MixBytes()

Lido Easy Track Security Audit Report

SEPTEMBER 16, 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 contracts within the scope include EasyTrack EVM factories for requesting validator exits from the Curated and sDVT modules. These exit requests are expected to be performed by the Node Operators or the sDVT Committee.

The audit was completed in one day by 3 auditors.

During the audit the following attack vectors were thoroughly checked:

1. Empty or Invalid Exit Request Arrays. Submitting empty arrays or arrays with invalid data structures represents a critical attack vector that could bypass validation or cause unexpected behavior. The validateExitRequests function in SubmitExitRequestHashesUtils contains comprehensive checks:
- Ensures the exit requests array is not empty (reverts with "EMPTY_REQUESTS_LIST")
- Validates that the array length does not exceed MAX_REQUESTS_PER_MOTION (200 items) (reverts with "MAX_REQUESTS_PER_MOTION_EXCEEDED")
- Checks for duplicate exit requests by comparing public key hashes (reverts with "DUPLICATE_EXIT_REQUESTS")

These validations were analyzed to ensure that no malformed payloads—whether empty, oversized, or containing duplicates—could pass through undetected, effectively neutralizing this attack path. It was also assumed that the configured limit of 200 requests per motion would match the one returned via the ValidatorsExitBus.getMaxValidatorsPerReport() function.

2. Public Key Validation Bypass. Manipulating validator public keys to bypass validation checks may lead to unwanted reverts. The validation logic contains comprehensive checks:
- Ensures public key length is exactly 48 bytes (reverts with "INVALID_PUBKEY_LENGTH")
- Validates that the provided public key matches the registered signing key for the given node operator and key index (reverts with "INVALID_PUBKEY")
- Compares keccak256 hashes of provided and registered keys to prevent manipulation

These validations were carefully reviewed to confirm that no invalid public keys could be submitted, ensuring data integrity and preventing unauthorized exit requests.

3. Cross-Module Request Manipulation. Attempting to submit exit requests for different staking modules within the same transaction represents a sophisticated attack vector designed to bypass module-specific validation. The contract mitigates this by:

- Verifying that the module's stakingModuleAddress matches the NodeOperatorsRegistry address (reverts with "EXECUTOR_NOT_PERMISSIONED_ON_MODULE")
- Ensuring all subsequent requests have the same moduleId as the first request (reverts with "EXECUTOR_NOT_PERMISSIONED_ON_MODULE")

This logic was carefully reviewed to confirm that cross-module request manipulation is impossible, ensuring that all requests in a batch belong to the same validated module.

4. Unauthorized Access to Curated Module Exits. Misimplementation of access control for the Curated module could allow unauthorized users to submit exit requests for node operators they don't control. The `CuratedSubmitExitRequestHashes` contract properly enforces access control by:
- Retrieving the node operator's reward address from NodeOperatorsRegistry
- Requiring that the creator address matches the reward address of the first request's node operator (reverts with "EXECUTOR_NOT_PERMISSIONED_ON_NODE_OPERATOR")
- Ensuring all requests in the batch are for the same node operator (reverts with "EXECUTOR_NOT_PERMISSIONED_ON_NODE_OPERATOR")

We traced the access control chain and evaluated how reward addresses are verified, confirming that access to exit request submission is securely restricted to authorized node operators only.

5. Trusted Caller Bypass for SDVT Module. Bypassing the trusted caller restriction for the SDVT module constitutes a critical attack vector that could allow unauthorized access to exit request submission. The `SDVTSubmitExitRequestHashes` contract inherits from `TrustedCaller` and enforces access control by:
- Using the `onlyTrustedCaller` modifier to restrict access to `createEVMScript` (reverts with "CALLER_IS_FORBIDDEN")
- Setting the creator parameter to address(0) in `validateExitRequests` to skip node operator permission checks
- Ensuring only the designated trusted caller can create EVMScripts

We verified that the `TrustedCaller` inheritance chain is properly implemented and that the trustedCaller address is immutable, eliminating the possibility of unauthorized access.

6. Calldata Decoding Manipulation. Submitting malformed calldata to bypass decoding or cause unexpected behavior represents a fundamental attack vector that could compromise data integrity. The `decodeEVMScriptCallData` function contains comprehensive checks:
- Uses `abi.decode` to safely decode the calldata into `ExitRequestInput[]` array
- Reverts automatically if the calldata format is invalid
- Provides a public interface for external validation of calldata structure

These validations were analyzed to ensure that no malformed calldata could be processed, preventing decoding errors and ensuring data integrity.

The code is generally well-structured and adheres to Solidity best practices, but it can be improved by removing unused imports, eliminating redundant checks, and strengthening validation logic.

1.3 Project Overview

Summary

Title	Description
Client Name	Lido
Project Name	Easy Track
Type	Solidity
Platform	EVM
Timeline	25.06.2025 - 15.07.2025

Scope of Audit

File	Link
contracts/EVMScriptFactories/ CuratedSubmitExitRequestHashes.sol	CuratedSubmitExitRequestHashes.sol
contracts/EVMScriptFactories/ SDVTSubmitExitRequestHashes.sol	SDVTSubmitExitRequestHashes.sol
contracts/libraries/ SubmitExitRequestHashesUtils.sol	SubmitExitRequestHashesUtils.sol

Versions Log

Date	Commit Hash	Note
25.06.2025	941c1dd6231b9bbb6ae42291b0696ecb1bc26a40	Initial Commit
04.07.2025	860a2f6a61288a237ac1912c70385660bb0ecefe	Commit for Re-audit
04.07.2025	b1d96bd02ff3f0ad07602bc3dce0ab3eae4aacfd	Commit with updates

Date	Commit Hash	Note
15.07.2025	a4bbe78934bdec534c4b78871f0bdce57467eab1	Commit for Re-audit

Mainnet Deployments

File	Address	Blockchain
CuratedSubmitExitRequestHashes.sol	0x8aa34dAaF0fC263203A15Bcfa0Ed926D466e59F3	Ethereum Mainnet
SDVTSubmitExitRequestHashes.sol	0xB7668B5485d0f826B86a75b0115e088bB9ee03eE	Ethereum Mainnet

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	3

Findings Statuses

ID	Finding	Severity	Status
L-1	Validator Exit Status and valIndex Not Verified	Low	Fixed
L-2	Redundant Type-Bound Checks for moduleId and nodeOpId	Low	Fixed
L-3	Unused Imports: EVMScriptCreator.sol and IValidatorsExitBusOracle.sol	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	Validator Exit Status and valIndex Not Verified		
Severity	Low	**Status**	Fixed in 860a2f6a

Description

The validateExitRequests function in SubmitExitRequestHashesUtils verifies many aspects of an exit request—such as module and node operator ID ranges, public key length, and duplicate keys. However, it does not check whether the validator has already exited from the beacon chain or is otherwise inactive.

This creates a risk where an exit request could be redundantly submitted for a validator that has already exited, leading to wasted motion execution or unexpected behavior downstream.

There is also a missing check for the valIndex values in the ExitRequestInput[] array. Validator exit requests should be sorted in an ascending order by the valIndex in order to be accepted by the ValidatorsExitBusOracle.

Recommendation

We recommend adding an explicit check in validateExitRequests function to ensure that the validator associated with the exit request is active. We also recommend changing the check for duplicates. It is better to reduce the current check complexity by removing the logic associated with the valPubkey and checking instead that valIndex values in exit requests array are placed in a strictly ascending order – this will ensure that there are no duplicates.

Client's Commentary:

Client: We simplified the check based on the recommendation, but skipped the proposal for validation of the validator's index and CL status. Here's why:

- *A validator's status may change while the objection window is still open. This can cause the enactment transaction to revert on execution. The protocol assumes that validators can be requested again for exit even if the validator has already*

exited.

- *Providing CL proofs is expensive and complex. This significantly reduces the number of validators that can be included in a single batch request.*
- *The Validator Ejector (an off-chain tool hosted by Node Operators) strictly verifies all fields in the event. If any data is invalid, it ignores the event. This mechanism protects us from unchecked data that cannot be easily verified on-chain.*

MixBytes: We ensured the correctness of the `_exitRequests` sorting check implemented in commit `a4bbe78934bdec534c4b78871f0bdce57467eab1`, which takes into account the `moduleId`, `nodeOpId`, and `valIndex` parameters, and confirmed that this check matches the implementation in the `ValidatorsExitBusOracle._processExitRequestsList` function.

L-2	Redundant Type-Bound Checks for moduleId and nodeOpId		
Severity	Low	**Status**	Fixed in 860a2f6a

Description

In the validateExitRequests function, there are two checks:

```
require(_input.moduleId <= type(uint24).max, ERROR_MODULE_ID_OVERFLOW);
require(_input.nodeOpId <= type(uint40).max, ERROR_NODE_OP_ID_OVERFLOW);
```

These checks are redundant because subsequent validations already ensure the correctness of these values:
`require(_input.moduleId == moduleId, ERROR_EXECUTOR_NOT_PERMISSIONED_ON_MODULE);` ensures that `moduleId` is valid and matches the known value. `require(_input.nodeOpId < nodeOperatorsCount, ERROR_NODE_OPERATOR_ID_DOES_NOT_EXIST);` guarantees that `nodeOpId` fits within its intended range.

Recommendation

We recommend removing the two mentioned checks for `moduleId` and `nodeOpId`, since their correctness is already ensured by the logic that follows.

L-3	Unused Imports: EVMScriptCreator.sol and IValidatorsExitBusOracle.sol		
Severity	Low	**Status**	Fixed in 860a2f6a

Description

In SubmitExitRequestHashesUtils.sol, the following imports are declared but never used within the library:

```
import "../libraries/EVMScriptCreator.sol";
import "../interfaces/IValidatorsExitBusOracle.sol";
```

These files are not referenced in any function or type within the current implementation of the library.

Recommendation

We recommend removing the mentioned unused imports.

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