oracle price at a future report time — this 1-wei rounding discrepancy has negligible practical impact.

Recommendation

For previewSupply, replicate Mellow's exact calculation order on the sync path: apply penalty to price, compute shares, then subtract calculateDepositFee(shares). For previewWithdraw, align the fee calculation with the formula used in RedeemQueue.redeem().

Client's Commentary:

Fixed

M-2	finalizeRequestExit() can be called with a crafted requestId that maps to a valid timestamp		
Severity	Medium	Status	Fixed in 37b09995

Description

finalizeRequestExit() casts requestId to a timestamp via uint32 timestamp = uint32(uint256(requestId)). An attacker can supply a fabricated requestId whose lower 32 bits match a legitimate timestamp. The Mellow RedeemQueue.claim() call will succeed using the truncated timestamp, finalizing the underlying request, but MellowStrategy emits a StrategyExitFinalized event with the fake requestId. The legitimate requestId never gets its own event, which corrupts off-chain tracking and indexing.

Recommendation

Validate requestId against the actual value returned by _requestExit or store valid request IDs and check membership before finalizing.

Client's Commentary:

Fixed

2.4 Low

L-1	Immutable references to Mellow managers become stale if Mellow vault is upgraded		
Severity	Low	**Status**	Fixed in 37b09995

Description

In MellowStrategy.sol, the constructor stores feeManager(), oracle(), and shareManager() as immutables:

```
MELLOW_FEE_MANAGER = vault_.feeManager();
MELLOW_ORACLE = vault_.oracle();
MELLOW_SHARE_MANAGER = vault_.shareManager();
```

The Mellow vault stores these as upgradeable storage variables in ShareModule.__ShareModule_init(). While the current Mellow code does not expose setter functions for these values post-initialization, the vault itself sits behind a proxy. No setters exist in the current implementation — __ShareModule_init() sets them once during initialization, and there are no post-init setter functions in the current Mellow codebase. However, if Mellow governance upgrades the vault implementation and the new implementation changes any manager address, the strategy would continue using the old addresses. This would cause previewSupply/previewWithdraw/previewRedeem to read stale fees and prices, and sharesOf() to query the wrong share manager.

Recommendation

Read feeManager(), oracle(), and shareManager() dynamically from the vault instead of caching them as immutables.

Client's Commentary:

Fixed

L-2	finalizeRequestExit succeeds with zero claimed assets		
Severity	Low	**Status**	Fixed in 37b09995

Description

In MellowStrategy.sol, finalizeRequestExit calls redeemQueue.claim() and emits StrategyExitFinalized regardless of the claimed amount. RedeemQueue.claim() returns 0 when no oracle report exists yet, when the request timestamp is beyond the latest report, or when the batch hasn't been processed via handleBatches() yet.
The function doesn't revert or warn when assets == 0, so a user calling it prematurely gets a success event with zero assets. The underlying Mellow request remains in the queue (it's skipped, not deleted, when the batch isn't ready), but the misleading event can confuse off-chain tracking.

Recommendation

Add if (assets == 0) revert or document this behavior.

Client's Commentary:

Fixed

L-3	Empty supply() succeeds without action		
Severity	Low	**Status**	Fixed in 37b09995

Description

Calling supply(referral, 0, params) with msg.value=0 does not revert. previewSupply(0) returns (true, 0) — success with 0 shares. No ETH deposit occurs, no Mellow deposit occurs, and the transaction succeeds with gas wasted and empty events emitted.
Similarly, previewSupply(0) returns (true, 0) indicating success, which may confuse integrators expecting failure for zero-amount operations.

Recommendation

Add a zero-amount check at the beginning of supply():

```
if (assets == 0 && msg.value == 0) revert ZeroArgument("assets or msg.value");
```

Client's Commentary:

Fixed

L-4	ETH can get stuck on StrategyCallForwarder		
Severity	Low	**Status**	Fixed in 37b09995

Description

`StrategyCallForwarder` has `receive() external payable {}` – it accepts ETH. It also has `sendValue()` – which can send ETH, but is restricted to `onlyOwner`. `MellowStrategy` is the owner but has no method to call `sendValue()`. ETH sent to the forwarder is permanently stuck.

Recommendation

Add an ETH recovery method to `MellowStrategy`.

Client's Commentary:

Fixed

L-5	Missing `__AccessControlEnumerable_init()` call		
Severity	Low	**Status**	Fixed in 37b09995

Description

`MellowStrategy.initialize()` does not call `__AccessControlEnumerable_init()`. In OpenZeppelin 5.x, `__AccessControlEnumerable_init()` is an empty function with no actual initialization logic, so there is no functional impact.

Recommendation

Add the call for consistency.

Client's Commentary:

Fixed

L-6	claimShares, requestWithdrawalFromPool, burnWsteth, and safeTransferERC20 do not emit events		
Severity	Low	**Status**	Acknowledged

Description

The functions claimShares(), requestWithdrawalFromPool(), burnWsteth(), and safeTransferERC20() do not emit any events upon completion. This makes it difficult for off-chain systems to track and index these operations.

Recommendation

Add appropriate events for each of these functions to improve observability.

Client's Commentary:

Acknowledged.

requestWithdrawalFromPool / burnWsteth has own events in Withdrawal/Dashboard contracts

claimShares() - added MellowSharesClaimed

safeTransferERC20 - ack

L-7	`_requestExit()` balance check includes claimable shares, but `redeem()` only uses active shares		
Severity	Low	**Status**	Acknowledged

Description

`MellowStrategy._requestExit()` checks the user's share balance including both active and claimable shares. However, `MELLOW_ASYNC_REDEEM_QUEUE.redeem()` only considers active shares. This can lead to a late revert — the strategy's own check passes but the Mellow call fails — when a user's active shares are insufficient but their total (active + claimable) shares would cover the amount.

Recommendation

Align the balance check in `_requestExit()` to only consider active shares, matching the behavior of `redeem()`.

Client's Commentary:

Client: *Not an issue. Left the comments in the commit*

MixBytes: *As it was stated in the comments of the `_requestExit` function, the described behavior is by design: TokenizedShareManager (the primary integration) auto-claims shares during redeem, making the check correct in practice. For BasicShareManager, the mismatch between the balance check and the redeem call can cause a revert.*

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