OpenZeppelin | security

Lido Dual Governance Audit

LIDO

November 19, 2024

Table of Contents

Summary

Type	DeFi	Total Issues	26 (18 resolved, 2 partially resolved)
Timeline	From 2024-09-16 To 2024-10-25	Critical Severity Issues	0 (0 resolved)
Languages	Solidity	High Severity Issues	0 (0 resolved)
		Medium Severity Issues	1 (1 resolved)
		Low Severity Issues	9 (5 resolved, 1 partially resolved)
		Notes & Additional Information	16 (12 resolved, 1 partially resolved)

Scope

We audited the lidofinance/dual-governance repository at commit 8296824.

In scope were the following files:

```
contracts
├── DualGovernance.sol
├── EmergencyProtectedTimelock.sol
├── Escrow.sol
├── Executor.sol
├── ImmutableDualGovernanceConfigProvider.sol
├── ResealManager.sol
├── TimelockedGovernance.sol
├── committees
│   ├── HashConsensus.sol
│   ├── ProposalsList.sol
│   ├── TiebreakerCoreCommittee.sol
│   └── TiebreakerSubCommittee.sol
├── interfaces
│   ├── IDualGovernance.sol
│   ├── IDualGovernanceConfigProvider.sol
│   ├── IEmergencyProtectedTimelock.sol
│   ├── IEscrow.sol
│   ├── IExternalExecutor.sol
│   ├── IGateSeal.sol
│   ├── IGovernance.sol
│   ├── IOwnable.sol
│   ├── IResealManager.sol
│   ├── ISealable.sol
│   ├── IStETH.sol
│   ├── ITiebreaker.sol
│   ├── ITiebreakerCoreCommittee.sol
│   ├── ITimelock.sol
│   ├── IWithdrawalQueue.sol
│   └── IWstETH.sol
├── libraries
│   ├── AssetsAccounting.sol
│   ├── DualGovernanceConfig.sol
│   ├── DualGovernanceStateMachine.sol
│   ├── EmergencyProtection.sol
│   ├── EnumerableProposals.sol
│   ├── EscrowState.sol
│   ├── ExecutableProposals.sol
│   ├── ExternalCalls.sol
│   ├── Proposers.sol
│   ├── SealableCalls.sol
│   ├── Tiebreaker.sol
│   ├── TimelockState.sol
│   └── WithdrawalBatchesQueue.sol
└── types
    ├── Duration.sol
    ├── ETHValue.sol
    ├── IndexOneBased.sol
    ├── PercentD16.sol
    ├── SharesValue.sol
    └── Timestamp.sol
```

During this audit, the Lido team indicated plans to replace the committee contracts with an alternative system. Consequently, the audit placed less emphasis on securing the existing committee contracts. Effectively, three contracts where removed from the scope:

```
contracts
└── committees
    ├── EmergencyActivationCommittee.sol
    ├── EmergencyExecutionCommittee.sol
    └── ResealCommittee.sol
```

System Overview

The Dual Governance contracts have been designed to address situations where Lido stakers disagree with governance proposals that have been approved by LDO holders. Previously, Lido stakers could only vote on governance proposals, that would then be executed after a predetermined timelock period. However, the previous approach could lead to conflicts between Lido stakers and LDO holders, as there was no mechanism to adequately resolve disputes or manage conflicting interests. The newly introduced dual governance contracts empower Lido stakers to delay the implementation of proposals and exit the system before any governance changes are enacted.

Lido stakers who oppose a pending governance proposal can deposit StEth, wstEth, and Lido Withdrawal NFT's into an `Escrow` contract. These assets contribute towards the *Rage Quit* threshold, in proportion to the underlying ETH value. Assets deposited in the *Rage Quit* escrow have withdrawals restricted by a delay before they can be withdrawn, ensuring that stakers cannot repeatedly deposit and initiate *Rage Quit*. This allows stakers to delay proposals but not permanently block them.

At the smart contract level, transitions between different states in the `DualGovernance` contracts enforce this logic. The states include *Normal*, *Veto Signalling*, *VetoSignalling Deactivation*, *Veto Cooldown*, and *Rage Quit*. State transitions involve time delays and are primarily triggered by changes in *Rage Quit* thresholds. When the dual governance contract is initialized or when the state transitions to *Rage Quit*, a new `Escrow` contract is deployed. The *Veto Cooldown* state exists to prevent repeated shifts between *Rage Quit* and *Veto Signalling* states.

The gate seal and reseal functionality in the Lido contracts means that the key Lido functions such as withdrawals can be paused. However, this would also prevent the usual functioning of the Dual Governance system as users would not be able to deposit and withdraw from the `Escrow` contracts. Tiebreaker logic has been implemented to handle these emergency pauses. In the event of a tie, the tiebreaker committee can execute any pending proposal and unpause any of the paused protocol contracts.

Security Model And Trust Assumptions

Privileged Roles

The Dual Governance system relies on the integrity of both the committees and the admin executor to prevent misuse of their privileged access. The impact of committee actions is constrained by the admin executor's authority to revoke committee roles and the fact that committee privileges have been designed to only be executable once.

Committees:

The committee multisigs have certain privileged functionality.

- **Gate Seal Emergency Committee** can pause certain protocol functionality. After pausing once, the **Gate Seal Emergency Committee** loses its power.
- The **Tiebreakers Committee** can push proposals after a tie.
- The **Reseal Committee** can turn a temporary pause into a permanent pause.

Admin Executor

The Admin Executor of the `DualGovernance` contract can:

- configure dual governance settings.
- register and unregister proposers.
- add and remove tiebreaker-sealed withdrawal blockers.
- set the tiebreaker committee and activation timeout.
- set the reseal committee.

The Admin Executor of the `EmergencyProtectedTimelock` can:

- set the governance and emergency governance addresses.
- define the submission and scheduling delays.
- establish the emergency protection activation committee, execution committee, protection end date, and mode duration.
- transfer executor ownership to any address.

Medium Severity

M-01 `MAX_EMERGENCY_PROTECTION_DURATION` set to Incorrect Value

An incorrect value is being assigned to `MAX_EMERGENCY_PROTECTION_DURATION` in the constructor of the `EmergencyProtectedTimelock` contract.

In the constructor of the `EmergencyProtectedTimelock` contract, consider changing `sanityCheckParams.maxEmergencyModeDuration` to `sanityCheckParams.maxEmergencyProtectionDuration`.

Update: *Resolved in pull request #156.*

Low Severity

L-01 Undocumented Parameter

Within `TimelockedGovernance.sol`, inside the `submitProposal` function, the `metadata` parameter is undocumented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API.

Update: *Resolved in pull request #157.*

L-02 Incorrect Encoding of Sealable Resume Proposal

The `_encodeSealableResume` function in the `TiebreakerSubCommittee` contract incorrectly encodes the sealable resume proposal data by omitting the `ProposalType.ResumeSealable` value. Currently, it only encodes the `sealable` address and `nonce`, whereas it should also include a prefix `ProposalType.ResumeSealable`, similar to the encoding in the `TiebreakerCoreCommittee` contract.

This could lead to a collision with the encoded schedule proposal. For a schedule proposal, the encoding consists of `ProposalType.ScheduleProposal` (which is represented by value `0`) and the `proposalId`. In the case of a resume sealable, using a zero address for the `sealable` parameter along with the `nonce` equal to the `proposalId` could potentially end up with the same encoding.

Consider adding the `ProposalType.ResumeSealable` to the encoding within the `_encodeSealableResume` function.

Update: *Resolved in pull request #158.*

L-03 Ability to Vote Multiple Times on the Same Proposal in Tiebreaker Contracts

It is possible to vote for the same proposal multiple times using the `scheduleProposal` function in both the `TiebreakerSubCommittee` and `TiebreakerCoreCommittee` contracts. This results in the emission of multiple `Vote` events which could cause issues for off-chain applications that rely on the `Vote` event for monitoring.

Consider adding a check in the `scheduleProposal` function to ensure that a proposal cannot be voted on more than once.

Update: *Acknowledged, not resolved. The Lido team stated:*

> *Acknowledged. Off-chain services that rely on the Vote event for monitoring should be prepared to handle multiple votes on the same proposal to ensure accurate tracking and representation of the voting activity.*

L-04 The Parameters Passed Into The `InvalidClaimableAmount` Error Are Swapped

The parameters passed to the `InvalidClaimableAmount` error are in the wrong order. The `claimableAmount` is passed as the expected value, while `unstETHRecord.claimableAmount` is passed as the actual value. This should be reversed.

Consider switching the order of the parameters in the `InvalidClaimableAmount` error. Alternatively, change the order of the `expected` and `actual` parameters within the error definition.

Update: *Resolved in pull request #159.*

L-05 Missing Zero-Address Checks

When assigning an address from a user-provided parameter, it is crucial to ensure the supplied address is not zero. Accidentally setting storage variables, such as member addresses, or using the zero address for sealable addresses may lead to incorrect behavior or misconfiguration of the protocol.

Throughout the codebase, multiple instances of missing zero-address checks were identified:

- The addition of new members through the `newMembers` function within `HashConsensus.sol` should not allow a zero address to be added as a member.
- The address of `sealable` used by the `voteReseal` function within `ResealCommittee.sol` should not be a zero address.
- The address of `sealable` used by the `sealableResume` function within `TiebreakerCoreCommittee.sol` should not be a zero address.
- The address of `sealable` used by the `sealableResume` function within `TiebreakerSubCommittee.sol` should not be a zero address.

Consider adding zero-address checks to the instances listed above.

Update: *Resolved in pull request #160.*

L-06 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- The entire `DualGovernance`, `Escrow`, `Executor` contracts.
- Within libraries `AssetsAccounting`, `DualGovernanceConfig`, `DualGovernanceStateMachine`, `DualGovernanceStateTransitions`, `ExecutableProposals`, `ExternalCalls`.
- Within types libraries `Durations`, `ETHValues`, `IndicesOneBased`, `PercentsD16`, `SharesValues`, and `Timestamps`.
- Within `EmergencyProtectedTimelock.sol`, the `MAX_AFTER_SUBMIT_DELAY`, `MAX_AFTER_SCHEDULE_DELAY`, `MAX_EMERGENCY_MODE_DURATION`,

`MAX_EMERGENCY_PROTECTION_DURATION` states variables and the `setGovernance`, `setupDelays`, `getGovernance`, `getAdminExecutor`, `getAfterSubmitDelay` and `getAfterScheduleDelay` functions.
- All the interfaces.
- In `ResealCommittee.sol`, the `DUAL_GOVERNANCE` state variable.
- In `ResealManager.sol`, the `PAUSE_INFINITELY` and the `EMERGENCY_PROTECTED_TIMELOCK` state variables.
- In `TiebreakerCoreCommittee.sol`, the `DUAL_GOVERNANCE` state variable and the `sealableResume` function.
- In `TiebreakerSubCommittee.sol`, the `TIEBREAKER_CORE_COMMITTEE` state variable

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #161. The Lido team stated:*

> *To address the issue of missing docstrings, documentation has been added to the main libraries and contracts, as outlined in PR #161. Types and interfaces were not covered in this update.*

L-07 Incorrect Comparison Used for Tiebreaker Voting

The quorum check should use `>=` instead of `==`. If the owner changes the `quorum` via `setQuorum`, some proposals may require scheduling through the `schedule` function if the current support value is greater than or equal to the new quorum, as the code within `_vote` will not execute.

Consider replacing `==` with `>=`.

Update: *Acknowledged, not resolved. The Lido team stated:*

> *Acknowledged. This scenario is expected to be very rare. However, to address it, the public method `HashConsensus.schedule` is intentionally provided to allow anyone to schedule proposals that meet or exceed the quorum without requiring committee members to use the `vote` method. This ensures proposals are properly processed even if the quorum has changed.*

L-08 `Escrow.requestNextWithdrawalsBatch` Might Be Executed While the Batch Queue Is in a Closed State

The `requestNextWithdrawalsBatch` function may be invoked multiple times until all stETH is converted into withdrawal NFTs. Upon each execution, the function tracks the IDs of the withdrawal requests generated by these invocations. When the remaining stETH balance of the contract falls below `WITHDRAWAL_QUEUE.MIN_STETH_WITHDRAWAL_AMOUNT()`, the generation of withdrawal batches concludes, and subsequent function calls will revert. However, it is possible to transfer stETH to the contract, causing the balance to exceed `minStETHWithdrawalRequestAmount`. In this scenario, `_batchesQueue.close()` is skipped, meaning that the verification that `_batchesQueue` is in the open state is also bypassed.

To prevent the issue described above, consider adding the `_checkInOpenedState` check to the `requestNextWithdrawalsBatch` function. This will ensure that `requestNextWithdrawalsBatch` can only be executed when the queue is in the open state.

Update: *Acknowledged, not resolved. The Lido team stated:*

> *Acknowledged. In the described scenario, the `_checkInOpenedState` check will not be skipped because it is invoked within the `_batchesQueue.addUnstETHIds()` method. This ensures that after the queue is closed, any attempt to call the `requestNextWithdrawalsBatch` method will be prevented by the state `check`.*

L-09 Missing Check in `Escrow.requestNextWithdrawalsBatch`

The `requestNextWithdrawalsBatch` function of the `Escrow` contract begins by checking if the remaining amount of stETH is smaller than either `_MIN_TRANSFERRABLE_ST_ETH_AMOUNT` or `minStETHWithdrawalRequestAmount`. If this condition is met, it closes the batch. At the end of the execution, a similar check is performed, but it only verifies against `minStETHWithdrawalRequestAmount`.

Consider including `_MIN_TRANSFERRABLE_ST_ETH_AMOUNT` in a check to verify if the `_batchesQueue` should be closed.

Update: *Resolved in pull request #173.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved.*

N-02 Unnecessary Cast

Within the `ResealManager` contract, the `ITimelock(emergencyProtectedTimelock)` cast is unnecessary.

To improve the overall clarity and intent of the codebase, consider removing any unnecessary casts.

Update: *Resolved in pull request #174.*

N-03 Unused Errors

Throughout the codebase, multiple instances of unused errors were identified:

- The `InvalidSecondSealRageSupport` error in `DualGovernanceConfig.sol`
- The `TimestampUnderflow` error in `Timestamp.sol`

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #162.*

N-04 Duplicate Event Emissions

If a setter function does not verify whether the value has changed, it opens up the possibility for event spamming, incorrectly signaling that the value has changed when it has not. Repeatedly emitting identical values can confuse off-chain clients. The `setResealCommittee` function sets `_resealCommittee` and emits an event without checking if the value has changed.

Consider adding a check statement to revert the transaction if the value remains unchanged.

Update: *Resolved in pull request #163.*

N-05 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Throughout the codebase, multiple instances of mappings without named parameters were identified:

- The `_hashStates` state variable in the `HashConsensus` contract
- The `approves` state variable in the `HashConsensus` contract
- The `_resealNonces` state variable in the `ResealCommittee` contract
- The `_sealableResumeNonces` state variable in the `TiebreakerCoreCommittee` contract

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Resolved in pull request #164.*

N-06 File and Library Name Mismatch

The file name of `WithdrawalBatchesQueue.sol` does not match the `WithdrawalsBatchesQueue` library name.

To make the codebase easier to understand for developers and reviewers, consider renaming the `WithdrawalBatchesQueue.sol` file to match the library name.

Update: *Resolved in pull request #165.*

N-07 Discrepancies Between Interfaces and Implementation Contracts

Throughout the codebase, multiple instances of discrepancies between the interfaces and implementation contracts were identified:

- The `resealSealable` function uses `sealable` as the parameter name, whereas the interface uses `sealables`.
- The `startRageQuit` function implementation uses the `rageQuitExtensionPeriodDuration` and `rageQuitEthWithdrawalsDelay` parameters, whereas the interface uses `rageQuitExtraTimelock` and `rageQuitWithdrawalsTimelock`.
- The `scheduleProposal` function uses the `proposalId` parameter, whereas the interface uses `_proposalId`.
- The `transferShares` function should return a value of type `uint256`.
- The interface function `getProposal` returns the parameter `proposal`, whereas the implementation returns `proposalDetails`.
- The interface function `setGovernance` uses the `governance` parameter, whereas the implementation uses `newGovernance`.

To ensure consistency and clarity, consider aligning the parameter names across interfaces and implementation contracts.

Update: *Resolved in pull request #166.*

N-08 Missing Underflow Check in `SharesValue`

The `minus` function does not include any check to ensure that the value of the `v1` parameter is greater than or equal to the value of the `v2` parameter.

Consider adding a check that reverts with an error message if the value of `v2` exceeds the value of `v1`.

Update: *Resolved in pull request #167.*

N-09 Lack of Indexed Event Parameters for State Changes

Throughout the codebase, there are state-change events that do not have indexed parameters:

- The `DualGovernanceStateChanged` event in `DualGovernanceStateMachine.sol`
- The `EscrowStateChanged` event in `EscrowState.sol`

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

Update: *Resolved in pull request #168.*

N-10 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions with unnecessarily permissive visibility were identified:

- The `canSubmitProposal` function in `DualGovernance.sol` with `public` visibility could be limited to `external`.
- The `isEmergencyProtectionEnabled`, `isEmergencyModeActive`, and `getEmergencyProtectionDetails` functions in `EmergencyProtectedTimelock.sol` with `public` visibility could be limited to `external`.
- The `addMembers`, `removeMembers`, `getMembers`, `isMember` `setTimelockDuration`, `setQuorum`, and `schedule` functions in `HashConsensus.sol` with `public` visibility could be limited to `external`.
- The `getProposals`, `getProposalAt`, `getProposal`, `getProposalsLength`, and `getOrderedKeys` functions in `ProposalsList.sol` with `public` visibility could be limited to `external`.
- The `voteReseal` and `getResealState` functions in `ResealCommittee.sol` with `public` visibility could be limited to `external`.

- The `reseal` and `resume` functions in `ResealManager.sol` with `public` visibility could be limited to `external`.
- The `scheduleProposal`, `getScheduleProposalState`, `executeScheduleProposal`, `getSealableResumeNonce`, `sealableResume`, and `getSealableResumeState` functions in `TiebreakerCoreCommittee.sol` with `public` visibility could be limited to `external`.
- The `scheduleProposal`, `getScheduleProposalState`, `executeScheduleProposal`, `sealableResume`, `getSealableResumeState`, and `executeSealableResume` functions in `TiebreakerSubCommittee.sol` with `public` visibility could be limited to `external`.

To better convey the intended use of functions and to potentially accrue additional gas savings, consider changing a function's visibility to be only as permissive as required.

Update: *Resolved in pull request #169.*

N-11 Multiple Library Declarations Per File

The `DualGovernanceStateMachine.sol` file currently contains definitions of two libraries: `DualGovernanceStateMachine` and `DualGovernanceStateTransitions`.

Consider separating the aforementioned two libraries into their own files to make the codebase easier to understand for developers and reviewers.

Update: *Resolved in pull request #172.*

N-12 Inconsistent Order Within Contracts

Throughout the codebase, there are multiple contracts that deviate from the Solidity Style Guide due to having inconsistent ordering of functions. Functions should be grouped and ordered based on permissiveness, and within a grouping the `view` and `pure` functions should come last.

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

Update: *Acknowledged, not resolved. The Lido team stated:*

> *Acknowledged. The function ordering in the contracts deviates from the Solidity Style Guide due to the project's size and complexity. The code is organized to generalize*

> *functionality and assist external researchers and auditors in understanding the system by grouping related functionalities together, even if it does not strictly adhere to standard ordering guidelines.*

N-13 Incorrect Event Parameter Name

The parameter name used in the `EmergencyExecutionCommitteeSet` event should be `newExecutionCommittee`, not `newActivationCommittee`.

Consider correcting the aforementioned parameter name.

Update: *Resolved in pull request #170.*

N-14 Confusing Variable Names

Throughout the codebase, multiple instances of confusing variable and parameter names were identified:

- In the `unlockWstETH` function, the result of the `accountStETHSharesUnlock` function is called `wstETHUnlocked`, which should be renamed to `unlockedStETHShares`. Conversely, the result of `WST_ETH.wrap` is called `unlockedStETHShares`, but it should be named `wstETHUnlocked`.
- The `RageQuitStarted` event uses `rageQuitExtensionDuration` as the parameter name, while `rageQuitExtensionPeriodDuration` is used consistently elsewhere. Consider changing the parameter name to `rageQuitExtensionPeriodDuration` for consistency.

Update: *Resolved in pull request #175.*

N-15 Discrepancies Between Documentation and Implementation

Certain contract behaviors differ from the provided documentation:

- The `3_Key_Properties_of_Dual_Governance` markdown file states that:

  > Proposals cannot be executed in the Veto Signalling (both parent state and Deactivation sub-state) and Rage Quit states.

However, the dual governance state is not verified in the `execute` function.

- The `isTie` function returns `true` when the state is `RageQuit` and a sealable contract is blocked. This does not match the specifications, which also require the pause duration to exceed `TiebreakerActivationTimeout`:

  > *Tiebreaker Condition A**: (governance state is Rage Quit) $\land$ (protocol withdrawals are paused for a duration exceeding `TiebreakerActivationTimeout`).

Consider updating the documentation to align with the new specification.

Update: *Partially resolved in pull request #171. The Lido team stated:*

> *First Point: `3_Key_Properties_of_Dual_Governance` markdown file is authored by an external research team and is not within the scope of our current work. Second Point: Fixed in pull request #171.*

N-16 Gas Optimizations

Throughout the codebase, multiple opportunities for gas optimization and code quality improvement were identified:

- There are multiple instances where a value is read from storage within an `if` statement and then either assigned to another variable or used within the `if` block. This results in increased gas usage due to two storage reads being performed. Consider caching the value from storage before the `if` statement to optimize gas consumption. Some of the examples include:
    - Use of `status`
    - Use of `lockedBy` in `AssetsAccounting.sol`
    - Use of `emergencyProtectionEndsAfter` in `EmergencyProtection.sol`
- In the EVM, it is more gas-efficient to read values from stack than from memory or state. As in a `for` loop, where a value is read repeatedly, it is more efficient to write the length of an array to stack and reuse the stack variable. Consider writing the array length to stack like `uint256 arrayLength = array.length;` and then using the `arrayLength` variable to save gas. There are various instances where this optimization could be applied:
    - In line 207 of `HashConsensus.sol`
    - In line 224 of `HashConsensus.sol`
    - In line 239 of `HashConsensus.sol`

    - In line 138 of `Proposers.sol`
- The difference between `map._orderedKeys.length` and `offset` is calculated twice. Once within the if statement and once when assigning the result to the `keysLength`. Consider calculating the difference once.

Update: *Acknowledged, not resolved.*

Conclusion

Lido's Dual Governance codebase is a work in progress that, during the audit, was undergoing multiple code and design-level changes. While the codebase benefits from formal verification tests which help strengthen the invariants, it lacks sufficient unit test coverage.

Communication with the team was smooth, and they openly expressed their considerations regarding the design. Furthermore, they promptly initiated the implementation of the suggested solutions to address the vulnerabilities disclosed early in the audit.

Monitoring Recommendations

While the audit helped identify code-level issues in the current implementation, the Lido team is encouraged to implement monitoring operations for deployed contracts. Specifically, it is recommended to monitor the Escrow contract and the generated rage threshold. In addition, monitoring the changes in the balances of StETH and Withdrawal NFTs within the Escrow contract throughout the stages of Signalling Escrow and Rage Quit Escrow could help detect any issues early.