

MixBytes()

Lido v3 Security Audit Report

NOVEMBER 10, 2025

Table of Contents

1. Introduction	3
1.1 Disclaimer	3
1.2 Executive Summary	3
1.3 Project Overview	4
1.4 Security Assessment Methodology	6
1.5 Risk Classification	8
1.6 Summary of Findings	9
2. Findings Report	11
2.1 Critical	11
2.2 High	11
H-1 Delayed Bunker Mode Activation After Bad Debt Internalization	11
2.3 Medium	12
M-1 Incorrect Accounting in internalizeExternalBadDebt May Overestimate Protocol Losses	12
M-2 Incorrect Balance Increase Calculation Due to Validators' Exit	15
M-3 Vault Activation Possible During Bunker Mode, Enabling Bad Debt Attacks	16
M-4 Staking Can Be Resumed While Protocol Is Stopped, Breaking Core Invariant	17
M-5 smoothenTokenRebase Function doesn't Account for the badDebtToInternalize variable	18
2.4 Low	19
L-1 Max External Shares Ratio Update Can Be Front-Run	19
L-2 Lack of Minting Cap May Lead to Oversupply via mintShares	20
L-3 Incorrect Comment Regarding Validator Count Comparison	21
L-4 Inconsistent Accessor Usage for Packed Storage Slot	22
L-5 Missing Upper Bound Check on Total Shares During Mint	23
L-6 Vault Data Can Be Repeatedly Updated Without Restriction	24
L-7 Sanity Check Ignores Time Since Last Report	25

L-8 Unsafe Downcast of Share Limit in Group Registration	26
L-9 Function Visibility Can Be Changed	27
L-10 Arithmetic Style Improvement for Readability and Accuracy Assurance	28
L-11 Ignoring return value from withdrawRewards	29
L-12 Misleading Comment in the OperatorGrid.changeTier() Function	30
L-13 Missing sharesToMintAsFees in Smoothing Calculation Limits the Pulled EL Rewards	31
3. About MixBytes	32

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

Lido v3 is an update to the existing Lido v2 protocol, which introduces staking vaults logic. There are vault owners and node operators, who is able to connect permissionlessly, run their own validator and receive staking rewards. Together with that, vault stakers are able to mint `stETH` at a configured reserve ratio to their deposit in ETH. Lido v2 accounts for the external shares allocated for the vaults mints and processes accounting oracle report, which internalizes potential bad debt among `stETH` holders and updates the report data in the `LazyOracle` utilized by staking vaults.

The audit was conducted over 22 days by 4 auditors.

During this audit, the team immersed themselves in the entire new Lido v3 architecture. At the same time, the audit was conducted specifically for two contracts – Accounting and Lido. The main goal of the audit was to ensure that the external debt created by the vaults part does not harm the main part of Lido. It was also necessary to verify that all internal accounting occurs correctly. In addition to the checklist, the following vectors were checked and the following observations were made:

StETH Exchange Rate. The rate does not change during normal mint and burn operations of external shares. The only thing that affects the rate is bad debt distribution. Integrity of the `totalPooledEther` invariant was verified, ensuring it cannot be manipulated by malicious `VaultHub` users.

Migrations. Correctness of the Burner contract migration process. Correct implementation of the accounting logic migration from the Lido contract to the new Accounting contract. Accurate reading and updating of merged storage slots in the Lido contract following recent upgrades.

Calculations. Mathematical correctness of all formulas implemented across the Lido and Accounting contracts. Correctness of all inline `keccak256` hashes used across the smart contracts.

Theoretical Attacks. Sandwich attack on the report handling with huge bad debt internalization leading to negative rebase was analyzed. The user can theoretically exit their position by selling `stETH` on the market before the report is handled and then re-enter Lido at a lower rate. However, the selling price on the market will likely already account for the negative rebase, as there will probably be entities monitoring vaults health. Entering the withdrawal queue before

the bad debt was internalized from the vaults, leading to negative rebase, also makes no sense, as the lower StETH rate between entry and finalization will be taken.

In conclusion, the project demonstrates a high level of security, operational integrity, and good overall code quality. The contracts in scope are well-written and follow established best practices. However, several findings require careful consideration to ensure a balanced distribution of risks between the staking vaults and Lido V2. Addressing these issues will further enhance the protocol's efficiency and reliability.

An ongoing security contest for this project is still in progress, in which our team will participate as triagers. Therefore, the final codebase commit may change as the contest concludes. Additionally, we plan to perform a detailed verification of the deployment. This report represents an **interim** version and will be updated once the contest and final deployment review are completed.

1.3 Project Overview

Summary

Title	Description
Client Name	Lido
Project Name	Lido v3
Type	Solidity
Platform	EVM
Timeline	17.06.2025 - 07.11.2025

Scope of Audit

File	Link
<code>contracts/0.4.24/Lido.sol</code>	Lido.sol
<code>contracts/0.8.25/Accounting.sol</code>	Accounting.sol

Versions Log

Date	Commit Hash	Note
17.06.2025	22cab0f0372015f2d2fce8bede64e98beae28571	Initial Commit
18.09.2025	4af82f0d0851ec514b32c9ce40c7ac0cd2915d69	Commit for Re-audit
06.10.2025	d50d46573ff0254ea7d3a576cb59e31b2b186437	Commit with Updates
08.10.2025	88ce9647db982414133e5bab80aac399cb930106	Commit with Updates
23.10.2025	25c2be321bae1b379ee477db96714ab9a3aebf90	Commit with Updates
28.10.2025	b9839f5110bb7faf6e78892c2649c93b09c32883	Commit with Updates
29.10.2025	79870b4809fb48d6f8dc4b72edf3670826928b2f	Commit with Updates
07.11.2025	33006a9c68d880a608b7b537ffde9c7abae0d3c6	Commit with Updates

Mainnet Deployments

The deployment verification will be conducted later after the full deployment of the protocol into the mainnet.

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	<p>Project Architecture Review:</p> <ul style="list-style-type: none">• 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 <p>OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.</p>
	<p>Code Review with a Hacker Mindset:</p> <ul style="list-style-type: none">• 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. <p>OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.</p>
	<p>Code Review with a Nerd Mindset:</p> <ul style="list-style-type: none">• 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, Mytril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. <p>OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.</p>

Stage	Scope of Work
	<p>Consolidation of Auditors' Reports:</p> <ul style="list-style-type: none"> • Cross-check findings among auditors • Discuss identified issues • Issue an interim audit report for client review <p>OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.</p>
Re-audit	<p>Bug Fixing & Re-Audit:</p> <ul style="list-style-type: none"> • 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 <p>OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.</p>
Final audit	<p>Final Code Verification & Public Audit Report:</p> <ul style="list-style-type: none"> • 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 <p>OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.</p>

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: 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	1
Medium	5
Low	13

Findings Statuses

ID	Finding	Severity	Status
H-1	Delayed Bunker Mode Activation After Bad Debt Internalization	High	Acknowledged
M-1	Incorrect Accounting in <code>internalizeExternalBadDebt</code> May Overestimate Protocol Losses	Medium	Unknown
M-2	Incorrect Balance Increase Calculation Due to Validators' Exit	Medium	Acknowledged
M-3	Vault Activation Possible During Bunker Mode, Enabling Bad Debt Attacks	Medium	Fixed
M-4	Staking Can Be Resumed While Protocol Is Stopped, Breaking Core Invariant	Medium	Fixed
M-5	<code>smoothenTokenRebase</code> Function doesn't Account for the <code>badDebtToInternalize</code> variable	Medium	Acknowledged
L-1	Max External Shares Ratio Update Can Be Front-Run	Low	Acknowledged
L-2	Lack of Minting Cap May Lead to Oversupply via <code>mintShares</code>	Low	Acknowledged
L-3	Incorrect Comment Regarding Validator Count Comparison	Low	Fixed

L-4	Inconsistent Accessor Usage for Packed Storage Slot	Low	Fixed
L-5	Missing Upper Bound Check on Total Shares During Mint	Low	Fixed
L-6	Vault Data Can Be Repeatedly Updated Without Restriction	Low	Fixed
L-7	Sanity Check Ignores Time Since Last Report	Low	Acknowledged
L-8	Unsafe Downcast of Share Limit in Group Registration	Low	Fixed
L-9	Function Visibility Can Be Changed	Low	Fixed
L-10	Arithmetic Style Improvement for Readability and Accuracy Assurance	Low	Acknowledged
L-11	Ignoring return value from <code>withdrawRewards</code>	Low	Acknowledged
L-12	Misleading Comment in the <code>OperatorGrid.changeTier()</code> Function	Low	Fixed
L-13	Missing <code>sharesToMintAsFees</code> in Smoothing Calculation Limits the Pulled EL Rewards	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	Delayed Bunker Mode Activation After Bad Debt Internalization		
Severity	High	Status	Acknowledged

Description

The `VaultHub.internalizeBadDebt` function is responsible for internalizing vault bad debt into Lido V2. It reduces the vault's liability and incorporates the `badDebtToInternalize` value when calculating the `stETH` rebase during the oracle report.

However, the creation and internalization of bad debt can trigger bunker mode in Lido V2. This mode is only activated after the oracle report is submitted, which may occur some time after the bad debt has been internalized via `VaultHub`. During this delay, the protocol continues to operate normally—even though a known and acknowledged bad debt exists.

This issue is classified as **High** severity because it enables front-running of bunker mode activation. Users can exit the protocol without any restrictions before the oracle report is submitted, despite the fact that bunker mode activation is guaranteed once the report arrives.

Recommendation

We recommend monitoring the health of vaults in the off-chain logic and triggering bunker mode as soon as bad debt is detected—before it is internalized via `VaultHub.internalizeBadDebt`. This approach will more accurately reflect the current state of the protocol and help prevent bypassing exit limitations.

Client's Commentary:

Client: <https://github.com/lidofinance/core/issues/1338#issuecomment-3249283991>

Comment from the analytics team:

Even in the worst-case scenario, the protocol's impact occurs only at network-wide slashing levels above 1.4%.

Given the extremely low probability of these conditions, it is suggested to:

Not implement any new bunker mode conditions on initial stVaults rollout

Consider implementing Ethereum-wide bunker mode condition - triggering bunker mode based on observed network slashing share (suggested level: 1.7%-1.8%)

So looks like we will acknowledge the issue, and probably have to lower the severity of the finding, what do you think?

MixBytes: Based on client's feedback, we agree to downgrade this finding from Critical to High severity. While the potential for front-running bunker mode activation remains a valid concern, the likelihood is indeed low.

2.3 Medium

M-1	Incorrect Accounting in <code>internalizeExternalBadDebt</code> May Overestimate Protocol Losses		
Severity	Medium	Status	Acknowledged

Description

This issue has been identified within the `internalizeExternalBadDebt` Lido.sol#L768 of the Lido contract.

The function is designed to convert external bad debt into internal accounting by decreasing `externalShares`, which reduces the protocol's `totalPooledEther`. However, due to how the internal and external shares are coupled, the reduction of `externalShares` without the reduction of `totalShares` already decreases the `shareRate` (because it increases `internalShares` without changing the `internalEther`, which leads to the reduction of the `shareRate`). If `externalShares` are reduced, it affects `totalPooledEther` in 2 ways: the internal `shareRate` is being reduced, and the `externalEther` is being reduced, which may overestimate the losses for the protocol.

Here is an example of the script that shows how the overestimation affects the protocol:

```

def main():

    # input parameters
    internal_ether = 10_000_000 * 10**18
    external_shares = 500_000 * 10**18
    internal_shares = 9_500_000 * 10**18
    liability_shares = 110 * 10**18
    total_vault_value = 105 * 10**18

    total_shares = external_shares + internal_shares
    total_pooled_ether = internal_ether \
        + external_shares * internal_ether // internal_shares

    total_vault_shares = total_vault_value \
        * total_shares // total_pooled_ether
    real_bad_debt = liability_shares \
        * total_pooled_ether // total_shares - total_vault_value
    print("The real bad debt: ", real_bad_debt / 10**18)

    bad_debt_shares = liability_shares - total_vault_shares

    total_pooled_ether_upd = internal_ether \
        + (external_shares - bad_debt_shares) \
        * internal_ether // (internal_shares + bad_debt_shares)

    ether_decrease = total_pooled_ether - total_pooled_ether_upd
    print("Ether decrease:     ", ether_decrease / 10**18)

    vault_unlocked = total_vault_value \
        - (liability_shares - bad_debt_shares) \
        * total_pooled_ether_upd // total_shares
    print("Vault unlocked:     ", vault_unlocked / 10**18)

if __name__ == "__main__":
    main()

```

And the results:

```

The real bad debt: 10.789473
Ether decrease:    11.357328
Vault unlocked:    0.000113

```

This leads to an overestimation of around 5%.

Recommendation

We recommend reviewing the off-chain service responsible for `VaultHub.sol#L609 _maxSharesToWriteOff` to ensure that it implements strict validation to avoid internalizing more

than necessary.

M-2	Incorrect Balance Increase Calculation Due to Validators' Exit		
Severity	Medium	Status	Acknowledged

Description

This issue has been identified within the `_checkAnnualBalancesIncrease` function of the `OracleReportSanityChecker` contract.

The `_preCLBalance` parameter does not account for the withdrawal vault balance, yet it includes new validators. It is possible for N new validators to enter the protocol while simultaneously N old validators exit the protocol. In such a scenario, the `balanceIncrease` `OracleReportSanityChecker.sol#L710` than the actual growth of the total consensus layer balance. As a result, the subsequent check will fail to detect anomalous balance growth. Also, `OracleReportSanityChecker.sol#L704` may pass and return early without any checks for balance increase.

The issue is classified as **Medium** severity because it undermines the effectiveness of an important security control designed to detect abnormal consensus layer balance increases.

Recommendation

We recommend modifying the balance increase calculation to properly account for validators' exit.

Client's Commentary:

Client: <https://github.com/lidofinance/core/issues/1372>

Example?

MixBytes: `_preCLBalance = _pre.clBalance + (_post.clValidators - _pre.clValidators) * DEPOSIT_SIZE`
 If there were some deposits on validators between the 2 reports (let's say we deposited 100 new validators) and some of the withdrawals were finalized between the 2 reports (let's say 20 validators were finalized and their principal balance was sent to EL) the `_preCLBalance` will account only for 100 new validators (since the withdrawal vault balance is not accounted here), but the `_postCLBalance` will increase only by 80 validators (100 - 20)

Client: There is no simple way to track validator exits in accounting. If we simply add the balance of the withdraw vault, that opens a DoS vector on the oracle report by allowing anyone to inflate the withdraw vault balance from the outside. It's not expensive to execute, but it's available to anyone.

It looks like the current approach is optimal from the sanity-checks perspective. We can update the docs and state that we limit growth of the aggregate balance (excluding deposits) over a period to prevent the misunderstanding in the future.

MixBytes: We would also recommend emitting an event inside the `if (_preCLBalance >= _postCLBalance) return;` branch of the `_checkAnnualBalancesIncrease` function to track such cases in the off-chain logic.

M-3	Vault Activation Possible During Bunker Mode, Enabling Bad Debt Attacks		
Severity	Medium	Status	Fixed in 4af82f0d

Description

Even while [Lido v2](#) is in bunker mode, it is still possible to activate vaults and make deposits into them. While this does not directly affect the share rate, it opens up the possibility for an attack scenario where a malicious Node Operator intentionally increases bad debt through the malicious operation as a validator, which causes slashing penalties. This can worsen the protocol's state during bunker mode and potentially prolong the period in which Lido v2 remains in this critical state as the bad debt value increase is uncapped. The root cause is that the [VaultHub](#) contract's pause mechanism is independent from the main protocol's bunker mode, so vault operations are not automatically restricted when the protocol is in bunker mode.

The issue is classified as **Medium** severity because it allows a malicious Node Operator worsen the bunker mode state and allows staking vaults to perform normally, while the Lido v2 protocol experiences the restrictions related to the bunker mode.

Recommendation

We recommend adding explicit checks for bunker mode status in the [VaultHub](#) contract, especially in functions that allow vault activation or deposits. The pause logic of [VaultHub](#) should be synchronized with the main protocol's bunker mode to prevent vault operations that could worsen protocol health during critical periods.

Client's Commentary:

Client: Need example

Mixbytes: The risk of increasing the bad debt is very low since it is not that easy to cause slashing in the CS that would cause bad debt, but there is an invariant that there cannot be new deposits to validators during the bunker mode (this invariant exists in Lido v2) which is better to hold, since bunker mode means some abnormality in CS and it is better to resolve it before setting up new validators.

M-4	Staking Can Be Resumed While Protocol Is Stopped, Breaking Core Invariant		
Severity	Medium	Status	Fixed in 4af82f0d

Description

The `resumeStaking` function in `Lido` allows resuming staking (accepting new deposits) even when the protocol is globally stopped (i.e., after calling `stop`). This breaks a core protocol invariant, explicitly stated in the code, that protocol pause must always imply staking pause. The `_submit` function, which handles deposits, relies on this invariant and does not check the protocol's stopped state, only the staking pause flag. As a result, an authorized actor with the `STAKING_CONTROL_ROLE` can call `resumeStaking` after `stop`, enabling deposits and minting of `stETH` when the protocol is not ready to process them.

The issue is classified as **Medium** severity because it exposes the protocol to operational mistakes and breaks the intended safety model.

Recommendation

We recommend adding an explicit check in `resumeStaking` to ensure the protocol is not globally stopped (e.g., `require(!isStopped(), "PROTOCOL_STOPPED")`). This will enforce the invariant and prevent staking from being enabled while the protocol is paused.

Client's Commentary:

<https://github.com/lidofinance/core/issues/1374>

M-5	smoothenTokenRebase Function doesn't Account for the badDebtToInternalize variable		
Severity	Medium	Status	Acknowledged

Description

There is a call to `oracleReportSanityChecker.smoothenTokenRebase` in the `_simulateOracleReport` function of the `Accounting` contract. This function smoothens the token rebase while limiting how much ether may be used to update the `totalPooledEther` variable, which is used in the `stETH` share rate calculation.

This function doesn't account for the `_pre.badDebtToInternalize` variable, which in a very rare scenario can cause the smoothing logic to pull fewer `update.elRewards` than available, so as not to increase the exchange price of the share too much in the oracle update. However, the `_pre.badDebtToInternalize` will decrease the share price, and if not all the execution layer rewards are pulled from the vault, it can cause a negative rebase.

The issue is classified as **Medium** severity because it may result in a very rare case of a negative rebase that could be resolved by pulling more of the execution layer rewards that are available in the vault.

Recommendation

We recommend also smoothing the `badDebtToInternalize` variable in the `smoothenTokenRebase` function of the `OracleReportSanityChecker` contract, as missing that variable in the smoothing logic may lead to pulling less ether from the execution layer rewards vault than is possible, which may lead to a negative rebase of `stETH`.

Client's Commentary:

Client: <https://github.com/lidofinance/core/issues/1375>

Actually, we have a question about this. While it's true that `_pre.badDebtToInternalize` can increase `update.postInternalShares`, as it is used as a denominator, it will only decrease the share rate, so it won't lead to a larger token rebase. Or did we misunderstand the issue somehow?

MixBytes: We analyzed a highly unlikely yet theoretically possible scenario where `elRewards` collection would be insufficient due to `OracleSanityChecker` constraints, leading to inadequate coverage of the total bad debt (despite potential sufficiency if all rewards were collected), thus forcing a negative rebase that could have been prevented.

Client: Looks like it's better to divide this issue in two, because of different origin and structure:

`badDebtToInternalize`-related is definitely correct and can be fixed, while accounting `sharesToMintAsFees` is more complicated (because its value depends on the smoothing result) and better be done by switching the semantical context from limiting the rebase to limiting the gross APR(before fee deduction)

MixBytes: The issue description related to the `sharesToMintAsFees` was moved to the Low #13 finding.

Client: Supporting increased internal shares would require rewriting the limit logic and complicating the code, but the feature is unlikely to be used. The smoothing mechanism (0.075%/day ≈ 27% APR) almost never triggers, and bad debt internalization is a manual, DAO-governed process that should never occur under normal conditions. Adding complexity for this edge case provides negligible benefit.

2.4 Low

L-1	Max External Shares Ratio Update Can Be Front-Run		
Severity	Low	Status	Acknowledged

Description

This issue has been identified within the `setMaxExternalRatioBP` function of the [Lido](#) contract. The function updates the maximum allowed ratio of external shares, expressed in basis points. However, because this function is publicly callable by an authorized party and immediately updates a critical protocol parameter, it introduces the potential for front-running. A malicious vault owner could monitor the mempool and quickly increase their share of external stake just before the new, more restrictive ratio is enforced, effectively bypassing the intended limitation.

Recommendation

We recommend executing this function through private mempools to prevent front-running.

Client's Commentary:

Acknowledged

L-2	Lack of Minting Cap May Lead to Oversupply via <code>mintShares</code>		
Severity	Low	Status	Acknowledged

Description

This issue has been identified within the `mintShares` function of the `Lido` contract. The function is intended to be called only by the accounting module to mint stETH shares, typically representing protocol fees. However, there is currently no upper bound or sanity check on the amount of shares that can be minted. If a bug or misconfiguration occurs in the accounting module, it may result in an excessive minting of shares, which could dilute user balances or cause economic inconsistencies.

Recommendation

We recommend adding a sanity check to ensure that the number of minted shares does not exceed a reasonable fraction of the total supply (e.g., 1%). This would provide a safeguard against unintended oversupply due to bugs in the accounting module.

Client's Commentary:

Acknowledged

L-3	Incorrect Comment Regarding Validator Count Comparison		
Severity	Low	Status	Fixed in 4af82f0d

Description

This issue has been identified within the `_getInternalEther` function of the `Lido` contract. The inline comment above the assertion incorrectly states that "clValidators can never be less than deposited ones." However, the assertion itself correctly checks that `depositedValidators >= clValidators`, which reflects the actual expected behavior: **deposited validators may exceed the number of Consensus Layer (CL) validators** due to validators being in a transient state (submitted to the Deposit contract but not yet activated on the CL).

Recommendation

We recommend updating the comment to correctly reflect the validator relationship. Suggested revision:

```
// clValidators can never exceed depositedValidators.
```

Client's Commentary:

<https://github.com/lidofinance/core/issues/1364>

L-4	Inconsistent Accessor Usage for Packed Storage Slot		
Severity	Low	Status	Fixed in 4af82f0d

Description

Within the `_getLidoLocator()` function of the `Lido` contract, the contract retrieves the address stored in a packed storage slot using `getStorageAddress()`. However, the paired setter method `_setLidoLocator()` uses `setLowUint160`, suggesting that the lower 160 bits of the slot should be interpreted as an address.

This inconsistency may introduce confusion for maintainers and creates room for bugs in future upgrades if assumptions about the storage format diverge.

Recommendation

To ensure consistency and clarity, we recommend replacing:

```
LOCATOR_AND_MAX_EXTERNAL_RATIO_POSITION.getStorageAddress()
```

with:

```
LOCATOR_AND_MAX_EXTERNAL_RATIO_POSITION.getLowUint160()
```

This will align the getter and setter semantics, reduce confusion, and make the storage layout more predictable.

Client's Commentary:

Issue <https://github.com/lidofinance/core/issues/1363>

L-5	Missing Upper Bound Check on Total Shares During Mint		
Severity	Low	Status	Fixed in ac68c5c6

Description

This issue has been identified within the `_mintShares` function of the `stETH` contract. The function calculates the new total number of shares and writes it to a storage slot using `setLowUint128`, assuming the value always fits within 128 bits. However, there is no explicit upper bound check on `newTotalShares`. If the protocol ever grows beyond $2^{128} - 1$ shares, this could result in unexpected behavior due to silent truncation.

Recommendation

We recommend adding an explicit check to ensure `newTotalShares` fits within `uint128`.

Client's Commentary:

Correct, added to issue <https://github.com/lidofinance/core/issues/1329#issuecomment-3180224379>
fixed in ac68c5c69928edc1f2d1ea79f41dab60bcb82f5c

L-6	Vault Data Can Be Repeatedly Updated Without Restriction		
Severity	Low	Status	Fixed in 4af82f0d

Description

This issue has been identified within the `updateVaultData` function of the `LazyOracle` contract. The function currently allows the same vault data to be submitted multiple times without restriction. Although the internal accounting appears to handle repeated submissions correctly and avoids inconsistency, this behavior could potentially lead to unnecessary state changes and gas inefficiencies, and may cause confusion in external integrations.

Recommendation

We recommend adding a replay protection mechanism, such as marking vault reports as processed per report timestamp, to ensure that each unique vault report can only be submitted once per reporting period.

Client's Commentary:

Correct. See <https://github.com/lidofinance/core/issues/1200>

L-7	Sanity Check Ignores Time Since Last Report		
Severity	Low	Status	Acknowledged

Description

This issue has been identified within the `_processTotalValue` function of the `LazyOracle` contract.

The function calculates `maxSaneTotalValue` as a fixed percentage (determined by `maxRewardRatioBP`) above the previously reported total value, adjusted by `inOutDelta`. However, this check does not consider the time elapsed since the last report. This can result in overly strict or overly permissive bounds depending on how much time has passed, leading to unnecessary quarantining or undetected anomalies in rapidly accruing rewards.

Recommendation

We recommend modifying the sanity check to scale `maxRewardRatioBP` proportionally to the number of days since the last report. This ensures the acceptable increase in total value reflects the actual duration between reports and aligns with expected reward accrual over time.

Client's Commentary:

See <https://github.com/lidofinance/core/issues/1284#issuecomment-3151599003>

L-8	Unsafe Downcast of Share Limit in Group Registration		
Severity	Low	Status	Fixed in 25c2be32

Description

This issue has been identified within the `registerGroup` function of the `OperatorGrid` contract. When setting the `shareLimit` field in a `Group` struct, the value is directly downcasted from `uint256` to `uint96` without any safety checks. If the passed `_shareLimit` exceeds the maximum value of `uint96`, the cast will silently truncate the value, potentially resulting in a `shareLimit` of zero. This could unintentionally disable the group's ability to receive shares or lead to inconsistent accounting.

Recommendation

We recommend explicitly validating the `_shareLimit` against the `uint96` maximum value before casting, or using `SafeCast.toInt96()` from OpenZeppelin to ensure safe downcasting with proper reverts.

L-9	Function Visibility Can Be Changed		
Severity	Low	Status	Fixed in 4af82f0d

Description

The `simulateOracleReport` function of the `Accounting` contract is declared as `public view`, but it can be safely exposed externally.

Recommendation

Change the visibility of `simulateOracleReport` from `public` to `external` to:

- Reduce unnecessary gas overhead when called externally
- Clarify its intended interface usage for off-chain tooling or integrations

Client's Commentary:

<https://github.com/lidofinance/core/issues/1362>

L-10	Arithmetic Style Improvement for Readability and Accuracy Assurance		
Severity	Low	Status	Acknowledged

Description

The formula used to compute `update.postInternalEther` in the `_simulateOracleReport()` function in the `Accounting` contract is technically correct, but the expression is not grouped optimally. The current logic:

```
update.postInternalEther =
    _pre.totalPooledEther - _pre.externalEther
    + _report.clBalance + update.withdrawals - update.principalClBalance
    + update.elRewards
    - update.etherToFinalizeWQ;
```

is functional, but we recommend grouping parts of the formula in more secure way.

Recommendation

Refactor the expression in the following way:

```
update.postInternalEther =
    _pre.totalPooledEther - _pre.externalEther
    + update.elRewards
    + _report.clBalance + update.withdrawals - update.principalClBalance
    - update.etherToFinalizeWQ;
```

Client's Commentary:

Acknowledged

L-11	Ignoring return value from <code>withdrawRewards</code>		
Severity	Low	Status	Acknowledged

Description

In function `collectRewardsAndProcessWithdrawals`, the `withdrawRewards` function on `ElRewardsVault` is called, but its return value is ignored. Instead, the requested amount is always used to update `postBufferedEther`. This is not best practice, because the actual returned value from `withdrawRewards` should be used for accurate accounting. However, in the current implementation, this has no practical impact. The protocol performs a sanity check before accepting the oracle report, making sure that the requested withdrawal amount does not exceed the actual balance of the `ElRewardsVault`. As a result, it is not possible to inflate the internal accounting of ETH in Lido through this mechanism.

Recommendation

We recommend using the actual returned value from `withdrawRewards` when updating the internal buffer, to improve code clarity and robustness.

Client's Commentary:

Acknowledged

L-12	Misleading Comment in the <code>OperatorGrid.changeTier()</code> Function		
Severity	Low	Status	Fixed in 4af82f0d

Description

This issue has been identified within the `changeTier` function of the `OperatorGrid` contract. The function reverts if the vault has not yet been connected to `VaultHub`. The function checks whether the vault is connected using `vaultHub.isVaultConnected(_vault)` and then `OperatorGrid.sol#L454 vaultHub.updateConnection()` regardless of the connection status. This will cause a revert if a vault is not yet connected to `VaultHub`, which is intentional. The issue is classified as **Low** severity because there is a misleading comment before the call to `vaultHub.updateConnection`, which states that this call should be skipped if the vault is not connected.

Recommendation

We recommend modifying the comment to accurately reflect the implemented logic and to include a note about the potential revert if the vault is not connected.

Client's Commentary:

<https://github.com/lidofinance/core/issues/1361>

L-13	Missing <code>sharesToMintAsFees</code> in Smoothing Calculation Limits the Pulled EL Rewards		
Severity	Low	Status	Acknowledged

Description

The oracle report smoothing mechanism in `oracleReportSanityChecker.smoothTokenRebase()` calculates rebase limits and determines how much ETH to transfer from the EL rewards vault without considering the impact of `sharesToMintAsFees` on the final share rate. Since the actual post-rebase share rate will be affected by the minting of fee shares, there will be less ETH pulled from the EL rewards vault than could be safely utilized. This results in additional rewards remaining in the vault that could have been pulled without violating smoothing bounds. The issue is classified as **Low** severity because the impact is insignificant as it does not lead to a negative rebase of stETH, only a marginally lower positive rebase than optimal.

Recommendation

We recommend modifying the smoothing algorithm to incorporate the `sharesToMintAsFees` impact when calculating rebase bounds. This would allow the protocol to pull the maximum safe amount of rewards from the EL rewards vault while still maintaining the intended smoothing limits on the final share rate.

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>