MixBytes()

Lido DeFi Wrapper Security Audit Report

FEBRUARY 02, 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

The audited protocol is a staking pool system built on Lido's v3 Staking Vault infrastructure, implementing three pool types with varying capabilities: StvPool (basic staking), StvStETHPool (with stETH minting and liability management), and StvStrategyPool (with external strategy integration). The core architecture allows users to deposit ETH and receive STV tokens representing their share, with advanced pools enabling users to mint stETH against their deposits at configurable reserve ratios. The system features a comprehensive withdrawal queue mechanism with batch processing and finalization, role-based access control for administrative functions, and sophisticated rebalancing capabilities to manage user liabilities.

The audit was carried out over a period of 12 days by a team of 4 auditors, combining manual review with automated tooling

The following attack vectors were thoroughly examined during the security audit:

1. Loss socialization during withdrawal queue finalization: The `maxLossSocializationBP` limiter is enforced through `rebalanceMintedStethSharesForWithdrawalQueue`, preventing excessive loss socialization during both direct force rebalancing and withdrawal queue finalization.

2. The withdrawal queue behavior with `maxLossSocializationBP = 0` is by design and serves as a governance safety mechanism: When withdrawal requests become undercollateralized due to vault losses, the system first applies loss socialization by capping `stvToRebalance` to available STV. The `_checkAllowedLossSocializationPortion` check in `StvStETHPool._rebalanceMintedStethShares()` then provides a secondary protection against additional losses from rate changes during finalization itself. The default value of 0 means finalization will revert if any loss socialization is needed beyond the initial capping. This intentionally creates a stuck queue scenario that forces governance to determine an appropriate `maxLossSocializationBP` based on actual conditions. This attack vector was analyzed and confirmed to function as intended.

3. Non-recalculated STV rate during batch finalization: The STV rate is captured once at finalization start and used consistently for all batch requests. Remaining STV from rate changes is burned.

4. Factory two-phase deployment frontrunning: The deployment hash includes `msg.sender`, preventing frontrunning as only the original `createPoolStart` caller can successfully call `createPoolFinish`.

5. Vault disconnection during operations: VaultHub's NoLiabilitySharesShouldBeLeft check ensures proper disconnect procedure. When a vault enters disconnection state, withdrawals and minting are paused, preventing fund loss while maintaining system consistency.

6. ERC20 token recovery in GGVStrategy: The `recoverERC20` function allows users to recover tokens from their own call forwarder context. While unrestricted access to critical protocol tokens (wstETH, GGV shares) poses theoretical risk, the function operates within user-scoped context.

7. Vault bad debt scenario: If a vault enters bad debt (total value < total liability), the pool becomes effectively abandoned as resolving bad debt requires burning all pool ETH, resulting in zero STV token price.

8. The separation of concerns between the DeFi Wrapper and VaultHub: The Wrapper's functions (`calcStethSharesToMintForAssets` and `calcAssetsToLockForStethShares`) focus solely on the position's asset-to-liability ratio without incorporating VaultHub's `minimalReserve` constraint. This design ensures that the Wrapper remains decoupled from higher-layer policies that may vary across different vault implementations.
   Users interacting with the protocol must query both layers before executing minting operations:
   - Check the Wrapper via `remainingMintingCapacitySharesOf(account, ethToFund)` for position-specific capacity;
   - Check the VaultHub via `remainingMintingCapacityShares(ethToFund)` on the Dashboard contract for vault-wide constraints.

9. Node Operator Governance Controls and Withdrawal Queue Resilience: The protocol architecture provides vault owners with comprehensive governance mechanisms to manage Node Operator behavior and ensure withdrawal queue operations proceed as expected. Through the Dashboard interface, vault owners can `pauseBeaconChainDeposits` to prevent new validator activations and use `forceValidatorExit` to mandate exits when needed for withdrawal fulfillment.

The BoringVault GGVStrategy implementation was out of scope of this audit.

The audit identified 2 medium and 22 low severity findings. While the codebase demonstrates solid architecture with good test coverage and proper handling of critical security scenarios, special attention should be paid to addressing these issues before the deployment.

1.3 Project Overview

Summary

Title	Description
Client	Lido
Category	Liquid Staking
Project	DeFi Wrapper
Type	Solidity
Platform	EVM
Timeline	01.12.2025 - 29.01.2026

Scope of Audit

File	Link
src/AllowList.sol	AllowList.sol
src/Distributor.sol	Distributor.sol
src/Factory.sol	Factory.sol
src/StvPool.sol	StvPool.sol
src/StvStETHPool.sol	StvStETHPool.sol
src/WithdrawalQueue.sol	WithdrawalQueue.sol
src/factories/DistributorFactory.sol	DistributorFactory.sol
src/factories/GGVStrategyFactory.sol	GGVStrategyFactory.sol
src/factories/StvPoolFactory.sol	StvPoolFactory.sol
src/factories/StvStETHPoolFactory.sol	StvStETHPoolFactory.sol
src/factories/TimelockFactory.sol	TimelockFactory.sol
src/factories/WithdrawalQueueFactory.sol	WithdrawalQueueFactory.sol

File	Link
src/proxy/OssifiableProxy.sol	OssifiableProxy.sol
src/strategy/GGVStrategy.sol	GGVStrategy.sol
src/strategy/StrategyCallForwarder.sol	StrategyCallForwarder.sol
src/strategy/StrategyCallForwarderRegistry.sol	StrategyCallForwarderRegistry.sol
src/utils/FeaturePausable.sol	FeaturePausable.sol

Versions Log

Date	Commit Hash	Note
01.12.2025	ac004a859f1fb2c33b0e5767ffbffc78651130fe	Initial Commit
21.01.2026	4f519289df0fcb553a0e97c20148f35149d7f516	Commit for Re-audit
23.01.2026	16b3b106a01f65caf4c24f76baf3db4f730e0e89	Commit with Updates
26.01.2026	3c903b7f1b75ac12a81557a52973d6b367c1c62c	Commit with Updates
29.01.2026	f35fe13ddca2084d7b848172b2698feb4fa88025	Commit with Updates

Mainnet Deployments

The deployment has been verified and the bytecode confirmed to match the audited commit f35fe13ddca2084d7b848172b2698feb4fa88025. The 0x3f221b8E5bC098cC6C23611BEeacaeCfD77e1587 parameters have been verified to be correct, including addresses of 0x9FD67B2D5b88BeBaC741EE50510cf808B4854a5F, 0x5Def7fBC0211351139B928f307EDC794af845Bde, 0x671978CEEa7DAf405fA08E930E1047d1b7b21a69, 0x15f2C9ea98e5564d25A46eE39D19704476998786, 0xB011531857B6006479627e776feB6c0cEA5fc74a and 0x468029A88b6f75Eb1D13BB291fC3B82fc2C0232F, as well as the addresses for 0x5DB427080200c235F2Ae8Cd17A7be87921f7AD6c, 0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84, 0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0, 0x1d201BE093d847f6446530Efb0E8Fb426d176709, and 0x02Ca7772FF14a9F6c1a08aF385aA96bb1b34175A.

File	Address
Factory.sol	0x3f221b8E5bC098cC6C23611BEeacaeCfD77e1587
DistributorFactory.sol	0x9FD67B2D5b88BeBaC741EE50510cf808B4854a5F
StvPoolFactory.sol	0x5Def7fBC0211351139B928f307EDC794af845Bde
StvStETHPoolFactory.sol	0x671978CEEa7DAf405fA08E930E1047d1b7b21a69
TimelockFactory.sol	0x15f2C9ea98e5564d25A46eE39D19704476998786
WithdrawalQueueFactory.sol	0xB011531857B6006479627e776feB6c0cEA5fc74a
DummyImplementation.sol	0x468029A88b6f75Eb1D13BB291fC3B82fc2C0232F

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	22

Findings Statuses

ID	Finding	Severity	Status
M-1	Missing Oracle Freshness Check in `transferWithLiability` Can Worsen Recipient Position	Medium	Fixed
M-2	Stale Reserve Ratio Can Worsen Recipient Position in `transferWithLiability` Call	Medium	Acknowledged
L-1	Malicious Node Operator Can Revert Withdrawal Finalization	Low	Acknowledged
L-2	assetsOf Does Not Account for Slashing Socialization	Low	Fixed
L-3	Unclear Revert Message When Finalizing During Vault Pending Disconnection	Low	Acknowledged
L-4	Redundant Initializable Inheritance	Low	Fixed
L-5	Missing Report Freshness Check in View Functions	Low	Acknowledged
L-6	Deposit Before LazyOracle Update Allows Yield Stealing	Low	Acknowledged
L-7	Preview Function Can Return Zero	Low	Acknowledged
L-8	Front-Running Risk in rebalanceUnassignedLiabilityWithEther	Low	Acknowledged

L-9	Missing Basic Parameter Validation in createPoolStart	Low	Fixed
L-10	ALLOW_LIST_MANAGER_ROLE Role Not Revoked from Factory	Low	Fixed
L-11	Missing Validation for Minimum Withdrawal Request After Partial Withdrawal	Low	Acknowledged
L-12	findCheckpointHintBatch can revert if one of hints is not found	Low	Fixed
L-13	Unused Error Declarations	Low	Fixed
L-14	Functions Can Be Declared External	Low	Fixed
L-15	Missing Input Validation for Proposer in TimelockFactory	Low	Fixed
L-16	Gas Optimizations in StvStETHPool	Low	Fixed
L-17	Missing Input Validation in Multiple Functions	Low	Fixed
L-18	Incorrect Type Conversion Causes Potential Silent Truncation of GGV Withdrawal Amounts	Low	Fixed
L-19	Token Recovery Does Not Handle Non-Reverting Transfer Failures	Low	Fixed
L-20	Unrestricted Maximum Loss Socialization Parameter	Low	Acknowledged
L-21	Missing Zero Address Validation for Finalizer Role	Low	Fixed
L-22	Malicious Strategy Can Be Used to Steal User Deposits	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Missing Oracle Freshness Check in transferWithLiability Can Worsen Recipient Position		
Severity	Medium	**Status**	Fixed in 4f519289

Description
This issue has been identified within the transferWithLiability function of the StvStETHPool contract.
The function does not require a fresh oracle report before validating the transferred stv amount against the transferred stETH-share liability. As a result, an attacker who anticipates the next oracle update can transfer less stv than will actually be required under the updated pricing, while still passing the current checks. After the oracle report updates, the recipient's minting capacity may decrease, and in some cases the recipient's position may become unhealthy.
The issue is classified as Medium severity because while it enables third-party griefing that can degrade a victim's collateralization ratio, the recipient still receives net positive value (assets minus liability). Actual loss requiring the position to become unhealthy would require the STV rate to decrease by more than the reserve ratio in a single oracle update, which is highly unlikely under normal network conditions.

Recommendation
We recommend enforcing an oracle freshness check (e.g., _checkFreshReport()) within transferWithLiability so that liability transfers are validated against up-to-date pricing.
We also recommend making the transferWithLiability function permissioned.

Client's Commentary:
Client: *Oracle freshness check has been added to the transferWithLiability function.*
However, we believe the severity is overstated. While such a scenario may worsen the recipient's relative position (assets-to-liability ratio), their absolute position improves as they receive additional value (assets minus liability) despite reduced minting capacity.
The exception requiring actual loss would need an oracle report decreasing the STV rate by more than the reserve ratio in a single update. This would require either unprecedented mass slashing (which the network has never experienced and against which oracle freshness check provides no protection), or prolonged penalty accumulation without applied oracle reports

(unlikely since many operations require fresh reports). Such situation would trigger unhealthy vault state earlier, requiring forceRebalance which itself needs an oracle report.
We consider this probability extremely low.
MixBytes: *We agree to downgrade this finding to Medium severity, taking into account the low probability but still viable scenario. The downgrade appropriately reflects both the theoretical griefing capability and the low practical likelihood of material harm, especially with the implemented oracle freshness check.*

M-2	Stale Reserve Ratio Can Worsen Recipient Position in transferWithLiability Call		
Severity	Medium	**Status**	Acknowledged

Description

This issue has been identified within the transferWithLiability function of the StvStETHPool contract.

The function validates the minimum stv amount required to accompany a transferred stETH-shares liability using the pool's locally stored reserveRatioBP. If the reserve ratio has changed in the VaultHub but the pool has not been synchronized (via syncVaultParameters), this check may use an outdated value and allow transferring less stv than is actually required under the current parameters. As a result, the recipient's minting capacity may decrease, and in some cases the recipient's position may become unhealthy after synchronization, potentially exposing them to forced rebalancing.

The issue is classified as Medium severity because it can be exploited to grief recipients by degrading their collateralization and potentially triggering forced rebalancing, without requiring the recipient's consent. At the same time, changing a reserve ratio seems to be a rare event.

Recommendation

We recommend ensuring that transferWithLiability uses up-to-date vault parameters, for example by synchronizing parameters before performing the minimum-lock check (or by deriving the effective reserve ratio directly from current VaultHub/Dashboard values and reverting if parameters are stale).

It might be worth adding more incentives for an actor in the syncVaultParameters call.

We also recommend making the transferWithLiability function permissioned.

Client's Commentary:

Acknowledged

Changes to vault parameters such as the reserve ratio are expected to be very rare and operationally heavy. A vault is required to already satisfy the new reserve requirements before such a change can be applied, which is quite difficult to achieve without users participation. In practice, it is unlikely that reserve parameters will change on Vaults used with the Wrapper. If they do, this would happen in a highly manual and coordinated manner.

Given this, the risks described are intended to be mitigated operationally through careful coordination of parameter updates, and the likelihood of this scenario occurring in practice is considered low.

2.4 Low

L-1	Malicious Node Operator Can Revert Withdrawal Finalization		
Severity	Low	**Status**	Acknowledged

Description

The `finalize` function in `WithdrawalQueue.sol` calls `DASHBOARD.rebalanceVaultWithShares(remainingStethShares)` which requires ETH to be available on the vault. A malicious Node Operator (NO) can deposit unused ETH on the vault, causing the rebalance operation to fail or revert, which in turn blocks the withdrawal finalization process. This creates a DoS attack vector where withdrawals cannot be finalized.
This is classified as Low severity because the protocol provides existing mitigation mechanisms (`pauseBeaconChainDeposits` and `forceValidatorExit`) that vault owners can use to address such behavior, and the impact is limited to temporary delays rather than fund loss.

Recommendation

There are mechanisms available in the protocol that can force a Node Operator not to use available ETH for deposits on new validators, but these can only be used in emergency cases. We propose clearly stating such cases and how they will be handled in the documentation.

Client's Commentary:
Client: *The described behavior is incorrect. The finalize method accounts for available ETH balance and does not revert in this scenario. Finalization will simply be limited to the number of requests that can be processed with the available ETH. Revert may occur if there is nothing to finalize, but not due to a lack of ETH.*
In the normal workflow, the Node Operator deposits ETH to validators, exits validators to fulfill withdrawal requests, and finalizes those requests. If a Node Operator is malicious or fails to fulfill their duties (including depositing ETH instead of processing withdrawals), the vault owner has the necessary mechanisms to address this: pause deposits via pauseBeaconChainDeposits and force validator exits.
We consider this an emergency scenario, and the protocol already provides the required mechanisms to handle it.
MixBytes: *We decided to downgrade this finding to Low severity. The `finalize` function actually queries both `withdrawableValue` and `availableBalance` and processes requests in a loop that breaks when ETH availability constraints are reached. The only revert condition is when `finalizedRequests == 0`, which occurs when there are no processable requests.*
The Low severity classification reflects that vault owners have comprehensive governance controls to address Node Operator misbehavior and the impact is limited to operational workflow management rather than potential withdrawal finalization process blocking.

L-2	assetsOf Does Not Account for Slashing Socialization		
Severity	Low	**Status**	Fixed in 4f519289

Description

The assetsOf function in StvPool.sol returns nominalAssetsOf(_account) instead of using totalAssets() to calculate the real ETH value for the user. This means that if the pool is used in integrations that check user ETH balance, the integration will not account for possible slashing socialization in the pool. The function does not reflect the actual value after accounting for slashing socialization events, which can lead to incorrect balance calculations in external integrations.

Attack Vector:

1. A slashing socialization event occurs, reducing the actual value of the pool
2. An external integration calls assetsOf to check a user's ETH balance
3. The function returns the nominal assets without accounting for socialization
4. The integration makes decisions based on incorrect balance information
5. This can lead to incorrect calculations, failed transactions, or exploitation of the discrepancy

Recommendation

We recommend updating the assetsOf function to use totalAssets() for calculating the real ETH value, accounting for slashing socialization.

Client's Commentary:

Client: *Fixed: PR-85*

The severity is overstated, as slashing socialization should never occur on a Vault that does not mint stETH.

MixBytes: *We agree to downgrade this finding to Low severity. While the base StvPool uses nominalAssetsOf without accounting for totalUnassignedLiabilitySteth, this scenario requires unassigned liability to exist in the vault, which only occurs when liability is transferred from another vault or during vault rebalancing operations bypassing the wrapper, which are rare operational events. Also, the impact is limited to potential mispricing in external integrations rather than direct fund loss. The StvStETHPool already correctly overrides this function to use _convertToAssets, demonstrating that the extended contract properly handles this case.*

L-3	Unclear Revert Message When Finalizing During Vault Pending Disconnection		
Severity	Low	**Status**	Acknowledged

Description
The withdrawal finalization process in `WithdrawalQueue.sol` will be blocked when the vault is in a pending disconnection state. When a vault enters pending disconnection, `DASHBOARD.withdrawableValue()` returns 0. This causes the finalization loop to immediately break at the first request check (line 544 in `WithdrawalQueue.sol`), since any withdrawal request will fail the condition `(ethToClaim + gasCostCoverage) > withdrawableValue`. The function then reverts with `NoRequestsToFinalize()` error, which doesn't clearly indicate that the vault is in a pending disconnection state.
The error message is misleading to users who may not understand why their withdrawal requests cannot be finalized.

Recommendation
We recommend adding an explicit check at the beginning of the `finalize()` function to verify that the vault is not in a pending disconnection state using `VAULT_HUB.isPendingDisconnect(address(VAULT))` or ensure that `DASHBOARD.withdrawableValue()` is greater than 0. If the vault is in pending disconnection, revert with a descriptive error message such as `VaultPendingDisconnection()` to inform users why finalization cannot proceed.

Client's Commentary:
Acknowledged
When a vault enters pending disconnection from the VaultHub, the WithdrawalQueue is not expected to operate. The Wrapper is tightly coupled to VaultHub functionality and is not designed to function with a Vault that is disconnected or in the process of disconnection.
Vault disconnection implies pool wind-down and liquidation. As part of this procedure, the WithdrawalQueue is paused and all withdrawal requests are expected to be finalized before initiating the vault disconnection.

L-4	Redundant Initializable Inheritance		
Severity	Low	**Status**	Fixed in 4f519289

Description

The StvPool contract explicitly inherits from Initializable, but Initializable is already inherited in the AllowList contract. This creates redundant inheritance that should be removed for cleaner code.

Recommendation

Remove Initializable from the StvPool contract's inheritance list, as it is already inherited through AllowList.

Client's Commentary:
Fixed: PR-83

L-5	Missing Report Freshness Check in View Functions		
Severity	Low	**Status**	Acknowledged

Description
View functions such as `assetsOf()` and `totalAssets()` in `StvStETHPool` return state data that depends on oracle reports, but they do not provide any indication of whether the underlying report is fresh. While integrations can separately call `VAULT_HUB.isReportFresh(address(VAULT))` to check freshness, this requires an additional external call. Adding reverts to existing view functions would break the read path for integrations that prefer resilience over strict freshness guarantees.

Recommendation
We recommend implementing companion view functions that return both the value and a freshness indicator. The enhanced functions should call `VAULT_HUB.isReportFresh(address(VAULT))` internally and include the freshness status in the return value. Document these functions with comments indicating they are the safer option for integrations requiring strong freshness guarantees. This approach provides flexibility: integrations can choose the simple function for resilience or the enhanced function for safety.

Client's Commentary:
Acknowledged
The opposite trade-off is also valid: reverting in view functions due to stale reports may be more harmful for integrations than returning slightly outdated data, as it can unexpectedly break read paths and downstream logic.
For tighter integrations that require strong freshness guarantees, the protocol provides a way to verify data recency by checking report freshness directly in the VaultHub. Such integrations are expected to perform this validation explicitly, rather than relying on implicit reverts in view functions.

L-6	Deposit Before LazyOracle Update Allows Yield Stealing		
Severity	Low	**Status**	Acknowledged

Description
Deposits are possible before the LazyOracle update, which can allow yield stealing. The damage is limited to within 1 day of rewards, which is acceptable, but the quarantine case is interesting because it can increase rewards abruptly.

Recommendation
Consider adding checks or documentation about the timing of deposits relative to oracle updates. While the impact is limited, it's worth documenting this behavior and potentially adding safeguards if the risk increases.

Client's Commentary:
Acknowledged
This behavior is known and its impact is intentionally limited. Deposits require a fresh oracle report, which restricts entry to a single reporting frame. Exiting is not immediate: withdrawals must wait at least one oracle frame in the WithdrawalQueue. In addition, withdrawals finalization and validator deposits are controlled operationally by the node operator. A sophisticated actor cannot rely on immediate exit, as their ETH may remain in the pool for some time and be deposited to validators, effectively spreading any potential gains over time.
Additionally, large abrupt increases in the STV rate are expected to be very rare for the wrapper. Unlike generic vaults, the wrapper is funded only by user deposits; side deposits are not expected and unguaranteedDeposit is disallowed. As a result, scenarios that would allow extracting meaningful profit from a short-term observed opportunity are expected to be limited to exceptional cases, such as unusually large EL reward spikes or donations.

L-7	Preview Function Can Return Zero		
Severity	Low	**Status**	Acknowledged

Description
In a very improbable scenario, the previewDeposit function can return 0, which would result in 0 stv being minted for a deposit. This edge case should be handled to prevent unexpected behavior.

Recommendation
Add a check after calling previewDeposit to ensure the returned value is greater than 0. If it returns 0, revert the transaction with a clear error message explaining why the deposit cannot proceed.

Client's Commentary:
Acknowledged
This is a known and extremely unlikely edge case where previewDeposit() may return 0. In practice, this scenario would only be possible if the vault lost all assets, including the initial deposit, which could happen only in extreme situations such as a catastrophic hack or full validator slashing combined with zero EL balance. This is considered extraordinarily unlikely.
While reverting on a 0 return value would prevent the immediate outcome, it does not fully cover related edge cases around very small deposits and may introduce other issues (e.g., inflation-attack windows). A complete mitigation would add complexity to the codebase.
In such an extreme scenario, we expect this to be mitigated operationally by manually pausing deposits.

L-8	Front-Running Risk in rebalanceUnassignedLiabilityWithEther		
Severity	Low	**Status**	Acknowledged

Description

The rebalanceUnassignedLiabilityWithEther function can be front-run to distribute losses between pool holders. An attacker can monitor the mempool for transactions that will cause losses and front-run them to ensure the losses are distributed among all holders rather than affecting them directly.

Recommendation

Add a time gap or cooldown period between a loss event occurring and when rebalanceUnassignedLiabilityWithEther can be called. Alternatively, implement access controls or rate limiting to prevent front-running attacks.

Client's Commentary:

Acknowledged

The rebalanceUnassignedLiabilityWithEther function combines rebalanceUnassignedLiability and loss compensation in one operation. If the transaction is front-run, the actor who intended to compensate pool losses can simply make a direct ETH transfer to the StakingVault in a subsequent transaction. No additional mitigation is required.

L-9	Missing Basic Parameter Validation in createPoolStart		
Severity	Low	**Status**	Fixed in 4f519289

Description
The createPoolStart function in Factory.sol lacks basic validation checks for parameters. At minimum, it should validate zero addresses and ensure that minDelay is within a reasonable range (not too huge).

Recommendation
Add validation checks at the beginning of createPoolStart to ensure:
- No zero addresses are passed for critical parameters
- minDelay is within a reasonable range (e.g., not exceeding a maximum threshold)

Client's Commentary:
Partially fixed, partially acknowledged.
Checks for non-zero proposer and executor are added in PR-91
No limits for minDelay are added because the reasonable range hard to identify.

L-10	ALLOW_LIST_MANAGER_ROLE Role Not Revoked from Factory		
Severity	Low	**Status**	Fixed in 4f519289

Description
After pool deployment, the ALLOW_LIST_MANAGER_ROLE role is not revoked from the factory. This means the factory retains the ability to manage the allowlist even after the pool is fully deployed and ownership is transferred to the timelock.

Recommendation
Add a step in the pool deployment process to revoke the ALLOW_LIST_MANAGER_ROLE role from the factory after the pool is fully deployed and ownership is transferred. This ensures the factory cannot interfere with the pool's allowlist management after deployment.

Client's Commentary:
Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-11	Missing Validation for Minimum Withdrawal Request After Partial Withdrawal		
Severity	Low	**Status**	Acknowledged

Description
In the GGVStrategy.sol contract, when processing a withdrawal request, there is no check to ensure that after a partial withdrawal, the user will still be able to withdraw the remaining amount (i.e., that the remaining amount is not less than the minimum withdrawal request threshold).

Recommendation
Add a validation check after processing the withdrawal to ensure that if there is a remaining balance, it meets the minimum withdrawal request requirement. If the remaining balance is below the minimum, consider either reverting the transaction or providing a clear warning to the user.

Client's Commentary:
This finding is not specific enough: the system has two distinct exit/withdraw paths (requestExitByWsteth vs requestWithdrawalFromPoolGGV and wrapper-side). We also do not consider enforcing "post-partial remainder must be >= minimum" a security requirement; it can block integrations/users from withdrawing their full available amount. If a minimum exists, users can top up and submit a minimum-sized request later.

L-12	findCheckpointHintBatch can revert if one of hints is not found		
Severity	Low	Status	Fixed in 4f519289

Description

This issue has been identified within the findCheckpointHintBatch function of the WithdrawalQueue contract.
If findCheckpointHint() returns NOT_FOUND (0) for some request ID, the function assigns _firstIndex = hintIds[i], which sets _firstIndex to 0. On the next iteration, findCheckpointHint() is called with _start == 0, causing a revert (via InvalidRange). As a result, a single "not found" hint can make the entire batch call revert instead of returning hints for the remaining request IDs.
The issue is classified as Low severity because it affects a view helper used for hint discovery and can be worked around, but it can still break batch UX and integrations relying on this method.

Recommendation

We recommend handling the NOT_FOUND case explicitly (e.g., do not update _firstIndex when the hint is 0).

Client's Commentary:
Fixed: PR-86

L-13	Unused Error Declarations		
Severity	Low	**Status**	Fixed in 4f519289

Description
Several error declarations exist in the codebase, but are never used:
1. `StringTooLong(string str)` in `Factory.sol` - declared but never thrown
2. `ArraysLengthMismatch(uint256, uint256)` in `StvStETHPool.sol` - declared but never used

These unused declarations increase contract bytecode size and may cause confusion during code review.

Recommendation
Remove the unused error declarations from both contracts.

Client's Commentary:
Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-14	Functions Can Be Declared External		
Severity	Low	**Status**	Fixed in 4f519289

Description
Several public view functions are never called internally and could be declared `external` instead of `public`. Using `external` is slightly more gas-efficient for functions that are only called externally.
Affected functions:
- `WithdrawalQueue.getLastRequestId()`
- `WithdrawalQueue.getLastFinalizedRequestId()`
- `StvStETHPool.totalMintingCapacitySharesOf()`

Recommendation
Change the visibility ot these functions from `public` to `external`.

Client's Commentary:
Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-15	Missing Input Validation for Proposer in TimelockFactory		
Severity	Low	**Status**	Fixed in 4f519289

Description
The `TimelockFactory.deploy()` function does not validate that the `_proposer` parameter is a non-zero address. If `address(0)` is passed as the proposer, the resulting `TimelockController` will have no valid proposer, effectively creating a DoS condition where no proposals can ever be submitted.
Since the factory is used during pool creation, this could result in pools being deployed with non-functional governance.

Recommendation
We recommend adding input validation for the `_proposer` parameter.

Client's Commentary:
Client: *Invalid & Fixed: PR-91*
In the TimelockController, the schedule and scheduleBatch methods are protected by the onlyRole modifier with the PROPOSER_ROLE. Setting PROPOSER_ROLE to address(0) does not allow any address to call these methods, as the modifier will revert for unauthorized callers.
The TimelockController does support open-access functionality, but only for the executor role. The execute and executeBatch methods use the onlyRoleOrOpenRole modifier with EXECUTOR_ROLE, which allows execution by anyone if the role is granted to address(0). This design is intentional for the executor role but does not apply to the proposer role.
MixBytes: *We changed the description of this finding to better reflect the described issue.*

L-16	Gas Optimizations in StvStETHPool		
Severity	Low	Status	Fixed in 4f519289

Description
In both forceRebalance() and forceRebalanceAndSocializeLoss(), the check for WITHDRAWAL_QUEUE address is performed after the expensive previewForceRebalance() call. Moving this check earlier would save gas when someone attempts to rebalance the withdrawal queue.

Recommendation
We recommend moving the WITHDRAWAL_QUEUE check before the previewForceRebalance() call.

Client's Commentary:
Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-17	Missing Input Validation in Multiple Functions		
Severity	Low	**Status**	Fixed in 4f519289

Description
Multiple functions lack proper input validation, allowing invalid parameters that either undermine contract functionality or waste gas:

1. TimelockFactory.deploy(): Does not validate the `_minDelaySeconds` parameter. An extremely short delay (e.g., 0 or a few seconds) would undermine the purpose of a timelock, while an unreasonably long delay could make governance impractical.

2. requestExitByWsteth(): No validation that `_wsteth > 0`. When `_wsteth` is 0 and the user has GGV shares, the function passes both existing checks (`totalWstethFromGGV == 0` and `_wsteth > totalWstethFromGGV`) and computes `ggvShares = 0`, resulting in wasted gas for a clearly invalid input.

Recommendation
Add appropriate input validation:
• Add minimum and maximum bounds for timelock delay
• Add early zero-amount validation for fail-fast behavior

Client's Commentary:
Fixed GGVStrategy: PR-87
Timelock: Acknowledged - no limits are added because the reasonable range hard to identify.

L-18	Incorrect Type Conversion Causes Potential Silent Truncation of GGV Withdrawal Amounts		
Severity	Low	**Status**	Fixed in 4f519289

Description

In requestExitByWsteth(), the ggvShares value (uint256) is incorrectly converted to int256 when calling requestOnChainWithdraw(). However, the BoringQueue interface shows this function expects a uint128 parameter. While the function call will succeed in most cases due to ABI encoding compatibility, if ggvShares exceeds type(uint128).max (which is highly unlikely), the value will be silently truncated when decoded by the receiving function. This could result in users requesting withdrawal of a much smaller amount than intended, potentially leading to loss of funds. The issue occurs at line 204 where ggvShares.toInt256() should be ggvShares.toUint128().

Recommendation

Change ggvShares.toInt256() to use SafeCast's toUint128() method instead, which will revert if the value exceeds the uint128 range rather than silently truncating.

Client's Commentary:

Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-19	Token Recovery Does Not Handle Non-Reverting Transfer Failures		
Severity	Low	**Status**	Fixed in 4f519289

Description

The `recoverERC20()` function in `GGVStrategy.sol` calls `IERC20.transfer()` via the call forwarder's `doCall()` method. While `doCall()` uses OpenZeppelin's `Address.functionCall()` which checks for call success, it does not verify the boolean return value from the `transfer()` call itself. Some ERC20 tokens return `false` on failure instead of reverting. In such cases, the transfer would fail silently without reverting the transaction.

Recommendation

Decode and verify the boolean return value from the `transfer()` call, or restructure the recovery mechanism to use SafeERC20's `safeTransfer` pattern which properly handles both reverting and non-reverting token failures.

Client's Commentary:

Fixed in commit 22289ba2283b92f5f3c0fd138e60c4e85d58943a

L-20	Unrestricted Maximum Loss Socialization Parameter		
Severity	Low	**Status**	Acknowledged

Description
The setMaxLossSocializationBP() function in StvStETHPool.sol allows the admin to set maxLossSocializationBP to any value up to and including 10000 basis points (100%). While the function does validate that the value does not exceed TOTAL_BASIS_POINTS at line 746, it permits setting the parameter to 100%, which would allow complete loss socialization. When loss socialization occurs during rebalancing operations, users who are not being rebalanced bear the loss proportionally. Allowing 100% loss socialization means that a single user's complete debt could be spread across all other pool participants.

Recommendation
Add an upper bound check to prevent maxLossSocializationBP from being set to excessively high values.

Client's Commentary:
Acknowledged
The default value for maxLossSocializationBP is 0, which serves as the safe upper bound prohibiting loss socialization. In rare critical scenarios, loss socialization may be necessary to restore pool functionality. The setMaxLossSocializationBP function is the mechanism for adjusting this bound when required, with changes going through TimelockController for governance oversight. Adding an additional upper bound for the upper bound setter itself would be redundant and could prevent legitimate emergency interventions.

L-21	Missing Zero Address Validation for Finalizer Role		
Severity	Low	**Status**	Fixed in 4f519289

Description
The initialize() function in WithdrawalQueue.sol does not validate that the _finalizer parameter is a non-zero address before granting the FINALIZE_ROLE at line 247. While the function does check _admin for zero address at line 242 and conditionally checks _withdrawalsPauser and _finalizePauser at lines 248-253, the _finalizer parameter is always granted the role without validation. If address(0) is passed as the finalizer during initialization, the FINALIZE_ROLE would be granted to the zero address.

Recommendation
Add a zero address check for the _finalizer parameter similar to the check performed for _admin. Either reject zero address with a revert, or make the role granting conditional like _withdrawalsPauser and _finalizePauser if having no finalizer is an acceptable configuration.

Client's Commentary:
Fixed. PR-93

L-22	Malicious Strategy Can Be Used to Steal User Deposits		
Severity	Low	**Status**	Acknowledged

Description
The createPoolStart function in Factory.sol accepts a _strategyFactory address and _strategyDeployBytes parameter that are fully controlled by the user. An attacker can create a malicious pool via this Factory by providing a malicious strategy factory address and deployment bytes. This malicious strategy can be designed to steal all users' deposits, as there is no whitelist or validation mechanism to ensure only legitimate strategies are used.
Attack Vector:

1. Attacker calls createPoolStart with a malicious _strategyFactory address and _strategyDeployBytes
2. The factory deploys a pool with the malicious strategy
3. Users deposit funds into the pool, believing it to be legitimate since it was created through the official Factory
4. The malicious strategy can drain or steal all deposited funds through arbitrary logic in the strategy contract

This issue is assigned a Low severity. While a malicious strategy could theoretically steal deposited funds, the Factory is designed as a permissionless deployment tool rather than a trust mechanism. Users must perform due diligence on the vault deployers and managers, not rely on the Factory for security guarantees. The real trust boundary lies with the entities controlling vault management roles and governance, making this a documentation concern rather than a protocol vulnerability.

Recommendation
We recommend implementing a whitelist mechanism for strategy factories. Only pre-approved and audited strategy factories should be allowed. Add a mapping or access control check that validates the _strategyFactory address against a whitelist before allowing pool creation. Additionally, consider implementing a governance mechanism to manage the whitelist.

Client's Commentary:
Client: *We respectfully disagree with this finding. The assumption that users would trust a vault simply because it was created through the official Factory is flawed.*
Using the Factory does not provide trust guarantees. Security depends on the entities managing the Wrapper contracts. A malicious deployer could set management roles to their own addresses during deployment and subsequently connect any strategy or upgrade critical contracts after users deposit funds.
Users must trust the party that deployed and manages the vault infrastructure. The flexible design is intentional, not a vulnerability.
Partners developing strategies are expected to follow Lido's security standards, conduct audits, verify deployments, and use trusted independent actors for management roles. Contracts deployed through the Factory are not automatically trusted. Any Lido integrations or UI display require individual partner screening.
MixBytes: *We agree to downgrade this to Low severity. The issue highlights an important aspect of the trust model that should be clearly documented for integrators and users, but does not represent a direct vulnerability in the protocol's intended design.*

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes