STATE MIND

Lido Triggerable Withdrawals and CSM V2

06-06-2025 - 19-09-2025

Table of contents

1. Project brief

Title	Description
Client	Lido
Project name	Lido Triggerable Withdrawals and CSM V2
Timeline	06-06-2025 - 19-09-2025

Project Log

[easy-track]

Date	Commit Hash	Note
12-09-2025	e36fee12cf8dec32c552ed1131c61f53f1914636	Initial Commit for ET factory scope

[community-staking-module]

Date	Commit Hash	Note
21-07-2025	d63d123f24e2ed2fb2f039238e7562a3d61532b2	Initial Commit for CSM V2 scope
15-09-2025	0e4b562719cca51070c9cede5e5a8505eca18684	Reaudit commit for CSM V2 scope

[core]

Date	Commit Hash	Note
06-06-2025	8beee976ff15472e2ab01fb0247741989ca691ef	Initial Commit for TW scope
09-06-2025	628c8736d12478fc9e9a7dcba7dc2e7e6ebb8715	Updated commit for TW scope
20-06-2025	eb47249b786c816678490e981b5eebe91f524b49	Updated commit 2 for TW scope
23-07-2025	cfa0c6a3605aabed41d0200d6a7c32d6b71e91b4	Reaudit commit for TW scope
01-09-2025	acf3188c79e5616ef7594999f606473214e10f6b	Reaudit commit 2 for TW scope

Short Overview

The Lido Core update, which introduces triggerable withdrawals, is centered on supporting EIP-7002. In the Pectra hard fork, EIP-7002 enables validators to trigger withdrawals and exits using their execution-layer withdrawal credentials directly. This

removes the prior limitation where only the active validator key could initiate withdrawals, ensuring the holder of the withdrawal credentials can also manage staked ETH independently and securely.

Within Lido Core, several contracts were modified and new ones added:

- Since Lido operators' withdrawal credentials point to the WithdrawalVault address, its implementation was updated to allow submitting exit requests.
- A new contract, TriggerableWithdrawalsGateway.sol, triggers the relevant functions on WithdrawalVault and also notifies modules via the StakingRouter.
- Another new contract, ValidatorExitDelayVerifier, checks for delayed validator exits and reports them to the StakingRouter for penalization.
- The VEBO oracle was updated to adjust the roles permitted to submit exit requests and to add a permissionless method for on-chain exit triggering for validators that already have a submitted report.

CSM is a permissionless staking module designed to allow community stakers to participate in the Lido on Ethereum protocol as Node Operators. The only requirement to join CSM as a Node Operator is the ability to run validators and provide a bond as security collateral. Stake is allocated to validator keys in the order they are submitted, provided the keys are valid. CSM V2 is a major upgrade with the following features:

- EIP-7002 support. It enables voluntary exit triggers for Node Operators through CSEjector.sol and allows ejecting poorly performing validators through the Strike system.

- Configurable bond curves. This enables per operator type configuration for use by both on-chain and off-chain components of CSM.

- The Strike system, implemented in CSStrikes.sol, protects CSM from systematic poor performance. If validators exhibit sustained unacceptable performance, they can be ejected in a permissionless manner.

Project Scope

The audit covered the following files:

- CSMSetVettedGateTree.sol
- CSModule.sol
- SigningKeys.sol
- NOAddresses.sol
- QueueLib.sol
- PausableUntil.sol
- AssetRecovererLib.sol
- TransientUintUintMapLib.sol
- ValidatorCountsReport.sol
- AssetRecoverer.sol
- UnstructuredStorage.sol
- CSAccounting.sol
- CSBondCurve.sol
- CSBondCore.sol
- CSBondLock.sol
- CSStrikes.sol
- CSEjector.sol
- CSExitPenalties.sol
- ExitTypes.sol
- CSVerifier.sol
- SSZ.sol
- GIndex.sol
- Types.sol
- BaseOracle.sol
- CSFeeDistributor.sol
- CSFeeOracle.sol
- Versioned.sol
- HashConsensus.sol
- CSParametersRegistry.sol
- VettedGate.sol
- PermissionlessGate.sol
- VettedGateFactory.sol
- OssifiableProxy.sol
- NodeOperatorsRegistry.sol
- ValidatorExitDelayVerifier.sol
- LidoLocator.sol
- StakingRouter.sol
- TriggerableWithdrawalsGateway.sol
- WithdrawalVault.sol
- WithdrawalVaultEIP7002.sol
- AccountingOracle.sol
- ValidatorsExitBus.sol
- ValidatorsExitBusOracle.sol
- ExitLimitUtils.sol
- BeaconTypes.sol
- GIndex.sol
- SSZ.sol

2. Finding severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Client regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	5 (2 fixed, 3 acknowledged)
Informational	21 (15 fixed, 6 acknowledged)
Total	26 (17 fixed, 9 acknowledged)

4. Conclusion

During the audit of the codebase, 26 issues were found in total:

- 5 medium severity issues (2 fixed, 3 acknowledged)
- 21 informational severity issues (15 fixed, 6 acknowledged)

The final reviewed commits are acf3188c79e5616ef7594999f606473214e10f6b, 0e4b562719cca51070c9cede5e5a8505eca18684, e36fee12cf8dec32c552ed1131c61f53f1914636

Deployment

TW scope

Contract	Address
LidoLocator (Implementation)	0x2C298963FB763f74765829722a1ebe0784f4F5Cf
AccountingOracle (Implementation)	0xE9906E543274cebcd335d2C560094089e9547e8d
ValidatorsExitBusOracle (Implementation)	0x905A211eD6830Cfc95643f0bE2ff64E7f3bf9b94
WithdrawalVault (Implementation)	0x7D2BAa6094E1C4B60Da4cbAF4A77C3f4694fD53D

StakingRouter (Implementation)	0x226f9265CBC37231882b7409658C18bB7738173A
NodeOperatorsRegistry (Implementation)	0x6828b023e737f96B168aCd0b5c6351971a4F81aE
ValidatorExitDelayVerifier (Implementation)	0xbDb567672c867DB533119C2dcD4FB9d8b44EC82f
TriggerableWithdrawalsGateway	0xDC00116a0D3E064427dA2600449cfD2566B3037B
GateSeal (for Withdrawal Queue)	0x8A854C4E750CDf24f138f34A9061b2f556066912
GateSeal (for Triggerable Withdrawals scope)	0xA6BC802fAa064414AA62117B4a53D27fFfF741F1

CSM V2 scope

Contract	Address
CSAccounting (Proxy)	0x4d72BFF1BeaC69925F8Bd12526a39BAAb069e5Da
CSAccounting (Implementation)	0x6f09d2426c7405C5546413e6059F884D2D03f449
CSEjector	0xc72b58aa02E0e98cF8A4a0E9Dce75e763800802C
CSExitPenalties (Proxy)	0x06cd61045f958A209a0f8D746e103eCc625f4193
CSExitPenalties (Implementation)	0xDa22fA1CEa40d05Fe4CD536967afdD839586D546
CSFeeDistributor (Proxy)	0xD99CC66fEC647E68294C6477B40fC7E0F6F618D0
CSFeeDistributor (Implementation)	0x5DCF7cF7c6645E9E822a379dF046a8b0390251A1
CSFeeOracle (Proxy)	0x4D4074628678Bd302921c20573EEa1ed38DdF7FB
CSFeeOracle (Implementation)	0xe0B234f99E413E27D9Bc31aBba9A49A3e570Da97
CSModule (Proxy)	0xdA7dE2ECdDfccC6c3AF10108Db212ACBBf9EA83F
CSModule (Implementation)	0x1eB6d4da13ca9566c17F526aE0715325d7a07665
CSParametersRegistry (Proxy)	0x9D28ad303C90DF524BA960d7a2DAC56DcC31e428
CSParametersRegistry (Implementation)	0x25fdC3BE9977CD4da679dF72A64C8B6Bd5216A78
CSStrikes (Proxy)	0xaa328816027F2D32B9F56d190BC9Fa4A5C07637f
CSStrikes (Implementation)	0x3E5021424c9e13FC853e523Cd68ebBec848956a0
CSVerifier	0xdC5FE1782B6943f318E05230d688713a560063DC

PermissionlessGate	0xcF33a38111d0B1246A3F38a838fb41D626B454f0
VettedGate (Proxy)	0xB314D4A76C457c93150d308787939063F4Cc67E0
VettedGate (Implementation)	0x65D4D92Cd0EabAa05cD5A46269C24b71C21cfdc4
VettedGateFactory	0xFdab48c4D627e500207e9AF29c98579d90Ea0ad4
GateSeal (for CSM V2 scope)	0xE1686C2E90eb41a48356c1cC7FaA17629af3ADB3
NOAddresses	0xe4d5a7be8d7c3db15755061053f5a49b6a67fffc
QueueLib	0x6eff460627b6798c2907409ea2fdfb287eaa2e55

5. Findings report

MEDIUM-01	Griefing TriggerableWithdrawalsGateway limits	Acknowledged

Description

Lines:

- ValidatorsExitBus.sol#L291
- TriggerableWithdrawalsGateway.sol#L177

The **ValidatorsExitBus** does not track which validators were triggered via the **ValidatorsExitBus.triggerExits()** function.
Each **ValidatorsExitBus.triggerExits()** call consumes exit limits of the **TriggerableWithdrawalsGateway** contract.
During the low fees of the withdrawal requests (e.g. 1 wei fee in a single block), an arbitrary user can exhaust all exit limits of the **TriggerableWithdrawalsGateway** by continuously calling **ValidatorsExitBus.triggerExits()** with the same payload.

Recommendation

We recommend preventing duplicate processing of the validators that have been triggered.

Client's comments

Unfortunately, allowing this behavior is not safe. There are specific flows that can cause such trigger requests to be ignored on the CL, which can prevent us from ever being able to trigger an exit for that validator again. This risk arises in the following cases:

- The validator is not yet active.
- The validator has not served for at least SHARD_COMMITTEE_PERIOD epochs.
- The validator has pending partial withdrawals.
- The withdrawal credentials do not match.

Reference: Electra spec – process_withdrawal_request
While the last two scenarios are unlikely in practice, the first one (validator is too young) is very plausible.
We considered an alternative where we would allow retriggering exits for validators we had previously attempted to exit, without consuming exit limits. However, this introduced a new vulnerability: it would become possible (e.g. via the CSM Ejector) to trigger exits for brand new validators. Later on, the CSM may then bypass exit limits when actually executing the exit, which opens the door to abuse.
Thus, the current behavior is intentional.

MEDIUM-02	Penalties for invalid requests	Fixed at: 498d5da

Description

Lines:

- ValidatorsExitBus.sol#L311
- ValidatorExitDelayVerifier.sol#L165
- ValidatorExitDelayVerifier.sol#L210

The **ValidatorsExitBus** contract enforces that exit requests can only be triggered on the specific contract version where they were originally submitted.

The **ValidatorExitDelayVerifier**'s functions **verifyValidatorExitDelay()** and **verifyHistoricalValidatorExitDelay()** have all relevant checks (data validity, submitted, delivered) but lack the contract version check.

The following is possible:

1. The **ValidatorsExitBus** version is **x**;
2. **SUBMIT_REPORT_HASH_ROLE** submits the hash and data;
3. The **ValidatorsExitBus** contract upgrades to version x + 1.
4. The validator cannot trigger exit via **ValidatorsExitBus.triggerExits()** and gets penalized.

Recommendation

We recommend disallowing penalties for exit requests that are no longer valid.

MEDIUM-03	[CSM] Double counting of enqueued keys after using CSModule.migrateToPriorityQueue()	Acknowledged

Description

Lines:

- CSModule.sol#L613
- CSModule.sol#L1530

All node operators who had seats in the queue in CSM v1 are migrated into V2 with **LEGACY_QUEUE_PRIORITY**. Eligible operators can call **CSModule.migrateToPriorityQueue()**. During that migration, **CSModule._enqueueNodeOperatorKeys()** increases **no.enqueuedCount** in storage.

Example: a node operator with 10 keys queued in V1 calls **CSModule.migrateToPriorityQueue()**. Before migration, **no.enqueuedCount == 10**; assuming **maxDeposits >= 10**, after migration it becomes **no.enqueuedCount == 20**. This doubles the number of stored keys, but the number of keys available for deposit remains unchanged.

Furthermore, the operator can enqueue up to **no.enqueuedCount** additional keys, and those will be processed with **LEGACY_QUEUE_PRIORITY** (which outranks **QUEUE_LOWEST_PRIORITY**), granting them unjustified priority over operators limited to the lowest-priority queue.

Recommendation

We recommend implementing a mechanism to remove the legacy queue seats as part of the migration, so they aren't double-counted.

Client's comments

> We find the described behaviour expected. Corresponding comment to the function was updated in - https://github.com/lidofinance/community-staking-module/commit/2584b43815cf9cf52b2fbd92a007819e66912039

MEDIUM-04	[CSM] A new Node Operator can migrate via **migrateToPriorityQueue**	Acknowledged

Description

Lines:

- CSModule.sol#L587-L600
- CSModule.sol#L1494-L1512

Per **ICSModule.migrateToPriorityQueue()** description of the **CSModule.migrateToPriorityQueue()** function, it **Performs a one-time migration of allocated seats from the legacy queue to a priority queue for an eligible node operator**.
But user can migrate keys from lowest queue to priority queue, breaking the spec and recieving queue seats without adding new keys.
An eligible Node Operator from CSMv1 is identified via two conditions:

1. Has not used a priority queue (**no.usedPriorityQueue == False**);
2. Has a **curveID** with a priority higher than **QUEUE_LOWEST_PRIORITY**.

If a Node Operator with the lowest priority is later recognized for their activity and rewarded with improved parameters — including access to the priority queue — they can still invoke **CSModule.migrateToPriorityQueue()** as if they had participated in CSMv1.
This is possible because **usedPriorityQueue** is not flipped when enqueuing into the lowest priority.

Recommendation

We recommend repurposing the **usedPriorityQueue** flag to the **usedQueue** while setting it to **True** regardless of the queue priority.

Client's comments

We find the described behaviour expected. Corresponding comment to the function was updated in - https://github.com/lidofinance/community-staking-module/commit/2584b43815cf9cf52b2fbd92a007819e66912039

MEDIUM-05	[CSM] CSModule.getNodeOperatorSummary() does not cover all cases	Fixed at: 2bc03e9

Description

Lines: CSModule.sol#L1156

This view function retrieves information about a node operator, primarily to determine their position in the withdrawal queue.

It handles three scenarios:

1. Force mode enabled & unbonded > non-deposited:
   - Sets **targetLimitMode = FORCED_TARGET_LIMIT_MODE_ID**.
   - Calculates

```
targetValidatorsCount = min(
  no.targetLimit,
  no.totalAddedKeys - no.totalWithdrawnKeys - totalUnbondedKeys
);
```

2. Force mode disabled & unbonded > non-deposited:
   - Also sets **targetLimitMode = FORCED_TARGET_LIMIT_MODE_ID**.
   - Calculates

```
targetValidatorsCount =
  no.totalAddedKeys - no.totalWithdrawnKeys - totalUnbondedKeys;
```

3. Otherwise (no unbonded > non-deposited):
   - Uses the operator's configured mode and limit:

```
targetLimitMode = no.targetLimitMode;
targetValidatorsCount = no.targetLimit;
```

The second branch fails to respect **no.targetLimit** when **no.targetLimitMode == 1**. In effect, operators whose force mode is already enabled (mode 1) but who have some unbonded keys end up in branch 2 and bypass the **min(..., no.targetLimit)** clamp.

Example:

- Operator1: **totalAddedKeys = 10**, **totalDepositedKeys = 10**, **totalUnbondedKeys = 2** (bonded = 8), **targetLimitMode = 1**, **targetLimit = 6**.
- Operator2: **totalAddedKeys = 10**, **totalDepositedKeys = 10**, **totalUnbondedKeys = 0** (bonded = 10), **targetLimitMode = 1**, **targetLimit = 6**.

Both have **targetLimitMode == 1** and **totalWithdrawnKeys == 0**. On calling the view:

- Operator1 matches branch 2 → **targetValidatorsCount = 8** (ignores **targetLimit**).
- Operator2 matches branch 3 → **targetValidatorsCount = 6** (respects **targetLimit**).

If a high withdrawal demand clears validators in stages 1–3, Operator1 would lose 2 validators, while Operator2 loses 4, even though Operator2 has no unbonded keys. This inconsistency gives an unintended advantage to Operator1.

One possible way for Operator1 to make keys unbonded is to trigger **CSModule.reportELRewardsStealingPenalty()** and then wait for the bond to decrease after **CSModule.settleELRewardsStealingPenalty()**. This action requires bot EOA or EasyTrack activity. However, the 72-hour period in EasyTrack makes abuse more difficult, as the withdrawal demand must be anticipated several days in advance.

Recommendation

We recommend considering an operator with at least one unbonded key as having **targetLimitMode = 2** and the same **targetLimit**.

INFORMATIONAL-01	Inconsistent exit requests events	Fixed at: a9bc3df

Description

The **ValidatorsExitBusOracle** (including **ValidatorsExitBus**) contract supports two distinct flows to submit exit requests.
The first one is an old oracle reporting via **HashConsensus.submitReport()** and **ValidatorsExitBusOracle.submitReportData()**.
The second is a new logic added in the **ValidatorsExitBus** contract.
The main difference, apart from caller roles, is that the **ValidatorsExitBus** flow updates **RequestStatus** step-by-step (set **contractVersion**, set **deliveredExitDataTimestamp**). In contrast, the **ValidatorsExitBusOracle.submitReportData()** sets both at the end of the call.
There are scenarios, where a combination of these two reporting methods creates inconsistent event emissions.

1. **ValidatorsExitBus.submitExitRequestsHash()** - sets **contractVersion**, emits **RequestsHashSubmitted()**.
2. **ValidatorsExitBusOracle.submitReportData()** - sets both **contractVersion** and **deliveredExitDataTimestamp**, emits **ValidatorExitRequest()** and **RequestsHashSubmitted()**.

The **ValidatorsExitBusOracle** checks if exit requests were delivered, but regardless of that, it emits additional event.

1. **ValidatorsExitBus.submitExitRequestsHash()** - sets **contractVersion**, emits **RequestsHashSubmitted()**.
2. **ValidatorsExitBus.submitExitRequestsData()** - sets **deliveredExitDataTimestamp**, emits **ValidatorExitRequest()**.
3. **ValidatorsExitBusOracle.submitReportData()** - emits **ValidatorExitRequest()**.

It is worth noting that **ValidatorsExitBusOracle.submitReportData()** emits events in incorrect order.

```
_handleConsensusReportData(data); // emits ValidatorExitRequest
_storeOracleExitRequestHash(dataHash, contractVersion); // emits RequestsHashSubmitted
```

Recommendation

We recommend resolving inconsistencies to better accommodate off-chain tooling.

INFORMATIONAL-02	Redundant components in the TriggerableWithdrawalsGateway contract	Fixed at: a9bc3df

Description

Lines:

- TriggerableWithdrawalsGateway.sol#L53
- TriggerableWithdrawalsGateway.sol#L90

The **InvalidRequestsDataLength()** error in the **TriggerableWithdrawalsGateway** contract is redundant because the request length validation is already handled at the **ValidatorsExitBus** contract level before any interaction with TriggerableWithdrawalsGateway.
Also, the **PUBLIC_KEY_LENGTH** constant is defined in the **TriggerableWithdrawalsGateway** contract but is never used for validating the length of validator public keys.

Recommendation

We recommend removing or implementing unused components.

INFORMATIONAL-03	Incorrect NatSpec	Fixed at: a9bc3df

Description

Lines: ValidatorsExitBus.sol#L235-L250
In the natspec before **ValidatorsExitBus.submitExitRequestsData()**, it is not clarified that the function would be reverted if the pubkeys in the report, whose hash was submitted previously, are not sorted properly.
It reverts with this error.

Recommendation

We recommend adding a line to a natspec.

```
* - The pubkeys was not sorted before the report hash submit.
```

INFORMATIONAL-04	Gas optimization	Fixed at: a9bc3df

Description

Line: ExitLimitUtils.sol#L70
In the function **ExitLimitUtils.updatePrevExitLimit()** there are calculations of time that passed between the submits. It can be optimized by using less memory.

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

    _data.prevExitRequestsLimit = uint32(newExitRequestLimit);
    _data.prevTimestamp += passedTime;
...
//NEW:
...
    uint passedTime = timestamp - _data.prevTimestamp;
    passedTime -= passedTime % _data.frameDurationInSec;

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

This optimization reduces gas consumption for ~80 gas.

Recommendation

We recommend fixing this gas optimization to reduce gas consumption.

INFORMATIONAL-05	Incorrect comparison in ValidatorExitDelayVerifier._getSecondsSinceExitIsEligible() function	Fixed at: 8b2dc8f

Description

Line: ValidatorExitDelayVerifier.sol#L343

In the **ValidatorExitDelayVerifier._getSecondsSinceExitIsEligible()** function, there is an incorrect comparison operator used when checking if a validator's exit is eligible. The current check uses **<** when comparing **referenceSlotTimestamp** with **eligibleExitRequestTimestamp**. This allows the case where **eligibleToExitInSec** will be equal to 0 and **ValidatorExitDelayVerifier** will attempt to report this value to **StakingRouter**.

The delay value 0 is incorrect. For example, **NodeOperatorRegistry.reportValidatorExitDelay()** will revert if the **ValidatorExitDelayVerifier** passes 0.

Recommendation

We recommend changing the comparison operator from **<** to **<=**.

INFORMATIONAL-06	Gas optimizations	Fixed at: 00aabce

Description

Lines:

- GIndex.sol#L76
- ValidatorExitDelayVerifier.sol#L193

1. The **GIndex.concat()** function calculates the indices of **GI**'s twice. These indexes can be saved in memory and reused.
2. The **ValidatorExitDelayVerifier.verifyValidatorExitDelay()** function stores the **validatorWitnesses** for each validator. However, an unsaved **validatorWitnesses[i]** value is then passed to the **ValidatorExitDelayVerifier._verifyValidatorExitUnset()** function.

Recommendation

We recommend adding these optimizations.

Client's comments

Fixed partially: An index ranges within w, 2w−1 in the case of a single vector- or container-like object located at the very top of a tree. If the index refers to a child object or an array-like object requiring the inclusion of a length, this condition does not hold.

INFORMATIONAL-07	Extra check in the ValidatorExitDelayVerifier._getExitRequestDeliveryTimestamp() function	Fixed at: 52dfddb

Description

Lines:

- ValidatorExitDelayVerifier.sol#L366
- ValidatorsExitBus.sol#L510

The **ValidatorExitDelayVerifier._getExitRequestDeliveryTimestamp()** function checks if the **deliveryTimestamp** value is zero. However, this check is already in the **ValidatorsExitBus._checkDelivered()** function that is called inside the **ValidatorsExitBus.getDeliveryTimestamp()** function.

Recommendation

We recommend removing the extra check.

INFORMATIONAL-08	Withdrawal delays may lead to unjust penalties	Acknowledged

Description

EIP-7002 defines a limit on the number of withdrawals that can be processed per block: **MAX_WITHDRAWAL_REQUESTS_PER_BLOCK = 16**.
Under high load or DoS conditions (although unlikely, they are realistic with low fees), the withdrawal request contract may experience significant delays in processing. This can result in validators exceeding their **exitDeadlineThreshold**, potentially leading to unjust penalties for honest validators.

Recommendation

We recommend monitoring withdrawal request congestion and penalizing validators accordingly.

Client's comments

That's quite possible — and more likely not because of the number of requests in the queue, but because of the huge fees resulting from it, which will require more ether than the validator contains. We shift this responsibility to the Node Operator side. The exit should be completed within a few days, so the risk that the Node Operator cannot exit due to network overload can arise only if the exit is initiated during the last few epochs before the deadline. Moreover, the same assumption applies in the case of sending a voluntary message to the CL.

INFORMATIONAL-09	Missing AccountingOracle version update	Fixed at: 5c78974

Description

The **AccountingOracle** contract contains updated logic, which will cause the proxy implementation to be updated. Therefore, the contract version should also be updated.

Recommendation

We recommend adding the **finalizeUpgrade_v3()** function and adding **_updateContractVersion(3)** to the **initialize()** function:

```
contract AccountingOracle is BaseOracle {

  function initialize(
      address admin,
      address consensusContract,
      uint256 consensusVersion
  ) external {
      if (admin == address(0)) revert AdminCannotBeZero();

      uint256 lastProcessingRefSlot = _checkOracleMigration(LEGACY_ORACLE, consensusContract);
      _initialize(admin, consensusContract, consensusVersion, lastProcessingRefSlot);

      _updateContractVersion(2);
+      _updateContractVersion(3);
  }

+  finalizeUpgrade_v3() external {
+     _updateContractVersion(3);
+  }

...
}
```

INFORMATIONAL–10	Redundant version check	Acknowledged

Description

Lines:

- StakingRouter.sol#L210
- Versioned.sol

The **StakingRouter.finalizeUpgrade_v3()** function checks whether the current version is an expected one and then updates the number.

```
function finalizeUpgrade_v3() external {
    _checkContractVersion(2);
    _updateContractVersion(3);
}
```

The **Versioned._updateContractVersion()** guarantees that we can update the version number only by increments of 1, hence it checks that the version is 2:

```
if (newVersion != getContractVersion() + 1) revert InvalidContractVersionIncrement();
```

Recommendation

We recommend using **Versioned._setContractVersion()** to avoid redundant checks.

Client's comments

> Although the check is indeed redundant and duplicates the existing one in _updateContractVersion, we prefer to keep it as is for code clarity.

INFORMATIONAL–11	Extra check in the NodeOperatorsRegistry.reportValidatorExitDelay() function	Acknowledged

Description

Lines:

- NodeOperatorsRegistry.sol#L1138
- ValidatorExitDelayVerifier.sol#L228
- ValidatorsExitBus.sol#L640

The **NodeOperatorsRegistry.reportValidatorExitDelay()** function contains a check for the length of the validator pubkey. However, the value of this key is taken from the **ValidatorsExitBus._getValidatorData()** function in which the key is constructed strictly according to the length 48. This means that this check is unnecessary.

Recommendation

We recommend removing the extra check.

Client's comments

> Although higher up the call stack we construct a key with a fixed size, we prefer to keep an additional check at the module level. An arbitrary key length could theoretically pose issues in protocol configurations that differ from those described within the current scope. The extra check ensures that the module always receives a key of the correct length.

INFORMATIONAL-12	Triggerable exits do not increment TOTAL_REQUESTS_PROCESSED_POSITION	Fixed at: dc86473

Description

The **ValidatorsExitBusOracle** counts oracle processed requests in a **TOTAL_REQUESTS_PROCESSED_POSITION** variable, while **ValidatorsExitBus** does not have a similar counter.

Recommendation

We recommend adding a similar counter to the **ValidatorsExitBus** or expanding the existing one to count new exit flow requests.

Client's comments

This counter is intended to be used on the off-chain side as an indicator of the number of total exit requests. This way, off-chain bots can determine whether they have processed all the events within a given time frame, or if they need to fetch additional events in case this parameter has changed.
The fact that this parameter might be incremented twice due to the same exit request being received from different sources or even from the same source is considered a normal situation.

INFORMATIONAL-13	[CSM] Misleading comment	Fixed at: 7501c73

Description

Line: CSModule.sol#L1337
The existing comment says:

```
// Do not allow of multiple calls of addValidatorKeys* methods.
```

But the implementation only prevents multiple calls for the operator's creator, not for others. The comment is therefore inaccurate about the actual constraint.

Recommendation

We recommend changing the comment to the

```
-   // Do not allow of multiple calls of addValidatorKeys* methods.
+   // Do not allow of multiple calls of addValidatorKeys* methods for the creator contract.
```

INFORMATIONAL-14	[CSM] Missing sanity check in NOAddresses.changeNodeOperatorRewardAddress()	Fixed at: 7501c73

Description

Line: lib/NOAddresses.sol#L210
In all functions that propose or reset a variable to a new address, there is a check that prevents setting an address that is already set. However, **NOAddresses.changeNodeOperatorRewardAddress()** lacks this check. It is possible to call this function with **newAddress == no.rewardAddress** and not get any revert, unlike the other non-confirmation functions, which would reject redundant updates.

Recommendation

We recommend adding this sanity check to the function.

INFORMATIONAL-15	[CSM] Gas optimizations	Fixed at: 7501c73

Description

1. The storage variable reads in the **NOAddresses** contract can be optimized by storing values in the memory before checks, e.g.:
   In the **proposeNodeOperatorManagerAddressChange()** function, the **no.managerAddress** storage variable can be saved to the memory, and the **oldProposedAddress** declaration should be placed before the **AlreadyProposed()** revert. Same for **proposeNodeOperatorRewardAddressChange()**, **confirmNodeOperatorManagerAddressChange()**, **confirmNodeOperatorRewardAddressChange()**, **resetNodeOperatorManagerAddress()**, **changeNodeOperatorRewardAddress()**.
2. In the **CSModule.obtainDepositData()** function's for loop, the **no.depositableValidatorsCount** and **no.totalDepositedKeys** variables can be saved to memory.
3. In the **CSModule._addKeysAndUpdateDepositableValidatorsCount()** function, the **totalVettedKeys** variable can be saved to memory.

Recommendation

We recommend introducing these gas optimizations.

Client's comments

NOAddress - fixed CSModule.obtainDepositData() - not fixed due to stack overflow
CSModule._addKeysAndUpdateDepositableValidatorsCount() - fixed

INFORMATIONAL-16	[CSM] Keys removal logic can be optimized	Acknowledged

Description

Lines: SigningKeys.sol#L128-L142

The **SigningKeys.removeKeysSigs()** function performs batch removal of signing keys from storage by replacing each key to be removed with the last key in the array, then clearing the tail slot. However, when removing multiple keys in a single operation, the function may overwrite the same tail keys multiple times, leading to unnecessary sload/sstore operations. The issue appears when the number of remaining keys after the deletion range is less than the number of keys being deleted:

```
remainingKeys = totalKeysCount - (startIndex + keysCount)
keysToRemove = keysCount
remainingKeys < keysToRemove
```

This results in approximately **(keysToRemove – remainingKeys)** * **5** extra sload/sstore operations.

Illustrative Example

Consider a sequence of keys, represented here as an array:

[k1, k2, k3, k4, k5, k6]

We intend to remove keys k2–k4. Keys k5 and k6 will remain in the tail of the sequence.

- **totalKeysCount = 6**
- **startIndex = 1**
- **keysCount = 3**
- **keysToRemove = 3**
- **remainingKeys = 2**

Process states:

1. **[k1, k2, k3, k4, k5, k6]**
2. **[k1, k2, k3, k6, k5, 0]**
3. **[k1, k2, k5, k6, 0, 0]**
4. **[k1, k6, k5, 0, 0, 0]**

Key k6 was replaced one extra time compared to the optimal case.

Recommendation

We recommend calculating the final state size and moving the remaining tail keys in a single pass.

Client's comments

> We acknowledge that there is space for a minor optimisation here. However, given the rarity of the deletion operation, even higher rarity of the specific condition mentioned, planned migration to 0x02 WC validators in future releases, and the fact that the current code is also used in the other Lido staking modules, we prefer not to implement the proposed change.

INFORMATIONAL-17	[CSM] CSBondLock.MIN_BOND_LOCK_PERIOD can be zero	Fixed at: b80f5ce

Description

Lines:

- CSBondLock.sol#L57
- CSBondLock.sol#L133
- CSBondLock.sol#L97
- CSBondLock.sol#L80

The **CSBondLock** contract constructor allows setting the **MIN_BOND_LOCK_PERIOD** to zero. This, in turn, allows the **_setBondLockPeriod()** function to set the period to zero. This will cause new locks to be created with **bondLock.until** equal to the current **block.timestamp**. As a result, the **getActualLockedBond()** function will return zero, rendering the lock mechanism useless.

Recommendation

We recommend adding a zero-check for the **MIN_BOND_LOCK_PERIOD** variable.

INFORMATIONAL-18	[CSM] Missing NatSpec in constructor	Fixed at: b80f5ce

Description

Lines: CSAccounting.sol#L55-L59

In the NatSpec of the constructor, there is no description of the **_feeDistributor** parameter.

Recommendation

We recommend adding a comment describing this variable.

INFORMATIONAL-19	[CSM] Unclear settlement and compensation timings	Acknowledged

Description

The **CSBondLock** contract defines a **bondLockPeriod** during which a locked bond can be compensated or settled.
A node operator may compensate for the penalty at any time before it is settled. The **SETTLE_EL_REWARDS_STEALING_PENALTY_ROLE** role holder has exactly **bondLockPeriod** from lock start to settle the penalty. The settlement can occur immediately, even within the same block in the worst case (e.g., due to a misconfigured role holder).

Recommendation

We recommend introducing formal limitations to penalty settlement.

Client's comments

> The mechanism of penalty reporting and settling is separated into two distinct roles. In the real setup, the settle role is assigned to the Easy Track, which already features a 72-hour voting period.

INFORMATIONAL-20	[CSM] Following Checks-Effects-Interactions pattern when interacting with TriggerableWithdrawalsGateway	Fixed at: 2bc03e9

Description

Lines:

- CSStrikes.sol#L247
- CSEjector.sol#L220

When calling the **CSStrikes.processBadPerformanceProof()** method in **CSStrikes.sol**, first, an exit trigger for the operator with poor performance occurs via the **Gateway**, then a call to **CSExitPenalties.processStrikesReport()** on **EXIT_PENALTIES** follows. Between these calls, context is passed to the **Gateway** after the on-chain trigger and notification to the staking module. If an excess of ether is provided to pay for gas, a call will be made to the **recipient**. Thus, before the final state update of **EXIT_PENALTIES**, the context may be passed.

Recommendation

We recommend following the Checks-Effects-Interactions pattern and making all changes before passing the context to the external party.

INFORMATIONAL-21	[CSM] Sanity check for referralCurveId	Acknowledged

Description

Line: VettedGate.sol#L113

In the logic of the **VettedGate** contract, when the necessary conditions are met, the **referrer** can switch to **referralCurveId** by calling **VettedGate.claimReferrerBondCurve()**. There is also a public function **VettedGate.claimBondCurve()**, which allows a one-time switch to the preset **curveId** (upon inclusion in the Merkle tree). If **referralCurveId** and **curveId** coincide, the **referrer** will have no incentive to carry out their program, since in any case they can switch to the “bonus” **curveId**.

Recommendation

We recommend verifying that **referralCurveId** is not equal to **curveId** at the start of a new referral season.

Client's comments

Actors responsible for managing the referral seasons should verify the correctness of the parameters.

STATE
MIND