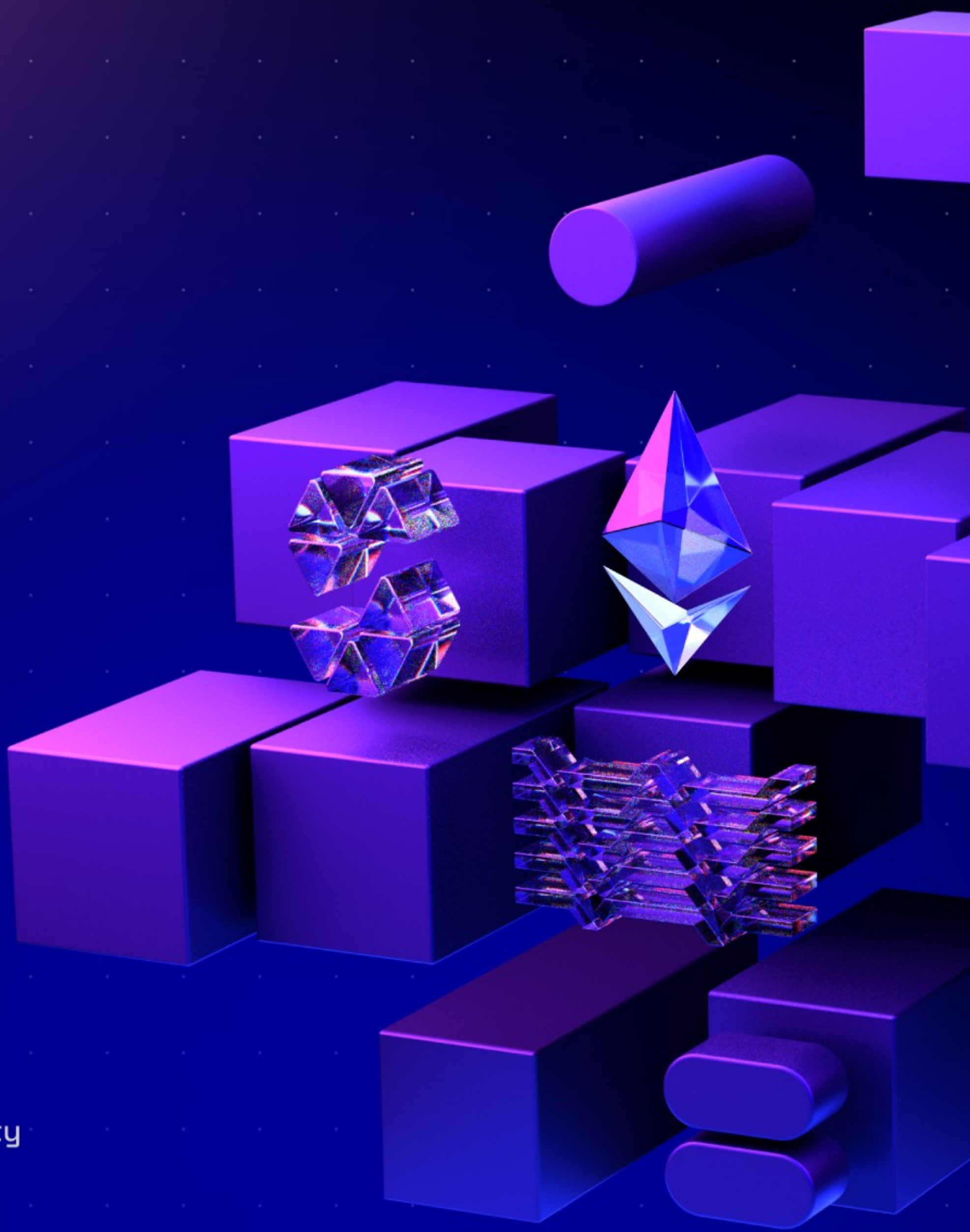
ackee
blockchain security

Lido

Triggerable Withdrawals

16.9.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	14.07.2025
1.1-draft	Fix Review Draft	25.07.2025
2.0	Final Report	16.09.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

https://ackee.xyz

hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**

   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**

   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**

   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**

   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**

   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	Auditor
Naoki Yoshida	Auditor
Martin Veselý	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Lido is a decentralized liquid staking protocol for Ethereum. Triggerable withdrawals is a new feature based on the EIP-7002 standard that allows for the ejection of validators previously deposited through the Lido protocol.

Revision 1.0

Lido engaged Ackee Blockchain Security to perform a security review of Lido Triggerable Withdrawals with a total time donation of 38 engineering days in a period between June 5 and July 14, 2025, with Michal Převrátil as the lead auditor. 10 engineering days were allocated to manually-guided fuzzing using Wake testing framework.

The audit was performed on the commit `628c873`[1] in the core repository and the scope was the following:

- `contracts/0.4.24/nos/NodeOperatorRegistry.sol`
- `contracts/0.8.9/LidoLocator.sol`
- `contracts/0.8.9/StakingRouter.sol`
- `contracts/0.8.9/TriggerableWithdrawalsGateway.sol`
- `contracts/0.8.9/WithdrawalVault.sol`
- `contracts/0.8.9/WithdrawalVaultEIP7002.sol`
- `contracts/0.8.9/oracle/AccountingOracle.sol`
- `contracts/0.8.9/oracle/ValidatorsExitBus.sol`
- `contracts/0.8.9/oracle/ValidatorsExitBusOracle.sol`
- `contracts/0.8.9/lib/ExitLimitUtils.sol`
- `contracts/0.8.25/lib/BeaconTypes.sol`
- `contracts/0.8.25/lib/GIndex.sol`

- `contracts/0.8.25/lib/SSZ.sol`
- `contracts/0.8.25/ValidatorExitDelayVerifier.sol`

The initial review commit `8beee97`[2] was changed 4 days after the beginning of the audit to the commit `628c873`[3].

We began our review by implementing and executing manually-guided differential fuzz tests in Wake testing framework to verify the correctness of the new functionalities and to ensure the changes do not break the existing invariants. Fuzzing was conducted with contracts forked from the mainnet and relevant contracts upgraded to the latest version. This ensured full compatibility with the mainnet deployment. 2 staking modules were used to test the triggerable withdrawals functionality — Node Operators Registry and Community Staking Module v2. More details about the fuzzing process can be found in Report Revision 1.0.

In parallel, we performed in-depth manual review of the code, especially focusing on the triggerable withdrawal functionality, compatibility with EIP-7002 and new code changes since the last audit (commit `1ffbb7e`[4]). During the review, we focused on the following aspects:

- permissionless mechanism of triggering validator exits;
- compatibility with EIP-7002;
- exploring new attack vectors due to triggerable withdrawals functionality;
- permissionless mechanism for reporting of exit delayed validators;
- checking that all state variables are properly updated and do not break any invariants;
- ensuring access controls are not too relaxed or too strict; and
- looking for common issues such as data validation.

The static analysis tools were used to check the code and yielded I4 and I5 findings.

Our review resulted in 11 findings, ranging from Info to Low severity. The most severe findings, L1 and L2, relate to the limited responsiveness of the system upon changing the exit limits configuration parameters and inconsistency in the total number of processed exit requests, respectively.

The code quality could be improved by contracts inheriting from their interfaces (finding W2) which consequentially resulted in two more findings — W1 and W3. The overall code quality is still high, with comprehensive documentation and good architecture.

Ackee Blockchain Security recommends Lido:

- ensure contract upgrade and initialization are atomic to prevent front-running attacks possibly leading to loss of control over the contract;
- always inherit from interfaces in the contracts that implement them; and
- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Lido engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

The review was performed between July 23 and July 25, 2025 on the commit `cfa0c6a`[5]. Except for the fixes, the reviewed commit contained additional minor changes (e.g., variable renaming, `AccountingOracle` version increment, documentation updates) and one major change related to processing historical validator exit delay proofs in the `ValidatorExitDelayVerifier` contract. These changes were reviewed by Ackee Blockchain Security as well.

From the reported 11 findings:

- 9 issues were fixed;
- 1 issue was acknowledged; and
- 1 minor issue was fixed partially.

No new findings were discovered.

Revision 2.0

Lido engaged Ackee Blockchain Security to perform a review of changes made since the previous revision with a total time donation of 0.5 engineering days in a period between September 8 and September 16, 2025, with Michal Převrátil as the lead auditor.

The audit was performed on the commit `acf3188`[6] in the core repository and the scope was all changes made to the `contracts/0.8.25/ValidatorExitDelayVerifier.sol` file since the previous revision.

The changes included:

- removal of unused `rootGIndex` parameter;
- stricter comparison of eligible exit request timestamp compared to reference slot timestamp; and
- improved calculation of historical block root gIndex through third intermediary slot.

The manual review focused on changes made since the previous revision and ensuring no new issues were introduced. Fuzz tests prepared in previous revisions were updated to cover the new changes and run to ensure correctness of the changes. Wake static analysis detectors were used to check the updated code.

No new findings were discovered. The project is of high quality and is ready for deployment.

[1] full commit hash: `628c8736d12478fc9e9a7dcba7dc2e7e6ebb8715`, link to commit

[2] full commit hash: `8beee976ff15472e2ab01fb0247741989ca691ef`, link to commit

[3] full commit hash: `628c8736d12478fc9e9a7dcba7dc2e7e6ebb8715`, link to commit

[4] full commit hash: `1ffbb7e49e112fcac678f59bf63ba57a7e522874`, link to commit

[5] full commit hash: `cfa0c6a3605aabed41d0200d6a7c32d6b71e91b4`, link to commit

[6] full commit hash: `acf3188c79e5616ef7594999f606473214e10f6b`, link to commit

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	0	0	2	4	5	11

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
L1: Inconsistent update of exit limits on config change	Low	1.0	Fixed
L2: Inconsistent calculation of total requests processed	Low	1.0	Fixed
W1: Unimplemented function called	Warning	1.0	Fixed
W2: Missing interface inheritance	Warning	1.0	Acknowledged
W3: Outdated `IConsensusContract` interface	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
W4: `_setExitDeadlineThreshold` underflow	Warning	1.0	Fixed
I1: Code optimizations	Info	1.0	Fixed
I2: Lack of event emission	Info	1.0	Fixed
I3: Lack of context in deprecated function NatSpec	Info	1.0	Fixed
I4: Unused errors	Info	1.0	Partially fixed
I5: Unused using-for directive	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	Auditor
Naoki Yoshida	Auditor
Martin Veselý	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The triggerable withdrawals functionality has been added to the Lido protocol to enable the initiation of validator exits without node operator involvement. Its primary purpose is to allow the Lido DAO to withdraw funds locked in a validator when the node operator refuses to submit a withdrawal request. This functionality became available after EIP-7002 was implemented on mainnet. Withdrawal triggering is permissionless; however, users must provide data whose hash matches one already submitted by Easy Track or trigger exits once the Oracle has provided the report data.

The Staking Router mediates interactions among protocol components such as the Staking Modules, Deposit Security Module, Accounting Oracle, etc. The main task of the Staking Router is to keep the state of the protocol consistent by properly updating state variables and notifying other components about important changes in the state. Most of the functions of the Staking Router are permissioned.

The primary change since the previous audit is the updated reporting mechanism. Late validators no longer exist in the system. The stuck keys and refunding functionality has been removed from the protocol.

Validators that were requested to exit but did not do so within the required timeframe, called exit-delayed validators, can now be reported permissionlessly via EIP-4788.

Trust Model

Lido allows permissionless triggering of validator exits if the validator is included in a report submitted via Easy Track or Oracle. The protocol relies on two trusted components for submitting withdrawal reports:

The flow for triggering validator exit via Easy Track is as follows:

1. The hash of the report is submitted by an address with the `SUBMIT_REPORT_HASH_ROLE` role, which is assigned to Easy Track;
2. Anyone can submit report data with the same hash provided in the first step; and
3. Anyone can trigger the exit of a validator included in the report.

The flow for triggering validator exit via Oracle is as follows:

1. The hash of the report is submitted by the consensus contract;
2. The report data is submitted by an address with the `SUBMIT_DATA_ROLE` role or by a consensus member; and
3. Anyone can trigger the exit of a validator included in the report.

Easy Track is an on-chain component that conducts lightweight voting; a proposal passes if the minimum objections threshold is not reached.

The Trigger Exits Bot is an off-chain component that ensures withdrawal requests are not unnecessarily stalled so that users experience smooth exits.

The Validator Late Prover Bot is an automated tool to detect and report late validators who have failed to exit within the required timeframe after an exit request.

An address with the `ADD_FULL_WITHDRAWAL_REQUEST_ROLE` role can submit withdrawal requests via the Triggerable Withdrawal Gateway.

Staking Router functions require specific roles to be assigned to the caller's address.

Fuzzing

Manually-guided differential fuzz tests were developed during the review to test the correctness and robustness of the system. The fuzz tests employ the fork testing technique to test the system with external contracts exactly as they are deployed on the mainnet. This is crucial to detect any potential integration issues.

The differential fuzz tests keep their own Python state according to the system's specification. Assertions are used to verify the Python state against the on-chain state in contracts.

A minor well-controlled change to the contracts was needed to extend the bit size of variables holding the timestamp to be able to conduct long-running fuzzing campaigns.

The list of all implemented execution flows and invariants is available in Appendix B.

The fuzz test was integrated with a Community Staking Module fuzz test prepared by Ackee Blockchain Security during a parallel audit to ensure compatibility and integration between the two systems.

The full source code of the fuzz tests is available at https://github.com/Ackee-Blockchain/tests-lido-csm-v2.

Additionally, extra fuzz tests were implemented to only focus on the `ExitLimitUtils` library functionality, and deeper testing of Node Operators Registry and Staking Router integration (based on the fuzz test implemented in the previous audit of the core repository by Ackee Blockchain Security).

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

L1: Inconsistent update of exit limits on config change

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	ExitLimitUtils.sol	Type:	Logic error

Description

The `ExitLimitUtils.setExitLimits` function allows updating the exit limits configuration parameters. The current exits limit represented by the `_data.prevExitRequestsLimit` variable possibly changes upon updating the limits.

Listing 1. Excerpt from [ExitLimitUtils.setExitLimits]()

```
if (
    // new maxExitRequestsLimit is smaller than prev remaining limit
    maxExitRequestsLimit < _data.prevExitRequestsLimit ||
    // previously exits were unlimited
    _data.maxExitRequestsLimit == 0
) {
    _data.prevExitRequestsLimit = uint32(maxExitRequestsLimit);
}
```

The current limit is only updated if no previous maximum limit was set (which is correct), or if the new maximum limit is lower than the current limit. However, the current limit is just reset to the new maximum limit, not proportionally decreased.

Additionally, the current limit is not proportionally increased when the new maximum limit increases (compared to the previous maximum limit). This is especially important when the frame duration changes as well.

Exploit scenario

Maximum limit increases

Upon calling the `ExitLimitUtils.setExitLimits` function, the parameters are set as follows:

- previous maximum limit (`_data.maxExitRequestsLimit`): 10
- current limit (`_data.prevExitRequestsLimit`): 0
- new maximum limit (`maxExitRequestLimit`): 100
- restored exits per frame (`exitsPerFrame`): 10

Since neither of the `maxExitRequestsLimit < _data.prevExitRequestsLimit || _data.maxExitRequestsLimit == 0` conditions are met, the current limit is not updated. It takes at least 10 frames for the current limit to reach the maximum value.

Maximum limit decreases

Upon calling the `ExitLimitUtils.setExitLimits` function, the parameters are set as follows:

- previous maximum limit (`_data.maxExitRequestsLimit`): 25
- current limit (`_data.prevExitRequestsLimit`): 20
- new maximum limit (`maxExitRequestLimit`): 10

Since the new maximum limit is lower than the current limit, the current limit is reset to the new maximum limit (i.e. 10). However, the number of consumed exits (given the previous maximum limit) is 5. Therefore, it would be more adequate to set the current limit to `10 - 5 = 5`.

Recommendation

Always update the current limit (`_data.prevExitRequestsLimit`) when changing the configuration parameters, proportionally to the number of exits already consumed (given the previous maximum limit).

```
if (_data.maxExitRequestsLimit == 0) {
    // no limit was set before, set the new limit
    _data.prevExitRequestsLimit = uint32(maxExitRequestsLimit);
} else {
    // update current limit proportionally as `newLimit - exitsUsed`
    // where `exitsUsed` is relative to the previous limit
    uint32 exitsUsed = _data.maxExitRequestsLimit - _data.prevExitRequestsLimit;
    if (exitsUsed >= maxExitRequestsLimit) {
        _data.prevExitRequestsLimit = 0;
    } else {
        _data.prevExitRequestsLimit = uint32(maxExitRequestsLimit - exitsUsed);
    }
}
```

Note that this change reduces system predictability but improves system responsiveness.

Fix 1.1

The `ExitLimitUtils.setExitLimits` function was updated with the following code:

```
if (_data.maxExitRequestsLimit == 0) {
    // no limit was set before, set the new limit
    _data.prevExitRequestsLimit = uint32(maxExitRequestsLimit);
} else {
    uint256 currentLimit = calculateCurrentExitLimit(_data, timestamp);
    // update current limit proportionally as `newLimit - exitsUsed`
    // where `exitsUsed` is relative to the previous limit
    uint32 exitsUsed = _data.maxExitRequestsLimit - uint32(currentLimit);
    if (exitsUsed >= maxExitRequestsLimit) {
        _data.prevExitRequestsLimit = 0;
    } else {
        _data.prevExitRequestsLimit = uint32(maxExitRequestsLimit - exitsUsed);
    }
}
```

The updated code correctly accounts for the case when the current limit (`_data.prevExitRequestsLimit`) should be updated before the parameters configuration is changed because some exit requests were restored since the last update.

[Go back to Findings Summary]

L2: Inconsistent calculation of total requests processed

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	ValidatorsExitBusOracle.sol	Type:	Logging

Description

The `ValidatorsExitBusOracle` contract maintains a variable that tracks the total number of processed triggerable withdrawal requests. There are two distinct paths for processing these requests:

Via Easy Track path:

- Easy Track (with `SUBMIT_REPORT_HASH_ROLE` role) submits a hash of the report containing eligible validators for withdrawal

Listing 2. Excerpt from [ValidatorsExitBus]()

```
function submitExitRequestsHash(bytes32 exitRequestsHash) external
    whenResumed onlyRole(SUBMIT_REPORT_HASH_ROLE) {
```

- Any user submits the report data

Listing 3. Excerpt from [ValidatorsExitBus]()

```
function submitExitRequestsData(ExitRequestsData calldata request) external
    whenResumed {
```

- Any user triggers the exit for validators in the report

Listing 4. Excerpt from [ValidatorsExitBus]()

```
function triggerExits(
```

Via Consensus path:

- The consensus contract submits a hash of the report containing eligible validators for withdrawal

Listing 5. Excerpt from [BaseOracle]()

```
function submitConsensusReport(bytes32 reportHash, uint256 refSlot, uint256
    deadline) external {
```

- A consensus member or address with `SUBMIT_DATA_ROLE` role submits the report data

Listing 6. Excerpt from [ValidatorsExitBusOracle]()

```
function submitReportData(ReportData calldata data, uint256 contractVersion)
    external whenResumed {
```

- Any user triggers the exit for validators in the report

Listing 7. Excerpt from [ValidatorsExitBus]()

```
function triggerExits(
```

The issue is that the tracking variable is only updated during the Consensus path (in the second step) and not during the Easy Track path. This leads to inconsistent accounting of the total number of processed requests.

Exploit scenario

Alice, a protocol monitor, relies on the `TOTAL_REQUESTS_PROCESSED_POSITION` counter to track the protocol's triggerable withdrawal requests via Easy Track or via Consensus.

1. The hash of a report containing 100 validator exit requests is submitted by the Easy Track.

2. The report data is submitted by regular user Bob.
3. The exit is triggered by regular user Bob, successfully processing all 100 requests.
4. The `TOTAL_REQUESTS_PROCESSED_POSITION` counter remains at its previous value, failing to account for the 100 processed requests.
5. Alice checks the counter and sees no change, leading her to miss critical protocol activity.
6. Later, when 50 requests are processed via the Consensus path, the counter only shows 50 total requests instead of 150.
7. This inconsistency prevents Alice from accurately monitoring the protocol's withdrawal activity and taking necessary operational actions.

Recommendation

Update the tracking variable in both paths by adding the counter update in the `submitExitRequestsData` function, which is used by the Easy Track path.

Fix 1.1

The `TOTAL_REQUESTS_PROCESSED_POSITION` storage slot definition was moved to the `ValidatorsExitBus` contract. The counter now updates in both request processing paths.

[Go back to Findings Summary]

W1: Unimplemented function called

Impact:	Warning	Likelihood:	N/A
Target:	StakingRouter.sol, NodeOperatorsRegistry.sol	Type:	Logic error

Description

The `StakingRouter.updateRefundedValidatorsCount` function calls the `NodeOperatorsRegistry.updateRefundedValidatorsCount` function which is not implemented, causing all calls to revert. The `NodeOperatorsRegistry` contract should implement the `IStakingModule` interface, which defines the `updateRefundedValidatorsCount` function, but the implementation is missing.

Listing 8. Excerpt from [StakingRouter]()

```
function updateRefundedValidatorsCount(
    uint256 _stakingModuleId,
    uint256 _nodeOperatorId,
    uint256 _refundedValidatorsCount
) external onlyRole(STAKING_MODULE_MANAGE_ROLE) {
    _getIStakingModuleById(_stakingModuleId).updateRefundedValidatorsCount(
        _nodeOperatorId, _refundedValidatorsCount
    );
}
```

Recommendation

Implement the `updateRefundedValidatorsCount` function in the `NodeOperatorsRegistry` contract either as an empty or reverting function and make the `NodeOperatorsRegistry` contract inherit from the `IStakingModule` interface.

Fix 1.1

The `updateRefundedValidatorsCount` function was removed from the

`IStakingModule` interface and the function with the same name was also removed from the `StakingRouter` contract.

Go back to Findings Summary

W2: Missing interface inheritance

Impact:	Warning	Likelihood:	N/A
Target:	**/*.sol	Type:	Code quality

Description

The codebase contains several contracts that do not inherit from their own interfaces. Namely, the following interfaces are not inherited by their corresponding contracts:

- `IStakingRouter`
- `IWithdrawalVault`
- `ITriggerableWithdrawalsGateway`
- `IConsensusContract`
- `IValidatorsExitBus`
- `IStakingModule`

Not inheriting from the interfaces is a bad practice that prevents easy code navigation, makes it difficult for static analysis tools to analyze external calls, and may lead to issues such as W1 and W3.

Recommendation

Define all needed interfaces in extra files and inherit from them in the contracts that implement them.

Note that even though the project uses multiple Solidity versions, it is still possible to define the needed interfaces in a common directory either without `pragma solidity` or with a loose pragma version. This allows the interface to be used in multiple parts of the codebase compiled with different versions.

Acknowledgment 1.1

The Lido team acknowledged the issue with the following comment:

> *Our current architecture is not conducive to implementing interface inheritance with classes. Doing so would require refactoring many contracts, which would result in changes to the bytecode during compilation. We implemented the necessary interfaces locally in each contract and wrote a small script that compares the interfaces with the source contract signatures.*

Go back to Findings Summary

W3: Outdated `IConsensusContract` interface

Impact:	Warning	Likelihood:	N/A
Target:	BaseOracle.sol, AccountingOracle.sol	Type:	Code quality

Description

The `IConsensusContract` interface defined in the `BaseOracle.sol` file is outdated.

The interface defines the `getFrameConfig` function in the following way:

Listing 9. Excerpt from [BaseOracle]()

```
function getFrameConfig() external view returns (uint256 initialEpoch,
  uint256 epochsPerFrame);
```

However, the only contract that implements the `getFrameConfig` function is the `HashConsensus` contract. However, the function signature is different:

Listing 10. Excerpt from [HashConsensus]()

```
function getFrameConfig() external view returns (
    uint256 initialEpoch,
    uint256 epochsPerFrame,
    uint256 fastLaneLengthSlots
) {
```

Consequently, the `_checkOracleMigration` function of the `AccountingOracle` contract will fail since it uses the outdated interface.

Listing 11. Excerpt from [AccountingOracle._checkOracleMigration]()

```
(uint256 initialEpoch,
    uint256 epochsPerFrame) =
IConsensusContract(consensusContract).getFrameConfig();
```

Recommendation

Update the `IConsensusContract` interface to match the `HashConsensus` contract and add the missing inheritance as recommended in the W2 finding.

Fix 1.1

The `IConsensusContract` interface was renamed to `IHashConsensus` and the `getFrameConfig` function interface was updated to match the `HashConsensus` contract. The function call in `AccountingOracle._checkOracleMigration` was updated as well.

Go back to Findings Summary

W4: `_setExitDeadlineThreshold` underflow

Impact:	Warning	Likelihood:	N/A
Target:	NodeOperatorsRegistry.sol	Type:	Overflow/Underflow

Description

`NodeOperatorsRegistry._setExitDeadlineThreshold` is a privileged function that is responsible for setting the exit delay reporting cut-off timestamp and a threshold. The function first computes the new cut-off timestamp based on the `_threshold` and `_lateReportingWindow` input parameters. The computed value is then required to be greater or equal to the currently set cut-off timestamp and then stored in the contract storage using the `Packed64x4` library.

Listing 12. Excerpt from [NodeOperatorsRegistry._setExitDeadlineThreshold]()

```
uint256 currentCutoffTimestamp = block.timestamp - _threshold -
_lateReportingWindow;
require(exitPenaltyCutoffTimestamp() <= currentCutoffTimestamp,
"INVALID_EXIT_PENALTY_CUTOFF_TIMESTAMP");

Packed64x4.Packed memory stats = Packed64x4.Packed(0);
stats.set(EXIT_DELAY_THRESHOLD_OFFSET, _threshold);
stats.set(EXIT_PENALTY_CUTOFF_TIMESTAMP_OFFSET, currentCutoffTimestamp);
EXIT_DELAY_STATS.setStorageUint256(stats.v);

emit ExitDeadlineThresholdChanged(_threshold, _lateReportingWindow);
```

The `Packed64x4` library reverts if the value being stored is greater than the maximum value of the `uint64` type.

Listing 13. Excerpt from [Packed64x4]()

```
function set(Packed memory _self, uint8 n, uint256 x) internal pure {
```

```
    require(x <= UINT64_MAX, "PACKED_OVERFLOW");
    _self.v = _self.v & ~(UINT64_MAX << (64 * n)) | ((x & UINT64_MAX) << (64 * n));
}
```

However, the `_setExitDeadlineThreshold` function does not check for a possible underflow of the computed value, i.e., the case when `_threshold + _lateReportingWindow` is greater than `block.timestamp`. Even though the execution would often revert given the `Packed64x4` library limitation, it is still possible to pass the input parameters such that the computation underflows and the result fits in the `uint64` type. In such a case, the incorrect cut-off timestamp would be stored in the contract and it would not be possible to set the correct value again because of the `exitPenaltyCutoffTimestamp() <= currentCutoffTimestamp` require condition in `_setExitDeadlineThreshold`.

Recommendation

Check if the `_setExitDeadlineThreshold` function is called with the `_threshold` and `_lateReportingWindow` input parameters such that the computed value underflows. If so, the function should revert.

Fix 1.1

An underflow check was added to the `_setExitDeadlineThreshold` function.

[Go back to Findings Summary]

I1: Code optimizations

Impact:	Info	Likelihood:	N/A
Target:	ExitLimitUtils.sol, ValidatorExitDelayVerifier.sol	Type:	Gas optimization

Description

The project code may be optimized for gas usage in multiple instances.

Listing 14. Excerpt from [ValidatorExitDelayVerifier.verifyValidatorExitDelay]

```
_verifyValidatorExitUnset(beaconBlock.header, validatorWitnesses[i], pubkey, valIndex);
```

The line can be optimized as follows:

```
_verifyValidatorExitUnset(beaconBlock.header, witness, pubkey, valIndex);
```

The following function can be optimized:

Listing 15. Excerpt from [ExitLimitUtils]

```
function updatePrevExitLimit(
    ExitRequestLimitData memory _data,
    uint256 newExitRequestLimit,
    uint256 timestamp
) internal pure returns (ExitRequestLimitData memory) {
    if (_data.maxExitRequestsLimit < newExitRequestLimit) revert LimitExceeded();

    uint256 secondsPassed = timestamp - _data.prevTimestamp;
    uint256 framesPassed = secondsPassed / _data.frameDurationInSec;
    uint32 passedTime = uint32(framesPassed) * _data.frameDurationInSec;

    _data.prevExitRequestsLimit = uint32(newExitRequestLimit);
    _data.prevTimestamp += passedTime;

```

```
    return _data;
}
```

with the following code, which is more gas efficient:

```
    function updatePrevExitLimitNew(
        ExitRequestLimitData memory _data,
        uint256 newExitRequestLimit,
        uint256 timestamp
    ) internal pure returns (ExitRequestLimitData memory) {
        if (_data.maxExitRequestsLimit < newExitRequestLimit) revert
LimitExceeded();

        uint256 passedTime = timestamp - _data.prevTimestamp;
        passedTime -= passedTime % _data.frameDurationInSec;

        _data.prevExitRequestsLimit = uint32(newExitRequestLimit);
        _data.prevTimestamp += uint32(passedTime);

        return _data;
    }
```

Recommendation

Consider the code optimizations.

Fix 1.1

Both code optimizations were implemented.

[Go back to Findings Summary]

I2: Lack of event emission

Impact:	Info	Likelihood:	N/A
Target:	ValidatorsExitBus.sol	Type:	Logging

Description

The `ValidatorsExitBus._setMaxValidatorsPerReport` function changes the maximum number of validators' exits per report, but this state change is not emitted as an event.

Listing 16. Excerpt from [ValidatorsExitBus]()

```
function _setMaxValidatorsPerReport(uint256 maxValidatorsPerReport) internal
{
    if (maxValidatorsPerReport == 0) revert
ZeroArgument("maxValidatorsPerReport");

MAX_VALIDATORS_PER_REPORT_POSITION.setStorageUint256(maxValidatorsPerReport)
;
}
```

Recommendation

Emit an event when the maximum number of validator exits per report is changed via the `ValidatorsExitBus._setMaxValidatorsPerReport` function.

Fix 1.1

A new event `SetMaxValidatorsPerReport` was added to the `ValidatorsExitBus` contract and the `_setMaxValidatorsPerReport` function was updated to emit the event.

[Go back to Findings Summary]()

I3: Lack of context in deprecated function NatSpec

Impact:	Info	Likelihood:	N/A
Target:	NodeOperatorsRegistry.sol	Type:	Code quality

Description

The following code snippet shows a deprecated function with insufficient NatSpec documentation:

Listing 17. Excerpt from [NodeOperatorsRegistry]()

```
/// @param _nodeOperatorId Id of the node operator
/// @param _isTargetLimitActive Flag indicating if the soft target limit is active
/// @param _targetLimit Target limit of the node operator
/// @dev This function is deprecated, use updateTargetValidatorsLimits instead
function updateTargetValidatorsLimits(uint256 _nodeOperatorId, bool _isTargetLimitActive, uint256 _targetLimit) public {
    updateTargetValidatorsLimits(_nodeOperatorId, _isTargetLimitActive ? 1 : 0, _targetLimit);
}
```

The NatSpec comment does not explain the differences between the deprecated function and its replacement. The deprecated function uses a boolean parameter `_isTargetLimitActive` while the new function uses a numeric `_targetLimitMode` parameter that supports more modes (0 = disabled, 1 = soft mode, 2 = boosted mode).

Recommendation

Update the NatSpec to explain the differences between the deprecated function and the new function. For example:

```
/// @dev The function updateTargetValidatorsLimits(uint256, bool, uint256) is
```

```
deprecated
/// @dev Use updateTargetValidatorsLimits(uint256, uint256, uint256) instead.
```

Fix 1.1

The NatSpec documentation was updated to clearly explain the differences between the deprecated function and the new function.

[Go back to Findings Summary]

I4: Unused errors

Impact:	Info	Likelihood:	N/A
Target:	ValidatorExitBusOracle.sol, OracleReportSanityChecker.sol, TriggerableWithdrawalsGateway.sol	Type:	Code quality

Description

The codebase contains multiple unused errors. See Appendix B for the full list.

Recommendation

Remove the unused errors.

Partial solution 1.1

The unused errors in `ValidatorExitBusOracle` and `TriggerableWithdrawalsGateway` were removed. The unused error in `OracleReportSanityChecker` was kept intact as the contract was not affected by the Triggerable Withdrawals upgrade.

Go back to Findings Summary

I5: Unused using-for directive

Impact:	Info	Likelihood:	N/A
Target:	ValidatorsExitBusOracle.sol	Type:	Code quality

Description

The `ValidatorsExitBusOracle` contract contains unused using-for directives. See Appendix B for the full list.

Recommendation

Remove the unused using-for directives.

Fix 1.1

The unused using-for directives were removed.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Lido: Triggerable Withdrawals, 16.9.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Fuzzing

The following table lists all implemented execution flows in the Wake fuzzing framework.

ID	Flow	Added
F1	Adding new consensus members responsible for validator exit request reporting	1.0
F2	Removing consensus members responsible for validator exit request reporting	1.0
F3	Depositing new validator keys from NOR and CSM	1.0
F4	Submitting VEBO exit report data	1.0
F5	Submitting VEB exit report hash	1.0
F6	Submitting VEB exit report data	1.0
F7	Triggering validator exits	1.0
F8	Setting exit request limiting parameters	1.0
F9	Setting maximum exit requests per VEB report	1.0
F10	Verifying validator exit delays	1.0
F11	Verifying historical validator exit delays	1.0
F12	Adding node operators to NOR	1.0
F13	Activating node operators in NOR	1.0
F14	Deactivating node operators in NOR	1.0
F15	Setting node operator name in NOR	1.0

ID	Flow	Added
F16	Setting node operator reward address in NOR	1.0
F17	Setting node operator staking limit in NOR	1.0
F18	Adding node operator signing keys in NOR	1.0

Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Transactions do not revert except where explicitly expected and with the expected data	1.0	Success
IV2	Contracts emit expected events with correct parameters only when expected	1.0	Success
IV3	VEBO's report processing state matches expected values	1.0	Success
IV4	VEBO's report delivery timestamp matches expected values	1.0	Success
IV5	VEB's report delivery timestamp matches expected values	1.0	Success
IV6	`ValidatorsExitBus.unpackExitRequest` function returns expected values for given inputs	1.0	Success
IV7	Exit request limits info matches expected values for `TriggerableWithdrawalsGateway` and `ValidatorsExitBus`	1.0	Success
IV8	`ValidatorsExitBusOracle.getTotalRequestsProcessed` function returns correct number of processed VEBO requests	1.0	Success

ID	Invariant	Added	Status
IV9	`ValidatorsExitBus.getMaxValidatorsPerReport` function returns correct value	1.0	Success
IV10	`isValidatorExitDelayPenaltyApplicable` return value correctly reflects the behavior of `reportValidatorExitDelay` with the same parameters in NOR	1.0	Success
IV11	Node operator info matches expected values in NOR	1.0	Success

Table 5. Wake fuzzing invariants

B.2. Detectors

```
wake detect unused-error

[INFO][HIGH] Unused error [unused-error]
    error AdminCannotBeZero();
    error SenderNotAllowed();
    error UnexpectedRequestsDataLength();
    error ArgumentOutOfBounds();

    event WarnDataIncompleteProcessing(uint256 indexed refSlot, uint256 requestsProcessed, uint256 reques

contracts/0.8.9/oracle/ValidatorsExitBusOracle.sol
```

Figure 1. Unused error

```
wake detect unused-error

[INFO][HIGH] Unused error [unused-error]
    error IncorrectCLBalanceIncrease(uint256 annualBalanceDiff);
    error IncorrectAppearedValidators(uint256 appearedValidatorsLimit);
    error IncorrectNumberOfExitRequestsPerReport(uint256 maxRequestsCount);
    error IncorrectExitedValidators(uint256 exitedValidatorsLimit);
    error IncorrectRequestFinalization(uint256 requestCreationBlock);
    error ActualShareRateIsZero();
    error IncorrectSimulatedShareRate(uint256 simulatedShareRate, uint256 actualShareRate);
contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol
```

Figure 2. Unused error

wake detect unused-error

```
[INFO][HIGH] Unused error [unused-error]
    /**
     * @notice Thrown when exit request has wrong length
     */
    error InvalidRequestsDataLength();

    /**
     * @notice Thrown when a withdrawal fee insufficient
contracts/0.8.9/TriggerableWithdrawalsGateway.sol
```

Figure 3. Unused error

wake detect unused-using-for

```
[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
contract ValidatorsExitBusOracle is BaseOracle, ValidatorsExitBus {
    using UnstructuredStorage for bytes32;
    using SafeCast for uint256;
    using ExitLimitUtilsStorage for bytes32;
    using ExitLimitUtils for ExitRequestLimitData;

    error AdminCannotBeZero();
contracts/0.8.9/oracle/ValidatorsExitBusOracle.sol

[WARNING][LOW] Unused contract in using-for directive [unused-using-for]
    using UnstructuredStorage for bytes32;
    using SafeCast for uint256;
    using ExitLimitUtilsStorage for bytes32;
    using ExitLimitUtils for ExitRequestLimitData;

    error AdminCannotBeZero();
    error SenderNotAllowed();
contracts/0.8.9/oracle/ValidatorsExitBusOracle.sol
```

Figure 4. Unused using-for

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz