MixBytes()

Lido EasyTrack stVaults Security Audit Report

DECEMBER 11, 2025

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 Lido Easy Track VaultHub and OperatorGrid integration introduces EVMScript factories that enable governance motions for vault management operations through the Easy Track system. These factories provide frictionless governance capabilities for routine vault administration tasks including share limits management, fee adjustments, validator exits, and tier configurations while maintaining the security guarantees of the existing Easy Track framework.

The codebase was audited over 3 days by 4 auditors through a combination of manual review and automated tooling.

Our team performed comprehensive analysis of the smart contract implementations, focusing on access controls, parameter validation, EVMScript generation correctness, and integration security with external VaultHub and OperatorGrid systems.

During the audit, the following attack vectors were examined:

1. **Access Control Verification**: Only trusted callers can invoke critical functions such as createEVMScript, setValidatorExitFeeLimit, and withdrawETH across all factory contracts.
2. **VaultHub Adapter Security**: Sufficient additional validation checks for vault connection parameters in the VaultHubAdapter contract, including proper handling of vault disconnection scenarios after motion creation.
3. **EVMScript Generation Integrity**: Calldata for generated EVMScripts is constructed correctly with appropriate function selectors and parameters for all factory implementations.
4. **Motion Fault Tolerance**: Motion execution resilience through try-catch mechanisms in VaultHub interactions, ensuring partial failures don't compromise the entire motion process.

The scope excluded the core VaultHub and OperatorGrid contracts, focusing exclusively on the EVMScript factory implementations and the VaultHubAdapter contract.

Key observations and recommendations:

- **Specification Discrepancies Identified**: Several minor inconsistencies were found in the technical specification document (UPD: All the issues listed below were resolved):
  - Incorrect parameter descriptions for VaultHubAdapter.socializeBadDebt() where zero addresses are documented as acceptable but would cause motion creation failures;

  - Data type mismatches in `OperatorGrid` documentation (`Group.tierIds[]` documented as `uint128[]` but implemented as `uint256[]`);
  - Typographical errors in methods names (`updateShareLimits` and `updateVaultsFees` in specification vs `updateShareLimit` and `updateVaultFees` in the actual implementation).

- **Robust Parameter Validation**: All factory contracts implement comprehensive input validation including array length matching checks, zero address validation, and logic constraints that prevent invalid motion creation.

- **Consistent Architecture**: All factory implementations follow the established Easy Track patterns with proper `TrustedCaller` inheritance and `IEVMScriptFactory` interface compliance, ensuring seamless integration with the existing governance framework.

No critical vulnerabilities were identified during the audit. The code demonstrates solid engineering practices with appropriate input validation, access controls, and error handling mechanisms.

EasyTrack factory deployments verification included validating that all deployed bytecode corresponds to the audited commit. Parameter configuration review covered the tiered limit structure across deployment phases: initial caps of 50,000 stETH for group registration and share limit adjustments, scaling to 1,000,000 stETH in subsequent phases, alongside tier management capabilities with defined reserve ratios and fee boundaries. We also confirmed appropriate authorization paths through the EVMScriptExecutor to target contracts.

1.3 Project Overview

Summary

Title	Description
Client Name	Lido
Project Name	EasyTrack stVaults
Type	Solidity
Platform	EVM
Timeline	23.07.2025 - 26.11.2025

Scope of Audit

File	Link
contracts/EVMScriptFactories/vaultFactories/AlterTiersInOperatorGrid.sol	AlterTiersInOperatorGrid.sol
contracts/EVMScriptFactories/vaultFactories/DecreaseShareLimitsInVaultHub.sol	DecreaseShareLimitsInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/DecreaseVaultsFeesInVaultHub.sol	DecreaseVaultsFeesInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/ForceValidatorExitsInVaultHub.sol	ForceValidatorExitsInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/RegisterGroupsInOperatorGrid.sol	RegisterGroupsInOperatorGrid.sol
contracts/EVMScriptFactories/vaultFactories/RegisterTiersInOperatorGrid.sol	RegisterTiersInOperatorGrid.sol
contracts/EVMScriptFactories/vaultFactories/SetVaultRedemptionsInVaultHub.sol	SetVaultRedemptionsInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/SocializeBadDebtInVaultHub.sol	SocializeBadDebtInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/UpdateGroupsShareLimitInOperatorGrid.sol	UpdateGroupsShareLimitInOperatorGrid.sol
contracts/EVMScriptFactories/vaultFactories/VaultHubAdapter.sol	VaultHubAdapter.sol
contracts/EVMScriptFactories/vaultFactories/UpdateVaultsFeesInOperatorGrid.sol	UpdateVaultsFeesInOperatorGrid.sol

File	Link
contracts/EVMScriptFactories/vaultFactories/SetJailStatusInOperatorGrid.sol	SetJailStatusInOperatorGrid.sol
contracts/EVMScriptFactories/vaultFactories/SetLiabilitySharesTargetInVaultHub.sol	SetLiabilitySharesTargetInVaultHub.sol
contracts/EVMScriptFactories/vaultFactories/VaultsAdapter.sol	VaultsAdapter.sol

Versions Log

Date	Commit Hash	Note
23.07.2025	577792736e25502e3c1f211bd6adcc0a7ef3aaf9	Initial Commit
04.08.2025	df9c9d09c49bc2df84edafa9a52760d3997374a4	Commit for Re-audit
10.10.2025	7055c7b87b9940a0d6a483aeb79b1b430e33bc4f	Commit for Re-audit
06.11.2025	0dc9ba2befba405a8da37dc892a21dc1ac9498c6	Commit with Updates
19.11.2025	cad11be3e59309c0c992dc0b8ecd69043e9431de	Commit with Updates
21.11.2025	8612b190eef4314b1546483cde8fee5dedc2e8ab	Commit with Updates
25.11.2025	47698cf312efaf82e158193ca9b089843f598ee7	Commit with Updates
26.11.2025	c64468ca5126237c33e17b71d9307a6aea0ee5cc	Commit with Updates

Mainnet Deployments

RegisterGroupsInOperatorGrid.sol (0x194A46DA1947E98c9D79af13E06Cfbee0D8610cC) is deployed for Phase 1 transition;
RegisterGroupsInOperatorGrid.sol (0xE73842AEbEC99Dacf2aAEec61409fD01A033f478) is deployed for Phase 2 and 3 transition;
UpdateGroupsShareLimitInOperatorGrid.sol (0x8Bdc726a3147D8187820391D7c6F9F942606aEe6) is deployed for Phase 1 transition;
UpdateGroupsShareLimitInOperatorGrid.sol (0xf23559De8ab37fF7a154384B0822dA867Cfa7Eac) is deployed for Phase 2 and 3 transition;
AlterTiersInOperatorGrid.sol (0xa29173C7BCf39dA48D5E404146A652d7464aee14) is deployed for Phase 1 transition;
AlterTiersInOperatorGrid.sol (0x73f80240ad9363d5d3C5C3626953C351cA36Bfe9) is deployed for Phase 2 and 3 transition.

File	Address
RegisterGroupsInOperatorGrid.sol	0x194A46DA1947E98c9D79af13E06Cfbee0D8610cC
RegisterGroupsInOperatorGrid.sol	0xE73842AEbEC99Dacf2aAEec61409fD01A033f478

File	Address
UpdateGroupsShareLimitInOperatorGrid.sol	0x8Bdc726a3147D8187820391D7c6F9F942606aEe6
UpdateGroupsShareLimitInOperatorGrid.sol	0xf23559De8ab37fF7a154384B0822dA867Cfa7Eac
RegisterTiersInOperatorGrid.sol	0x5292A1284e4695B95C0840CF8ea25A818751C17F
AlterTiersInOperatorGrid.sol	0xa29173C7BCf39dA48D5E404146A652d7464aee14
AlterTiersInOperatorGrid.sol	0x73f80240ad9363d5d3C5C3626953C351cA36Bfe9
SetJailStatusInOperatorGrid.sol	0x93F1DEE4473Ee9F42c8257C201e33a6Da30E5d67
UpdateVaultsFeesInOperatorGrid.sol	0x5C3bDFa3E7f312d8cf72F56F2b797b026f6B471c
ForceValidatorExitsInVaultHub.sol	0x6C968cD89CA358fbAf57B18e77a8973Fa869a6aA
SocializeBadDebtInVaultHub.sol	0x1dF50522A1D868C12bF71747Bb6F24A18Fe6d32C
SetLiabilitySharesTargetInVaultHub.sol	0x4E5Cc771c7b77f1417fa6BA9262d83C6CCc1e969
VaultsAdapter.sol	0xe2DE6d2DefF15588a71849c0429101F8ca9FB14D

1.4 Security Assessment Methodology

Project Flow

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

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

1.5 Risk Classification

Severity Level Matrix

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

Impact

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

Likelihood

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

Action Required

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

Finding Status

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

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	4

Findings Statuses

ID	Finding	Severity	Status
L-1	Unbounded Motion Size May Cause Gas Exhaustion	Low	Acknowledged
L-2	No check for duplicate nodeOperator in RegisterGroupsInOperatorGrid	Low	Fixed
L-3	Missing Upper-Bound Limits in a Few Factories	Low	Acknowledged
L-4	Misleading Comment in ForceValidatorExitsInVaultHub	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Unbounded Motion Size May Cause Gas Exhaustion		
Severity	Low	**Status**	Acknowledged

Description

There is no limit on the number of items that can be included in a motion. Consequently, a motion can be created successfully but may exceed the block gas limit when executed, leading to a failed transaction. This risk is particularly acute in factory contracts such as DecreaseVaultsFeesInVaultHub, SocializeBadDebtInVaultHub, and similar cases where multiple vault addresses are passed as parameters.

The issue is classified as **Low** severity because, although it can block the execution of overly large motions, it does not compromise funds or system integrity and can be mitigated by operational best practices.

Recommendation

We recommend introducing a reasonable upper bound on the number of items permitted in a motion.

Client's Commentary:

We plan to mitigate this risk off-chain through operational best practices by ensuring that the number of items in any motion remains within safe and executable bounds.

Given that the issue does not affect protocol security or fund safety, and is already manageable through process-level safeguards, we do not plan to introduce an on-chain hard limit at this time.

L-2	No check for duplicate nodeOperator in RegisterGroupsInOperatorGrid		
Severity	Low	**Status**	Fixed in df9c9d09

Description

This issue has been identified within the _validateInputData function of the RegisterGroupsInOperatorGrid contract.

The function does not verify that the nodeOperators array contains unique, non-repeating addresses. Consequently, a motion can be created successfully even when duplicate operators are provided, but the motion will revert when executed.

The issue is classified as **Low** severity because it causes a transaction failure without risking loss of funds or compromising system integrity.

Recommendation

We recommend enforcing ascending order for the nodeOperators array elements and explicitly checking for duplicates during validation.

Client's Commentary:

Fixed - df9c9d09c49bc2df84edafa9a52760d3997374a4

L-3	Missing Upper-Bound Limits in a Few Factories		
Severity	Low	**Status**	Acknowledged

Description

Several factory contracts lack checks that enforce an upper limit for numeric arguments. As a result, some motions may not be executed successfully and this can be foreseen in advance. Such factories and parameters:

1. `SocializeBadDebtInVaultHub` – `_maxSharesToSocialize`
2. `SetVaultRedemptionsInVaultHub` – `_redemptionsValues`
3. `DecreaseShareLimitsInVaultHub` – `_shareLimits`

Additional nuance for `DecreaseShareLimitsInVaultHub`:
`_shareLimit` can be lower than the current vault share limit yet still exceed `maxSaneShareLimit` enforced by `VaultHub._requireSaneShareLimit()`. Because `maxSaneShareLimit` is calculated from `LIDO.getTotalShares()`, which may decline over time, an otherwise valid decrease may now violate this dynamic threshold and cause the entire motion to revert.
The issue is classified as **Low** severity because it leads to a predictable failure without threatening funds or overall system integrity.

Recommendation

We recommend defining a max sane constant for each affected parameter in the constructor of every factory contract, and adding explicit upper-bound checks for these constants inside `_validateInputData()`.

Client's Commentary:

Comments regarding why explicit upper bounds are not enforced for the listed parameters:

- *_maxSharesToSocialize (SocializeBadDebtInVaultHub):*
  This parameter already serves as an upper bound within the logic of VaultHub.socializeBadDebt, so defining a fixed maximum (e.g., the total ETH supply) would not be meaningful in this context. And establishing a hardcoded cap that is appropriate across all vault sizes and scenarios is not practically feasible.
- *_redemptionsValues (SetVaultRedemptionsInVaultHub):*
  Similarly, this parameter functions as a cap in the logic of VaultHub.setVaultRedemptions.
- *_shareLimits (DecreaseShareLimitsInVaultHub):*
  During motion execution, the VaultHubAdapter.updateShareLimit function enforces that the new shareLimit is strictly lower than the current one — which effectively acts as an upper bound. Moreover, if this condition fails for any particular vault, that vault is simply skipped, and the motion continues to execute successfully for the rest.
- *Regarding shareLimit potentially exceeding maxSaneShareLimit in VaultHub._requireSaneShareLimit():*
  While this is theoretically possible, it is an extremely rare edge case. We intentionally avoid duplicating the logic for calculating maxSaneShareLimit inside the factory contract to minimize code duplication and maintain clarity. We accept the minor operational risk that such a motion may revert and need to be resubmitted.

Overall, we believe the current design strikes a reasonable balance between safety and flexibility. We are comfortable handling edge cases operationally as they arise.

L-4	Misleading Comment in ForceValidatorExitsInVaultHub		
Severity	Low	**Status**	Fixed in df9c9d09

Description

There is a misleading comment in the createEVMScript function of the ForceValidatorExitsInVaultHub contract. It claims that the _creator will receive a refund. However, the actual implementation directs the refund to address(this), which corresponds to the VaultHubAdapter contract.

The issue is classified as **Low** severity because it could lead to incorrect assumptions about fund flows.

Recommendation

We recommend updating the comment next to the ForceValidatorExitsInVaultHub.createEVMScript function to accurately reflect that the refund is sent to address(this), and not to _creator.

Client's Commentary:

Fixed - df9c9d09c49bc2df84edafa9a52760d3997374a4

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