

STATE MIND

Lido V2

13-02-2023 – 28-04-2023

Table of contents



1. Project Brief		3
2. Finding Severity breakdown		5
3. Summary of findings		6
4. Conclusion		6
5. Findings report		7
Critical	Reentrancy in depositing buffered ether	7
	Funds stealing by malicious staking module	8
High	Not matching interface	9
	Wrong NFT balance	10
	Staking module can block withdrawal credential changing and oracle	11
	Zero deposits allocation after the pause the one of staking modules	12
	Withdrawal NFT burning after transfer to self	13
	Incorrect require leads to DOS	14
	A Node Operator can circumvent DAO validator key approval	15
The parameter depositCalldata can be hijacked by a front-run	16	

Medium	Blocking rewards distribution	17
	Wrong Burnt shares calculation	17
	reportProcessor can have a report that doesn't reach a consensus	18
	Wrong burntWithdrawalsShares calculation	19
	Possible to set stuck/exited validators count to be bigger than deposited	20
	Wrong offsets for stuck and refunded validators	20
	Possible to submit report extra data in the next frame after the report	21
	No check for return value size in isValidSignature()	22
	NodeOperatorsRegistry keys nonce doesn't change	23
	Restriction of total fee	24
	Manipulating active validators count	24
	Withdrawn stETH is not fully burnt	25
	FastLane length change leads to the disruption of the voting logic	26
	Quorum change leads to the disruption of the voting logic	27
	A single staking module can block whole deposits	28
	Deposits through DepositSecurityModule can be blocked by front-run	28
	Excessive penalty	29

Improper comments	29
Redundant onlyRole modifier	29
Array length check	34
Redundant v component check	30
Gas optimization for unlimited allowance	30
Guard checks for zero secondsPerSlot	30
Useless payable conversion for call	31
Stuck penalty delay extension	31
Hardcoded StakingModule limit	31
Code duplication	32
Gas optimization	59
Non-callable function	32
Roles mismatching	33
Withdrawal finalization before actual shares burn	33
Updating variables after external calls and transfers	33
Precise calculations	33
Active validators calculation	34
Payable owner	34
Transfer of instructions necessary for initialization	34
Wrong event argument	34
Division before multiplication	35
Multiple transfers of stETH token instead of a single transfer	35
No check on the refSlot in submitting a ConsensusReport	35
clearNodeOperatorPenalty() can be restricted to external	36
Unsafe memory handling in getStakingRewardsDistribution()	36
Request owner should not be payable	36

Informational	Inconsistent input string for role hash	36
	Incorrect check	36
	No zero check for _maxPositiveTokenRebase	37
	Unsafe casting	37
	Unsafe math	37
	Unused return value	38
	Exited validators count might be bigger than total deposited	38
	Missing checks for _requestId out of range	38
	Unused errors	38
	Unclear comments	38
	Missing zero check when reporting minted rewards	39
	Possible underflow in trimUint256Array()	39
	Typo in comment	39
	Typo in _addSigningKeys()	39
	Unable to get a consensus state if the quorum is unreachable	40
	Insufficient if statement in WithdrawalQueueBase contract	40
	Lack of zero address check	41
	State mutability can be restricted to view	41
	Wrong comment in DepositSecurityModule	41
	State mutability	42
	Modifier doesn't guarantee anything, default enum status	42
	Check arrays length	42
	Binary search boundary	42
	Unnecessary variable	42
	Penalty delay type	43
	Unclear purpose of limits in Sanity Checker	43

Type mismatch	43
Incorrect information in the comment	44
Consecutively emitting identical events	44
"Bunker mode" withdrawal implementation	44
Uncertain reward withdrawal	45
Prefinalize SLOAD reduction	45
Redundant type casting	46
Redundant checkpoint check	46
Claim SLOAD reduction	47
Avoiding heavy computations	47
Affecting on burn request	48
Range restriction	48
Narrow range	48
Depositable ether zero check	49
Active node operator check	49
Limits for max deposits	49
No appropriate event	49
Possible overflow	50
No post deposit root check	50
Stuck validators update	51
_checkAnnualBalancesIncrease() doesn't include withdrawn ether	52
SanityChecker includes withdrawn ether from past reports	53
Unused function parameters	53
No checks for node operator id and validator index	54
No check for _etherToLockOnWithdrawalQueue	54
Missing length check in loadKeysSigs()	54

No checks for a different values in the setter functions	55
StakingRouter can return exited validators that hasn't been updated	55
Possible risk of DOS via ETH sending into Consensus Layer	56
Possibility to deposit the same validator address	57
Extra if statement in _checkConsensus method	57
Impossible to set reportProcessor if the initial epoch is in the future	58
DEADLINE_SLOT_OFFSET	58
Soft requirement for transfer	58
Gas optimization: MinAllocation	59

1. Project Brief



Title	Description
Client	Lido
Project name	Lido V2
Timeline	13-02-2023 - 28-04-2023
Initial commit	89ad4cc35407609081afb94df535d71b27bb44b7
Final commit	e45c4d6fb8120fd29426b8d969c19d8a798ca974

Short Overview


The Lido V2 protocol upgrade for Lido on Ethereum allows stETH token redemptions to native ether using Ethereum withdrawals introduced with the Shanghai/Capella hardfork.

The major features of the protocol upgrade beyond withdrawals support:

- A modular smart contracts architecture that unlocks the heterogenous Lido validators subsets onboarding represented as independent staking modules (e.g., community-driven validators or committees of DVT-enabled validators) for the StakingRouter contract.
- An updated oracle contract consensus mechanics that allows delivering huge data chunks (virtually unbounded) via a two-step procedure: reach a consensus about the data hash, and provide the data itself in a batched manner.


Project Scope

The audit covered the following files:


-  [SignatureUtils.sol](#)


 [ECDSA.sol](#)


 [MinFirstAllocationStrategy.sol](#)


 [LegacyOracle.sol](#)


 [Lido.sol](#)

 [Versioned.sol](#)

 [SigningKeys.sol](#)


 [Packed64x4.sol](#)


 [OssifiableProxy.sol](#)

 [OracleDaemonConfig.sol](#)


 [WithdrawalQueueBase.sol](#)


 [ValidatorsExitBusOracle.sol](#)

 [HashConsensus.sol](#)

 [WithdrawalQueue.sol](#)


 [AccessControlEnumerable.sol](#)

 [Versioned.sol](#)


 [UnstructuredRefStorage.sol](#)

 [Math.sol](#)

 [LidoExecutionLayerRewardsVault.sol](#)


 [WithdrawalQueueERC721.sol](#)

 [StakingRouter.sol](#)
-  [Math256.sol](#)

 [MemUtils.sol](#)

 [NodeOperatorsRegistry.sol](#)

 [StETH.sol](#)


 [Pausable.sol](#)

 [StETHPermit.sol](#)


 [StakeLimitUtils.sol](#)

 [WstETH.sol](#)


 [WithdrawalVault.sol](#)


 [Burner.sol](#)


 [AccountingOracle.sol](#)


 [BaseOracle.sol](#)

 [OracleReportSanityChecker.sol](#)


 [PausableUntil.sol](#)

 [AccessControl.sol](#)


 [DepositSecurityModule.sol](#)

 [UnstructuredStorage.sol](#)

 [PositiveTokenRebaseLimiter.sol](#)

 [EIP712StETH.sol](#)

 [BeaconChainDepositor.sol](#)

 [LidoLocator.sol](#)

2. Finding Severity breakdown



All vulnerabilities discovered during the audit are classified based on its potential severity and has 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 Customer 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 Customer 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	2 (1 fixed, 1 acknowledged)
High	8 (6 fixed, 2 acknowledged)
Medium	17 (9 fixed, 8 acknowledged)
Informational	93 (59 fixed, 34 acknowledged)
Total	120 (75 fixed, 45 acknowledged)

4. Conclusion



During the audit of Lido V2 codebase, 120 issues were found in total:

- 2 critical severity issues (1 fixed, 1 acknowledged)
- 8 high severity issues (6 fixed, 2 acknowledged)
- 17 medium severity issues (9 fixed, 8 acknowledged)
- 93 informational severity issues (59 fixed, 34 acknowledged)

The final reviewed commit is [e45c4d6fb8120fd29426b8d969c19d8a798ca974](#)

5. Findings report



CRITICAL-01

Reentrancy in depositing buffered ether

Fixed at [8ac425](#)

Description

In `DepositSecurityModule.depositBufferedEther` method different conditions are checked. However, in the case of a malicious `StakingModule` it is possible to call the method multiple times and still pass all the checks in the same transaction. As a result, depositing more buffered ether than `maxDepositsPerBlock` and increasing `targetShares` of the `StakingModule`. Approximate scheme of attack:

1. `depositBufferedEther` is called with a valid signature and calldata.
2. `StakingModule` reverts the transaction in `obtainDepositData` method.
3. Right after that, an attacker sends a transaction with the same calldata. In case no deposits were made to the `DepositContract` between these 2 transactions, the `onchainDepositRoot` does not change, like other signed data. As a result, all the checks in `DepositSecurityModule.depositBufferedEther` are passed.
4. When execution reaches `obtainDepositData` of the malicious `StakingModule` it does not revert but calls `DepositSecurityModule.depositBufferedEther` again. Thus `onchainDepositRoot` does not change and checks in `DepositSecurityModule.depositBufferedEther` are passed.
5. Repeat the previous step multiple times.

```
(bytes memory publicKeysBatch, bytes memory signaturesBatch) =  
    IStakingModule(stakingModule.stakingModuleAddress)  
        .obtainDepositData(_depositsCount, _depositCalldata); // <-- reentrancy here
```

6. After that, all reentrancy calls process through `_makeBeaconChainDeposits32ETH` and the following assert statement.

Recommendation

We recommend adding `nonces` to `DepositSecurityModule` that will change respectively to prevent reentrancy and possible replay of transactions.

Client's comments

Note on staking module trust assumptions: The finding presumes that the staking module can be 'malicious'.

The term is not accurate practically because each registered staking module has to be treated as a trusted part of the protocol, i.e.:

- passing the review of the 'Lido on Ethereum' protocol contributors team
- passing the audit assessments and deployed code verification with public audit report(s)
- passing the explicit approval via the Lido DAO governance process (proposal, discussion, snapshot, and Aragon)

Based on the described process methodology and enforcements, we assume that each staking module listed in the `StakingRouter` contract behaves ethically and follows integrity constraints.

According to the the note on staking module trust assumptions, modules with the described functionality can't be registered in the `StakingRouter` contract. However, following the recommendations in the finding "Updating variables after external calls and transfers", the order of the local state mutations and calls to external contracts was changed to prevent reentrancy.

Description

Malicious staking modules can put their own controlled withdrawal credentials for the deposits upcoming from Lido. In the function **deposit** there is an external call to **StakingModule** contract that returns public keys and signatures for the deposit. During this call malicious **StakingModule** can pre-deposit 1 ETH to these keys with its own controlled withdrawal credentials, this action leads to obtaining the control over 32 ETH deposited by Lido.

```
(bytes memory publicKeysBatch, bytes memory signaturesBatch) =
    IStakingModule(stakingModule.stakingModuleAddress)
        .obtainDepositData(_depositsCount, _depositCalldata);
//      ^-- perform deposit with malicious withdrawal credentials

uint256 etherBalanceBeforeDeposits = address(this).balance;
_makeBeaconChainDeposits32ETH(
    _depositsCount,
    abi.encodePacked(withdrawalCredentials),
    publicKeysBatch,
    signaturesBatch
);
// ^-- Lido puts deposits after a malicious withdrawal credentials are set
uint256 etherBalanceAfterDeposits = address(this).balance;
```

Recommendation

We recommend checking **deposit_root** right before deposit

Client's comments

According to the note on staking module trust assumptions (see the CRIT-01 client's comment), modules with the described functionality can't be registered in the StakingRouter contract. Moreover, we believe that the described attack can't proceed without a deposit signature change. In case of a deposit with the same signature and different withdrawal credentials and amount, the deposit will not pass on the beacon chain: see the process_deposit method in the beacon chain spec.

Description

OracleReportSanityChecker has the **IWithdrawalQueue** interface with the **getWithdrawalRequestStatus** function defined in it.

```
interface IWithdrawalQueue {  
  function getWithdrawalRequestStatus(uint256 _requestId)  
    external  
    view  
    returns (  
      uint256 amountOfStETH,  
      uint256 amountOfShares,  
      address recipient,  
      uint256 timestamp,  
      bool isFinalized,  
      bool isClaimed  
    );  
}
```

OracleReportSanityChecker calls **WithdrawalQueue** in internal **_checkRequestIdToFinalizeUpTo**, which is part of an oracle report flow.

```
(, , , uint256 requestTimestampToFinalizeUpTo, , ) = IWithdrawalQueue(_withdrawalQueue)  
  .getWithdrawalRequestStatus(_requestIdToFinalizeUpTo); // <-- function does not exist
```

At the same time, none of the contracts associated with **WithdrawalQueue** contain such a method.

```
function getWithdrawalStatus(uint256[] calldata _requestIds) // <-- WithdrawalQueue's function that should've been called  
  external  
  view  
  returns (WithdrawalRequestStatus[] memory statuses)
```

The **getWithdrawalRequestStatus** function were renamed in the commit [be5fbd70e95e34bb9911eccdae57f4f6b5a8c1c9](#)

Recommendation

We recommend changing **getWithdrawalRequestStatus** to **getWithdrawalStatus**.

HIGH-02	Wrong NFT balance	Fixed at bdb8f7
---------	-------------------	---------------------------------

Description

In the contract **WithdrawalRequestNFT**, the function **balanceOf** returns a user's balance, including claimed NFTs that the contract has already burned. This error may need to be clarified for users or, in some cases, break third-party contracts. These contracts will suppose that this function returns the actual balance of a user. This error may break the math of those contracts.

```
function balanceOf(address _owner) external view override returns (uint256) {
    if (_owner == address(0)) revert InvalidOwnerAddress(_owner);
    return _getRequestsByOwner()[_owner].length();
    //      ^-- includes already claimed
}
```

Recommendation

In the contract **WithdrawalQueue** in the functions [claimWithdrawals](#), [claimWithdrawalTo](#), and [claimWithdrawal](#), add this code:

```
_getRequestsByOwner()[msg.sender].remove(_requestId);
```

Description

In the function **setWithdrawalCredentials** there is a loop through all staking modules which notifies each module:

```
WITHDRAWAL_CREDENTIALS_POSITION.setStorageBytes32(_withdrawalCredentials);

uint256 stakingModulesCount = getStakingModulesCount();
for (uint256 i; i < stakingModulesCount; ) { // <-- loop through all staking modules
    IStakingModule(_getStakingModuleAddressByIndex(i)).onWithdrawalCredentialsChanged();
    // ^-- notification callback
    unchecked {
        ++i;
    }
}

emit WithdrawalCredentialsSet(_withdrawalCredentials, msg.sender);
```

However, if one of the staking modules is broken or malicious and triggers revert on **onWithdrawalCredentialsChanged** call whole **setWithdrawalCredentials** will be reverted. This issue can be fixed only by updating **StakingRouter** since there is no functionality to remove or omit reverting staking modules.

The same behavior is also found in the following place:

- In the function **onValidatorsCountsByNodeOperatorReportingFinished**

Recommendation

We recommend wrapping the callbacks with a **try-catch** and omitting reverting modules

Description

Currently, the function `_getDepositsAllocation` is responsible for deposits allocation across staking modules according to target shares and capacities. The allocation mechanic is pretty simple:

- Firstly, need to calculate **targetValidators** based on **targetShare** and **totalActiveValidators**
- Then use some strategy to allocate new deposits across staking modules based on **allocations** and **capacities**

```
function _getDepositsAllocation(
  uint256 _depositsToAllocate
) internal view returns (
  uint256 allocated, uint256[] memory allocations, StakingModuleCache[] memory stakingModulesCache) {
  // calculate total used validators for operators
  uint256 totalActiveValidators;

  (totalActiveValidators, stakingModulesCache) = _loadStakingModulesCache(true);

  uint256 stakingModulesCount = stakingModulesCache.length;
  allocations = new uint256[](stakingModulesCount);
  if (stakingModulesCount > 0) {
    /// @dev new estimated active validators count
    totalActiveValidators += _depositsToAllocate;
    uint256[] memory capacities = new uint256[](stakingModulesCount);
    uint256 targetValidators;

    for (uint256 i; i < stakingModulesCount; ) {
      allocations[i] = stakingModulesCache[i].activeValidatorsCount;
      targetValidators = (stakingModulesCache[i].targetShare * totalActiveValidators) / TOTAL_BASIS_POINTS;
      capacities[i] = Math256.min(targetValidators,
        stakingModulesCache[i].activeValidatorsCount + stakingModulesCache[i].availableValidatorsCount);
      unchecked {
        ++i;
      }
    }

    allocated = MinFirstAllocationStrategy.allocate(allocations, capacities, _depositsToAllocate);
  }
}
```

The allocation strategy that is used currently finds staking modules whose **allocation** is below calculated capacity and fills them starting by most "unfilled", otherwise if staking module allocation is above target it will not receive any new allocation. However, if we have several staking modules and one of them becomes paused it's possible that **allocation** for each validator becomes above **capacities** since **capacities = min(targetValidators, activeValidatorsCount + availableValidatorsCount)**. That happens because **totalActiveValidators** become lower than it was before pause and **targetShare** keeps the same value which leads to lower **targetValidators** value and **capacity** also becomes lower since **capacity = min(targetValidators, activeValidatorsCount + availableValidatorsCount)**

Recommendation

We recommend compensating target shares for active modules when staking module paused.

HIGH-05	Withdrawal NFT burning after transfer to self	Fixed at 8073c3
---------	---	---------------------------------

Description

Each withdrawal request is an NFT token, this NFT can be transferred to another address by the owner. However, if the owner transfers NFT to self the **balanceOf** function will return **0** due to:

```
function _transfer(address _from, address _to, uint256 _requestId) internal {
    if (_from == address(0)) revert TransferFromZeroAddress();
    if (_to == address(0)) revert TransferToZeroAddress();
    if (_requestId == 0 || _requestId > getLastRequestId()) revert InvalidRequestId(_requestId);

    WithdrawalRequest storage request = _getQueue()[_requestId];

    if (_from != request.owner) revert TransferFromIncorrectOwner(_from, request.owner);
    if (request.claimed) revert RequestAlreadyClaimed(_requestId);

    delete _getTokenApprovals()[_requestId];
    request.owner = payable(_to);

    _getRequestsByOwner()[_to].add(_requestId);
    // ^-- add `_requestId` to self (nothing happens, `add` method just returns `false`)
    _getRequestsByOwner()[_from].remove(_requestId);
    // ^-- remove `_requestId` from self, after that self balance decremented by 1
}
```

Link to the code: [WithdrawalRequestNFT.sol#L179-L180](#)

Recommendation

We recommend checking that **_to** is not equal to **_from**.

Description

In the function `AccountingOracle._processStakingRouterExitedValidatorsByModule` there is a check that number of exited validators can't be lower than the number from the previous report.

```
uint256 exitedValidators = 0;
for (uint256 i = 0; i < stakingModuleIds.length;) { // <-- loop through staking modules with newly exited validators
    if (numExitedValidatorsByStakingModule[i] == 0) {
        revert InvalidExitedValidatorsData();
    } else {
        exitedValidators += numExitedValidatorsByStakingModule[i];
    }
    unchecked { ++i; }
}

uint256 prevExitedValidators = stakingRouter.getExitedValidatorsCountAcrossAllModules();
//                               ^-- number of exited validators across all modules
if (exitedValidators < prevExitedValidators) { // <-- will almost always be true
    revert NumExitedValidatorsCannotDecrease();
}
```

However, this comparison is made on different sets of staking modules. In the first case, it is staking modules with newly number of exited validators (`_processStakingRouterExitedValidatorsByModule` takes **stakingModuleIdsWithNewlyExitedValidators** as argument) and in the second case, it is all staking modules. Thus, this check will almost always fail.

Recommendation

We recommend removing this check or making comparison of exited validator numbers for the same set of validators.

Description

This issue follows from a prior issue <https://github.com/lidofinance/lido-dao/issues/141> but with different steps:

Adding validator keys is performed as follows:

1. A node operator appends new keys to its set; the node operator's staking limit stays the same, making the added keys unusable yet.
2. The DAO performs off-chain verification of the submitted keys: it checks that withdrawal credentials, deposit amounts, and signatures are correct and that there are no duplicate keys in the updated key set.
3. The DAO votes to increase the node operator's limit to allow the added keys to be used by the pool.

However, a node operator can currently add keys that were not approved by the DAO using a simple trick:

1. A node operator submits **N** correct keys and then enacts an Easy Track motion to increase the staking limit.
2. The DAO validates the keys and votes for the motion with **stakingLimit = N**.
3. After the motion is passed the node operator submits **M ≤ N** more (bad) keys and removes the first **M** keys.
4. The node operator can enact the motion or front run the transaction enacting the motion.
5. The motion is enacted and sets the **stakingLimit = N** and the newly-added keys are allowed to be used in staking despite not being approved by the DAO.

Recommendation

A possible solution is to save a different nonce for each node operator and vote for motion with expected nonce, then check if the nonces match when enacting the motion.

Client's comments

At the launch of the StakingRouter, it will be used only with the Curated staking module (NodeOperatorsRegistry). All node operators registered in the NodeOperatorsRegistry pass strict selection and provide the best service on the market. For professional node operators, such an attack will lead to reputational damage in the short term and financial losses in the long term. But even though the opportunity of the described attack is very low, Lido makes the below action to minimize the damage:

- Constantly monitoring offchain uploaded/deposited keys by their node operators. If the described attack happens, Lido's alerting system will allow it to react fastly and pause deposits via **DepositSecurityModule** until the malicious node operator will be deactivated.
- the **DepositSecurityModule** has the upper bound for the number of deposits made in one transaction (**maxDepositsPerBlock**) and the cooldown between deposits for the same staking module. It gives enough time to react to pause deposits in response to the malicious actions of node operators. The fix for the highlighted issue is planned at the next update of the protocol and definitely will be eliminated before the introduction of a new staking module with a lower trust level for the node operators.

HIGH-08	The parameter <code>depositCalldata</code> can be hijacked by a front-run	Acknowledged
---------	---	--------------

Description

In the function `depositBufferedEther` the `depositCalldata` parameter can be hijacked by a malicious actor by front-running the initial transaction.

The function receives `sortedGuardianSignatures` parameter that includes signatures of digest from `depositRoot`, `blockNumber`, `blockHash`, `stakingModuleId`, `nonce` parameters but doesn't include `depositCalldata` to the digest, so an attacker can front-run initial transaction with changed `depositCalldata` parameter. That can lead to wrong call interpretation on the staking module side.

```
_verifySignatures(depositRoot, blockNumber, blockHash, stakingModuleId, nonce, sortedGuardianSignatures);
// ^-- this method doesn't include `depositCalldata` to the digest
//    so signatures are valid for any `depositCalldata`

LIDO.deposit(maxDepositsPerBlock, stakingModuleId, depositCalldata);
```

Recommendation

We recommend adding `depositCalldata` to the digest

Client's comments

The `depositCalldata` argument is considered as the data that might be required to prepare the deposit data by the staking module.

Depending on the staking module implementation, it might be empty or contain complete deposit data (e.g. if off-chain storage is used for the deposit data). Anyway, it's the responsibility of the staking module to validate the correctness of the passed data when the module relies on it.

For any staking module, the following is ALWAYS true: passed `depositCalldata` MUST NOT affect the deposit data set of the staking module (This requirement was explicitly added in the comments of `IStakingModule.obtainDepositData()` method in the commit <https://github.com/lidofinance/lido-dao/commit/4c80206e016967989bb948ce0c42b9a009edf88b>). In other words, call of `StakingModule.obtainDepositData()` with different valid `depositCalldata` for the same `StakingModule.nonce` leads to the same unambiguous set of deposited validators (e.g. the offchain staking module might store a hash of the expected `depositCalldata` onchain to validate it and reject any call with unexpected `depositCalldata`).

MEDIUM-01	Blocking rewards distribution	Fixed at 1c0cc4
<p>Description</p> <p>In function <code>reportRewardsMinted()</code>, there is a callback to StakingModule.</p> <pre>for (uint256 i = 0; i < _stakingModuleIds.length;) { // <-- loop through all staking modules address moduleAddr = _getStakingModuleById(_stakingModuleIds[i]).stakingModuleAddress; IStakingModule(moduleAddr).onRewardsMinted(_totalShares[i]); // ^-- callback on module to report minted rewards unchecked { ++i; } }</pre> <p>In case of a malicious module it can block and reward distribution to all modules, it also affects the whole AccountingOracle report.</p> <p>Recommendation</p> <p>It is recommended to wrap the callback with a try-catch and omit reverting modules.</p>		

MEDIUM-02	Wrong Burnt shares calculation	Fixed at 117b0f
<p>Description</p> <p>While handling the oracle report in Lido, function <code>_burnSharesLimited()</code> returns a number of burnt shares, but it does not include postponed burnt shares.</p> <p>E.g.</p> <pre>// Before burn request: coverSharesBurnRequested = 100; nonCoverSharesBurnRequested = 300; // After burn request of 1000 shares: coverSharesBurnRequested = 100; nonCoverSharesBurnRequested = 300 + 1000; // After burn commit with sharesToBurnLimit = 1100: coverSharesBurnRequested = 0; sharesToBurnLimit = 1000; nonCoverSharesBurnRequested = 1300 - 1000 = 300; sharesToBurnLimit = 0; sharesToBurnNow = 1100; postponedSharesToBurn = 0 + 300 = 300; burntWithdrawalsShares = 1000 - 300 = 700; // but actually 1000 was burnt</pre> <p>Recommendation</p> <p>It is recommended to leave a comment if it is the intended behavior or change function to return actual number of burnt shares because it is used for further sanity checks.</p> <p>Client's comments</p> <p>Shares burning was rewritten to make stETH rebases more predictable.</p>		

Description

In the contract **HashConsensus**, submitted reports have a floating number of supports. It can increase and decrease. It decreases in case the admin removes some member, or some member offers a report that differs from his previous report. The contract doesn't handle an edge case when the support of a report decreases. For example, some reports can have the number of support equal to the quorum. In this case, if some member that supported the report removed or started to support another report, it will no longer have quorum support and becomes invalid for the **reportProcessor** contract, but the **reportProcessor** contract won't find out about it and will process this report.

Another case is when the **DISABLE_CONSENSUS_ROLE** role sets the **UNREACHABLE_QUORUM** value. If some report has reached a quorum before this set, the **reportProcessor** contract will have some report that can be processed, even if it is forbidden to make a consensus.

```
function _submitReport(uint256 slot, bytes32 report, uint256 consensusVersion) internal {
    ...
    if (slot == memberState.lastReportRefSlot) {
        ...
        if (varIndex == prevVarIndex) {...} else {
            // the function doesn't handle the case when
            // `varIndex != prevVarIndex` and consensus was reached
            --_reportVariants[prevVarIndex].support;
        }
    }
    ...
}
...
function _checkConsensus(uint256 quorum) internal {
    ...
    if (consensusVariantIndex >= 0) {
        _consensusReached(frame, consensusReport, uint256(consensusVariantIndex), support);
    }
    // the function doesn't handle the case when consensus not reached
}
```

Recommendation

Add to the interface **IReportAsyncProcessor** a new function that will delete a submitted report in case this report doesn't have enough support, or it is forbidden to make a consensus.

Description

The simulated share rate is calculated by simulating an **eth_call** with no finalization of withdrawals. The burner contract can have postponed shares to burn from previous oracle reports.

The function **_burnSharesLimited()**:

```
if (_sharesToBurnLimit > 0) {
    uint256 sharesCommittedToBurnNow = _burner.commitSharesToBurn(_sharesToBurnLimit);

    if (sharesCommittedToBurnNow > 0) {
        _burnShares(address(_burner), sharesCommittedToBurnNow);
    }
}

(uint256 coverShares, uint256 nonCoverShares) = _burner.getSharesRequestedToBurn();
uint256 postponedSharesToBurn = coverShares.add(nonCoverShares);

burntCurrentWithdrawalShares =
    postponedSharesToBurn < _sharesToBurnFromWithdrawalQueue ?
    _sharesToBurnFromWithdrawalQueue - postponedSharesToBurn : 0;
```

In the real call **sharesToBurnLimit** increases compared to the simulated call. This can lead to burning of postponed shares in the real call which was not predicted in the simulated call, but **burntWithdrawalsShares = 0** in both calls.

Consider a case:

```
// Before burn request:
coverSharesBurnRequested = 1000;
nonCoverSharesBurnRequested = 3000;
// Simulated call:
sharesToBurnLimit = 1000;
// After burn commit with sharesToBurnLimit = 1000:
coverSharesBurnRequested = 0;
nonCoverSharesBurnRequested = 3000;
sharesToBurnNow = 1000;

// Real call:
sharesToBurnLimit = 3000;
// After burn request of 1000 shares:
coverSharesBurnRequested = 1000;
nonCoverSharesBurnRequested = 3000 + 1000;
// After burn commit with sharesToBurnLimit = 3000:
coverSharesBurnRequested = 0;
nonCoverSharesBurnRequested = 2000;
sharesToBurnNow = 3000;
postponedSharesToBurn = 2000;
burntWithdrawalsShares = 0;
// but actually 1000 cover and 2000 non cover shares were burnt in a real call
// compared to 1000 cover shares in a simulated call
```

This can lead to reverts when the simulated share rate is checked by the sanity checker contract.

Recommendation

We recommend accounting for **burntWithdrawalsShares** computed during the simulation call.

MEDIUM-05	Possible to set stuck/exited validators count to be bigger than deposited	Fixed at e4d0bf
-----------	---	---------------------------------

Description

The oracle extra data report sets the stuck validators for staking module before setting exited validators count at the lines [AccountingOracle.sol#L854-L860](#). This leads to a situation where it is possible to stuck and exited validators count to be bigger than deposited for a staking module due to a wrong check.

At the line [NodeOperatorsRegistry._updateStuckValidatorsCount\(\)](#), `_stuckValidatorsCount` is checked with a value exited count not yet updated.

```

    _requireValidRange(
        _stuckValidatorsCount
        <= signingKeysStats.get(DEPOSITED_KEYS_COUNT_OFFSET)
        - signingKeysStats.get(EXITED_KEYS_COUNT_OFFSET)
    );

```

Then during [NodeOperatorsRegistry._updateExitedValidatorsCount\(\)](#), `_exitedValidatorsKeysCount` is only checked to be smaller than deposited count without taking into consideration the stuck amount.

```

    _requireValidRange(_exitedValidatorsKeysCount <= signingKeysStats.get(DEPOSITED_KEYS_COUNT_OFFSET));

```

Recommendation

We recommend fixing the check in [NodeOperatorsRegistry._updateExitedValidatorsCount\(\)](#) to such that exited and stuck counts in total cannot exceed deposited.

MEDIUM-06	Wrong offsets for stuck and refunded validators	Fixed at e470fc
-----------	---	---------------------------------

Description

At the line [NodeOperatorsRegistry.sol#L614](#), `STUCK_VALIDATORS_COUNT_OFFSET` should be used.

```

uint64 curStuckValidatorsCount = stuckPenaltyStats.get(REFUNDED_VALIDATORS_COUNT_OFFSET);

```

At the line [NodeOperatorsRegistry.sol#L623](#), `REFUNDED_VALIDATORS_COUNT_OFFSET` should be used.

```

uint64 curRefundedValidatorsCount = stuckPenaltyStats.get(STUCK_VALIDATORS_COUNT_OFFSET);

```

Recommendation

We recommend using correct offsets.

Description

When submitting extra data, the function `_checkCanSubmitExtraData()` checks the processing deadline `_checkProcessingDeadline()`.

```
function _checkCanSubmitExtraData(ExtraDataProcessingState memory procState, uint256 format)
    internal view
{
    _checkMsgSenderIsAllowedToSubmitData();
    _checkProcessingDeadline(); // missing check for extra data ref slot

    if (procState.refSlot != LAST_PROCESSING_REF_SLOT_POSITION.getStorageUint256()) {
        revert CannotSubmitExtraDataBeforeMainData();
    }

    if (procState.dataFormat != format) {
        revert UnexpectedExtraDataFormat(procState.dataFormat, format);
    }
}
```

But it is possible to submit report extra data in the next frame after the main report in a certain scenario:

1. The main report data is submitted for the current frame.
2. The extra data is not submitted for the current frame.
3. In the next frame, the consensus contract submits a report which updates the `_storageConsensusReport()` value and emits the event **WarnExtraDataIncompleteProcessing**.
4. The report's extra data for the previous frame can be submitted before the processing of the new report.

Recommendation

We recommend checking the extra data ref slot with the current ref slot when submitting extra data.

MEDIUM-08	No check for return value size in isValidSignature()	Fixed at 2dbad6
-----------	--	---------------------------------

Description

At the line [SignatureUtils.sol#L41-L53](#):

```
bytes4 retval;
/// @solidity memory-safe-assembly
assembly {
    // allocate memory for storing the return value
    let outDataOffset := mload(0x40)
    mstore(0x40, add(outDataOffset, 32))
    // issue a static call and load the result if the call succeeded
    let success := staticcall(gas(), signer, add(data, 32), mload(data), outDataOffset, 32)
    if eq(success, 1) {
        retval := mload(outDataOffset)
    }
}
return retval == ERC1271_IS_VALID_SIGNATURE_SELECTOR;
```

retval takes the first 4 bytes of the return value without checking the size of the return value. If the **staticcall** returns more than 4 bytes with the first 4 bytes equalling **ERC1271_IS_VALID_SIGNATURE_SELECTOR**, then the function **isValidSignature()** will return **true**.

Recommendation

We recommend checking if the return data size is 4 bytes.

Description

The keys nonce should change when a node operator's ready-to-deposit data size is changed. The function `_applyNodeOperatorLimits()` can change ready-to-deposit data size:

```
function _applyNodeOperatorLimits(uint256 _nodeOperatorId) internal returns (int64 maxSigningKeysDelta) {
    Packed64x4.Packed memory signingKeysStats = _loadOperatorSigningKeysStats(_nodeOperatorId);
    Packed64x4.Packed memory operatorTargetStats = _loadOperatorTargetValidatorsStats(_nodeOperatorId);

    uint64 exitedSigningKeysCount = signingKeysStats.get(EXITED_KEYS_COUNT_OFFSET);
    uint64 depositedSigningKeysCount = signingKeysStats.get(DEPOSITED_KEYS_COUNT_OFFSET);
    uint64 vettedSigningKeysCount = signingKeysStats.get(VETTED_KEYS_COUNT_OFFSET);

    uint64 oldMaxSigningKeysCount = operatorTargetStats.get(MAX_VALIDATORS_COUNT_OFFSET);
    uint64 newMaxSigningKeysCount = depositedSigningKeysCount;

    if (isOperatorPenaltyCleared(_nodeOperatorId)) {
        if (operatorTargetStats.get(IS_TARGET_LIMIT_ACTIVE_OFFSET) == 0) {
            newMaxSigningKeysCount = vettedSigningKeysCount;
        } else {
            // correct max count according to target if target is enabled
            // targetLimit is limited to UINT64_MAX
            uint256 targetLimit = Math256.min(
                uint256(exitedSigningKeysCount).add(operatorTargetStats.get(TARGET_VALIDATORS_COUNT_OFFSET)),
                UINT64_MAX
            );
            if (targetLimit > depositedSigningKeysCount) {
                newMaxSigningKeysCount = uint64(Math256.min(vettedSigningKeysCount, targetLimit));
            }
        }
    } // else newMaxSigningKeysCount = depositedSigningKeysCount, so depositable keys count = 0

    if (oldMaxSigningKeysCount != newMaxSigningKeysCount) {
        operatorTargetStats.set(MAX_VALIDATORS_COUNT_OFFSET, newMaxSigningKeysCount);
        _saveOperatorTargetValidatorsStats(_nodeOperatorId, operatorTargetStats);
        maxSigningKeysDelta = int64(newMaxSigningKeysCount) - int64(oldMaxSigningKeysCount);
    }
}
```

However, there are cases when the nonce doesn't change:

- `updateStuckValidatorsCount()`, if the operator is penalized its depositable keys size is set to **0**
- `updateExitedValidatorsCount()`, when the exited count is updated the target limit changes which can change the depositable keys size
- `unsafeUpdateValidatorsCount()`, follows from above
- `updateTargetValidatorsLimits()` changes the target limit
- `clearNodeOperatorPenalty()` clears the penalty and changes depositable keys size

Recommendation

We recommend changing the nonce in these functions.

MEDIUM-10	Restriction of total fee	Acknowledged
-----------	--------------------------	--------------

Description

At Line 904 **totalFee** is checked if it is less than 100 percent, but it is not enough, because **totalFee** could 99,999% and almost all profit will go to node operators and treasury.

```

...

if (totalFee >= precisionPoints) revert ValueOver100Percent("totalFee");
//  ^-- here totalFee is checked if it is over 100 percent
//      but there is no other restrictions on it

...

```

Recommendation

It is recommended to add a limit variable controlled by DAO, to restrict **totalFee**.

Client's comments

Management rights of all staking module settings (including **stakingModuleFee**) will be laid to Lido DAO, and it's the responsibility of the DAO to control that the total fee is set correctly for each module.

MEDIUM-11	Manipulating active validators count	Acknowledged
-----------	--------------------------------------	--------------

Description

In function _loadStakingModulesCacheItem() there is an external call to **StakingModule** to get a number of total deposited, exited and available validators. Based on these variables the number of active validators is calculated.

```

uint256 totalExitedValidators;
uint256 totalDepositedValidators;
(totalExitedValidators, totalDepositedValidators, cacheItem.availableValidatorsCount)
    = IStakingModule(cacheItem.stakingModuleAddress).getStakingModuleSummary();
//      ^-- malicious staking module could return wrong data

// the module might not receive all exited validators data yet => we need to replacing
// the exitedValidatorsCount with the one that the staking router is aware of
cacheItem.activeValidatorsCount =
    totalDepositedValidators -
    Math256.max(totalExitedValidators, stakingModuleData.exitedValidatorsCount);

```

But in case of malicious **StakingModule**, it can falsify values to get more shares in allocations and rewards distributions.

Recommendation

It is recommended to store important variables in **StakingRouter** storage and update them in it (e.g. the number of total deposited validators can be stored and calculated in **StakingRouter** alongside with **exitedValidatorsCount**).

Client's comments

According to the requirements for the staking modules provided above, a module with described functionality can't be registered in the StakingRouter.

MEDIUM-12	Withdrawn stETH is not fully burnt	Acknowledged
-----------	------------------------------------	--------------

Description

One of the function's `smoothenTokenRebase()` returns is the limit of burnt shares. Due to consuming limits of positive rebase and discount of locked ether it is possible that this limit will be lower than the amount of shares to be burnt.

```

withdrawals = tokenRebaseLimiter.consumeLimit(_withdrawalVaultBalance);
elRewards = tokenRebaseLimiter.consumeLimit(_elRewardsVaultBalance);
tokenRebaseLimiter.raiseLimit(_etherToLockForWithdrawals);

sharesToBurnLimit = tokenRebaseLimiter.getSharesToBurnLimit();
//  ^-- due to consuming limits burn limit could be lower than
//      actual amount of shares to be burnt from Withdrawal Queue

```

Math equations are available via [link](#).

Recommendation

It is recommended to leave a comment if it is considered that there may be non-cover postponed shares because there is a comment before the actual burn that says excess shares will be burnt (withdrawn stETH at least).

Client's comments

Shares burning was rewritten to make stETH rebases more predictable. commit: <https://github.com/lidofinance/lido-dao/commit/117b0fa33f76e96cf41a3fa6fcd6cde15f2a66de> Apart from this, if protocol undergoes a massive positive rebase, some of the shares withdrawn shares wouldn't be burnt (i.e., 'rewards' part), the behavior is intended by design.

MEDIUM-13	FastLane length change leads to the disruption of the voting logic	Acknowledged
-----------	--	--------------

Description

In the contract **HashConsensus** in the functions **_setFastLaneLengthSlots** and **_setFrameConfig**, there is no appropriate processing of the edge case when the FastLane has ended, some non-FastLane members submitted their reports, and the new **fastLaneLengthSlots** variable enables FastLane again. In that case, the contract must delete those reports sent from non-FastLane members.

```
function _submitReport(uint256 slot, bytes32 report, uint256 consensusVersion) internal {
    ...
    // change of the `fastLaneLengthSlots` variable can change fast lane members
    if (currentSlot <= frame.refSlot + config.fastLaneLengthSlots &&
        !_isFastLaneMember(memberIndex, frame.index)
    ){
        revert NonFastLaneMemberCannotReportWithinFastLaneInterval();
    }
    ...
}
```

Recommendation

Functions **_setFastLaneLengthSlots** and **_setFrameConfig** should handle this edge case.

Client's comments

More docs explaining the expected behavior in this edge case are added in <https://github.com/lidofinance/lido-dao/commit/035429dec58df551040b1c2dc0764a8db642bc4a>. The fast lane mechanism is not a critical oracle functionality, doesn't affect the oracle correctness, and is only meant to simplify the identification of malfunctioning oracles by an automated monitoring system so that DAO members can initiate a vote to remove or replace them. Thus, we don't think handling a rare edge case is worth adding a non-trivial code. Apart from adding the edge case handling logic, two changes could be implemented:

1. Remove enforcing of the fast lane in **submitReport**, leaving the mechanism advisory only.
2. Move the mechanism entirely offchain. However, having the onchain enforcement in normal cases is desirable since it improves the reliability and ease of use of the mechanism: it's enough to monitor that a quorum is consistently achieved during the fast lane period to check that all oracles are functional. Thus, we're planning to resolve this by adding more docs to the related parts of the codebase explaining the expected behavior in the mentioned edge case.

MEDIUM-14	Quorum change leads to the disruption of the voting logic	Acknowledged
-----------	---	--------------

Description

In the contract **HashConsensus** in the function **_setQuorumAndCheckConsensus**, there is no appropriate processing of the edge case when the FastLane hasn't ended, FastLane members have submitted reports, and the **quorum** variable decreases. In that case, the number of FastLane members falls, and the contract must delete those reports sent from members that used to be FastLane members.

```
function _submitReport(uint256 slot, bytes32 report, uint256 consensusVersion) internal {
    ...
    if (currentSlot <= frame.refSlot + config.fastLaneLengthSlots &&
        !_isFastLaneMember(memberIndex, frame.index)
    ) { // change of the `quorum` variable affects on the return value of the `_isFastLaneMember` function
        revert NonFastLaneMemberCannotReportWithinFastLaneInterval();
    }
    ...
}
```

Recommendation

Function **_setQuorumAndCheckConsensus** should handle this edge case.

Client's comments

Same answer as in finding "FastLane length change leads to the disruption of the voting logic".

MEDIUM-15	A single staking module can block whole deposits	Acknowledged
-----------	--	--------------

Description

A single staking module (malicious or broken) can stop the whole deposit flow.
 In the function `_loadStakingModulesCacheItem` the contract loads staking module data into the cache, for the `activeValidatorsCount` calculation it asks `totalExitedValidators` and `totalDepositedValidators` from the staking module contract and assumes that `totalExitedValidators` more than `totalDepositedValidators`.

```
(totalExitedValidators, totalDepositedValidators, cacheItem.availableValidatorsCount)
  = IStakingModule(cacheItem.stakingModuleAddress).getStakingModuleSummary();
//    ^-- obtain data from staking module

// the module might not receive all exited validators data yet => we need to replacing
// the exitedValidatorsCount with the one that the staking router is aware of
cacheItem.activeValidatorsCount =
  totalDepositedValidators -
  Math256.max(totalExitedValidators, stakingModuleData.exitedValidatorsCount);
//    ^-- calculation might underflow
```

However, in case of a broken or malicious staking module that assumption can lead to a revert caused by underflow. Since `_loadStakingModulesCacheItem` is used in the `deposit()` call it will be reverted at all and deposits will be impossible while StakingRouter staking module is paused.

Recommendation

We recommend omitting and pausing staking modules in case of underflow.

Client's comments

The described scenario must never happen when the protocol functions normally, yet in case of a bug in such an important part of the StakingRouter, we would like to explicitly revert with error all following deposit transactions till the mitigation of the issue.

MEDIUM-16	Deposits through DepositSecurityModule can be blocked by front-run	Acknowledged
-----------	--	--------------

Description

The function `depositBufferedEther` receives the `depositRoot` parameter that refers to `get_deposit_root()` function value of the official ETH deposit contract. The call reverts if `depositRoot` is not equal to `get_deposit_root()`, however `get_deposit_root()` changes after every deposit. An attacker can make a deposit by front-running every time when he/she seeks the `depositBufferedEther` call in the mempool and it will change `get_deposit_root()` value and consequently block normal deposit flow by causing a revert.

```
bytes32 onchainDepositRoot = IDepositContract(DEPOSIT_CONTRACT).get_deposit_root(); <-- this value will be changed
if (depositRoot != onchainDepositRoot) revert DepositRootChanged(); <-- this check will cause revert
```

The attack cost is almost zero if an attacker has their own validator.

Recommendation

We recommend using private mempool for this transaction.

Client's comments

The private mempool is already used for this kind of transaction. Also, the described attack requires a huge amount of ETH because failed deposit transactions might be resent once in a couple of blocks.

MEDIUM-17	Excessive penalty	Acknowledged
<p>Description</p> <p>In the function <code>NodeOperatorsRegistry._updateRefundValidatorsKeysCount</code> there is a check if all validators have been refunded:</p> <pre> if (stuckPenaltyStats.get(STUCK_VALIDATORS_COUNT_OFFSET) <= _refundedValidatorsCount) { // ^-- check for refunded validators stuckPenaltyStats.set(STUCK_PENALTY_END_TIMESTAMP_OFFSET, uint64(block.timestamp + getStuckPenaltyDelay())); } </pre> <p>If not all were refunded, then the penalized time is extended, stimulating operators not to refund step by step, but only at once. This penalty seems to be excessive.</p> <p>Recommendation</p> <p>We recommend removing the extension of punishment time in <code>_updateRefundValidatorsKeysCount</code></p> <p>Client's comments</p> <p>The behavior is intended (penalization doesn't prolongs only if refunded >= stuck)</p>		

INFORMATIONAL-01	Improper comments	Fixed at 3eec54
<p>Description</p> <p>Functions <code>increaseAllowance()</code> and <code>decreaseAllowance()</code> have improper comments at lines StETH.sol#L250 and StETH.sol#L267.</p> <p>Recommendation</p> <p>We recommend replacing the URL in the comments with IERC20.sol#L57</p>		

INFORMATIONAL-02	Redundant <code>onlyRole</code> modifier	Fixed at 7a40ae
<p>Description</p> <p>The Burner contract's <code>recoverERC20()</code> and <code>recoverERC721()</code> functions have <code>onlyRole</code> modifier, which is used to restrict "abuse of recoverERC20/recoverERC721 API to perform grieving-like attack". However, there is no sense in such a check as anyone could transfer any token directly to the Treasury and get the same effect.</p> <p>Recommendation</p> <p>We recommend removing <code>onlyRole</code> modifiers.</p>		

INFORMATIONAL-03	Array length check	Fixed at 6db4af
<p>Description</p> <p>In functions <code>claimWithdrawalsTo()</code> and <code>claimWithdrawals()</code> two arrays are passed: <code>_requestIds</code> and <code>_hints</code>, but there is no check for equality of their lengths.</p> <p>Recommendation</p> <p>We recommend adding a check for the equality of these arrays' lengths.</p>		

INFORMATIONAL-04	Redundant v component check	Fixed at c956d9
------------------	-----------------------------	---------------------------------

Description

The **ECDSA** library's function **recover()** has a redundant check for a signature's **v** component ([PR#3591](#))

Recommendation

We recommend removing this check.

INFORMATIONAL-05	Gas optimization for unlimited allowance	Fixed at e9509d
------------------	--	---------------------------------

Description

The last ERC20 implementation doesn't decrease allowance if it's set to an unlimited allowance for the spender. Optimization avoids gas spending around 5000 gas for each call.

Discussion

Places for optimization:

- StEth.sol transferFrom
- StEth.sol transferShares

Unfortunately, optimization can't be applied to WstEth, because the contract is non-upgradable.

Recommendation

We recommend adding the function spendAllowance to the contract and replacing the **allowance** call to **spendAllowance**.

INFORMATIONAL-06	Guard checks for zero secondsPerSlot	Fixed at 5a94bc
------------------	--------------------------------------	---------------------------------

Description

Constructor BaseOracle allows setting **SECONDS_PER_SLOT = 0**. However, HashConsensus does not allow setting **secondsPerSlot = 0**.

Recommendation

We recommend adding a zero check for **SECONDS_PER_SLOT** in **BaseOracle** constructor.

INFORMATIONAL-07	Useless payable conversion for call	Fixed at a06653
------------------	-------------------------------------	---------------------------------

Description

Here is made an unnecessary conversion from **address** to **address payable**
 But, we can use **call** with **address** and **address payable** unlike **send** and **transfer**, due to solidity docs.

```
function _claim(uint256 _requestId, uint256 _hint, address _recipient) internal {
    ...
    _sendValue(payable(_recipient), ethWithDiscount); // <-- unnecessary conversion to payable(_recipient)
    ...
}
function _sendValue(address _recipient, uint256 _amount) internal { // <-- _recipient declared like non-payable address
    ...
    (bool success,) = _recipient.call{value: _amount}(""); // <-- `call` available for non-payable address
    ...
}
```

Recommendation

We recommend removing **payable(_recipient)** conversion for WithdrawalQueueBase.sol claim

INFORMATIONAL-08	Stuck penalty delay extension	Fixed at 99ec8c
------------------	-------------------------------	---------------------------------

Description

If the **StakingRouter** calls to **NodeOperatorsRegistry**'s functions **updateStuckValidatorsCount** or **updateRefundedValidatorsCount** while the module was **penalized** but not **cleared**, then the module's penalty would be restarted.

Recommendation

We recommend appending a new statement in internal functions (**_updateStuckValidatorsCount** and **_updateRefundValidatorsKeysCount**) to cut off described case:

```
if (stuckPenaltyStats.get(STUCK_VALIDATORS_COUNT_OFFSET) <= _refundedValidatorsCount &&
    _stuckValidatorsCount > stuckPenaltyStats.get(STUCK_VALIDATORS_COUNT_OFFSET)) {
    stuckPenaltyStats.set(STUCK_PENALTY_END_TIMESTAMP_OFFSET,
        uint64(block.timestamp + getStuckPenaltyDelay()));
}
```

INFORMATIONAL-09	Hardcoded StakingModule limit	Fixed at b5fd97
------------------	-------------------------------	---------------------------------

Description

StakingRouters function **addStakingModule** checks if the **StakingModule** amount does not exceed 31.

Recommendation

We recommend adding a new immutable/constant **STAKING_MODULE_LIMIT** variable.

INFORMATIONAL-10	Code duplication	Fixed at 4cbb8f
<p>Description</p> <p>StakingRouter's function <code>updateStakingModule</code> reads <code>stakingModule</code> via two function calls: <code>index</code> by <code>id</code> and <code>module</code> by <code>index</code></p> <pre>uint256 stakingModuleIndex = _getStakingModuleIndexById(_stakingModuleId); StakingModule storage stakingModule = _getStakingModuleByIndex(stakingModuleIndex);</pre> <p>Recommendation</p> <p>We recommend replacing those functions with the existing one <code>_getStakingModuleById</code>.</p> <pre>StakingModule storage stakingModule = _getStakingModuleById(_stakingModuleId);</pre>		

INFORMATIONAL-11	Gas optimization	Fixed at 39b224
<p>Description</p> <p>At Line 747 <code>report</code> is checked. But in case it is zero, some computations already would be done.</p> <p>Recommendation</p> <p>It is recommended to check the <code>report</code> at the beginning of function.</p>		

INFORMATIONAL-12	Gas optimization	Fixed at 39b224
<p>Description</p> <p>At Lines 892 - 899 we can break after finding a suitable report and remove in <code>for - loop</code> condition that checks the <code>report</code></p> <p>Recommendation</p> <p>It is recommended to remove the check for <code>ZERO_HASH</code> and add a break statement if a <code>report</code> is found.</p>		

INFORMATIONAL-13	Non-callable function	Fixed at 04f8c1
<p>Description</p> <p><code>msg.sender</code> of function <code>updateTargetValidatorsLimits()</code> should have <code>STAKING_ROUTER_ROLE</code>, but <code>StakingRouter</code> contract has no methods that call this function.</p> <p>Recommendation</p> <p>It is recommended to extend <code>StakingRouter</code> logic or to leave comments if this functionality will be implemented in future versions.</p>		

INFORMATIONAL-14	Roles mismatching	Fixed at a12788
------------------	-------------------	---------------------------------

Description

Function `onWithdrawalCredentialsChanged()` is called by **StakingRouter** and has a nested call of function `invalidateReadyToDepositKeysRange()`. But **invalidateReadyToDepositKeysRange()** restricts the caller to have the role **MANAGE_NODE_OPERATOR_ROLE**. This means **StakingRouter** should have **MANAGE_NODE_OPERATOR_ROLE**, but it doesn't implement all functions.

Recommendation

It is recommended to leave a comment if it is intended behavior or change restrictions for `msg.sender` of `invalidateReadyToDepositKeysRange()`.

INFORMATIONAL-15	Withdrawal finalization before actual shares burn	Fixed at ac8b87
------------------	---	---------------------------------

Description

At [Lines 1253 - 1258](#) rewards are distributed to **StakingModules** and there is a callback `onReportsMinted()` and **StakingModule** could take advantage of the arbitrage opportunity because **TOTAL_POOLED_ETHER** is already updated, but actual shares are burnt after.

Recommendation

It is recommended to check malicious modules and burn shares before distributing rewards.

INFORMATIONAL-16	Updating variables after external calls and transfers	Fixed at 8ac425
------------------	---	---------------------------------

Description

Variables `stakingModule.lastDepositAt` and `stakingModule.lastDepositBlock` at [Lines 995-996](#) are updated after external calls and ether transfer.

Recommendation

It is recommended to update these variables before any external call and ether transfer.

INFORMATIONAL-17	Precise calculations	Fixed at f9cd0b
------------------	----------------------	---------------------------------

Description

At [Line 937](#) `_totalRewardShares` is divided by `totalActiveValidatorsCount` but such simple division can lead to the loss of precision.

Recommendation

It is recommended to multiply by a constant (e.g. **10¹⁸**) and then divide. And at **shares** calculations divide the final value by this constant.

INFORMATIONAL-18	Array length check	Fixed at 31b99f
------------------	--------------------	---------------------------------

Description

In the function `reportRewardsMinted()` two arrays are passed `_stakingModuleIds` and `_totalShares` but there is no check for equality of their lengths.

Recommendation

It is recommended to add a check for the equality of lengths of two arrays.

INFORMATIONAL-19	Active validators calculation	Fixed at 889dd2
------------------	-------------------------------	---------------------------------

Description

At [Line 795](#) for `activeValidatorsCount` calculation `totalExitedValidatorsCount` variable is used obtained from `StakingModule`, but `StakingRouter` has its own state variable for `totalExitedValidatorsCount` for each `StakingModule` and they can be different.

Recommendation

It is recommended to use a maximum of two values for `totalExitedValidatorsCount`.

INFORMATIONAL-20	Payable owner	Fixed at 8932c1
------------------	---------------	---------------------------------

Description

In struct `WithdrawalRequest` `owner` variable is payable but during the request's claiming `_recipient` is used for ether transfer and is converted to `payable(_recipient)`.

Recommendation

It is recommended to use a simple `address` type for `owner` variable in `WithdrawalRequest` struct.

INFORMATIONAL-21	Transfer of instructions necessary for initialization	Fixed at ce3bda
------------------	---	---------------------------------

Description

In the contract `PausableUntil` in the initialization function `_initializePausable`, there should be an initialization of the `resumeSinceTimestamp` variable in the slot `RESUME_SINCE_TIMESTAMP_POSITION`. However, this functionality is made in the derived contracts `WithdrawalQueue` and `ValidatorsExitBusOracle` on lines [339](#) and [103](#), respectively.

Recommendation

In the contracts, `WithdrawalQueue` and `ValidatorsExitBusOracle` delete lines [339](#) and [103](#), respectively, and then paste them in the contract `PausableUntil` in the initialization function `_initializePausable`.

INFORMATIONAL-22	Wrong event argument	Fixed at 271f9c
------------------	----------------------	---------------------------------

Description

In the contract `WithdrawalQueue` on line [342](#), in the event `InitializedV1`, there is a wrong last argument.

Recommendation

Replace `msg.sender` with `_bunkerReporter`.

INFORMATIONAL-23	Division before multiplication	Fixed at f9cd0b
------------------	--------------------------------	---------------------------------

Description

In the contract **NodeOperatorsRegistry** in the function **getRewardsDistribution**, there is a division before multiplication on [L940](#).

Recommendation

Replace with code:

```
for (idx = 0; idx < activeCount; ++idx) {
  shares[idx] = shares[idx].mul(_totalRewardShares).div(totalActiveValidatorsCount);
}
```

INFORMATIONAL-24	Multiple transfers of stETH token instead of a single transfer	Fixed at ad130a
------------------	--	---------------------------------

Description

In the contract **NodeOperatorsRegistry** in the function **_distributeRewards**, there is a burn of penalized tokens and a [separate call](#) for each penalized recipient to the **Burner** contract instead of a single call.

Recommendation

Make a single call to the **Burner** contract to burn penalized tokens.

INFORMATIONAL-25	No check on the refSlot in submitting a ConsensusReport	Fixed at cae86d
------------------	---	---------------------------------

Description

In the contract **BaseOracle** in the function **submitConsensusReport** there is no check on the argument **refSlot** that it isn't too old. It should be the current RefSlot.

Recommendation

In the function **submitConsensusReport** add this check:

```
uint256 currentRefSlot = _getCurrentRefSlot();
if (refSlot != currentRefSlot) {
  revert RefSlotTooOld(refSlot, currentRefSlot);
}
```

Client's comments

There’s no requirement from the report processor that the submitted report hash should be for the current ref. slot. From the system perspective, the only correctness requirement is that no protocol state change is possible based on the report data sampled at an earlier moment than the latest report data already resulted in a state change. This requirement is already enforced by checking the report processing deadline and by how the deadline is calculated with respect to the next frame’s reference slot.

That said, it makes sense to reject submitting a hash for a report whose processing deadline has already been missed as it would help in troubleshooting. Added the check in <https://github.com/lidofinance/lido-dao/pull/716/commits/cae86db8f1622e05d8b3e975720e0b31d473d137>.

INFORMATIONAL-26	clearNodeOperatorPenalty() can be restricted to external	Fixed at 6f37ff
------------------	--	---------------------------------

Description

The function `clearNodeOperatorPenalty()` is not called internally, so it be restricted to **external**.

Recommendation

We recommend making the function **external**.

INFORMATIONAL-27	Unsafe memory handling in getStakingRewardsDistribution()	Fixed at 48e3a5
------------------	---	---------------------------------

Description

At the line `StakingRouter.sol#L944`, `stakingModuleIds` is trimmed to length `rewardedStakingModulesCount`. At the line `StakingRouter.sol#L911`, `stakingModuleIds` will be updated regardless if the last staking module has no active validators. Then when the array is trimmed it will leave dirty bytes in memory.

Recommendation

We recommend moving the line `StakingRouter.sol#L911` inside the **if** statement.

INFORMATIONAL-28	Request owner should not be payable	Fixed at 1e17ba
------------------	-------------------------------------	---------------------------------

Description

At the line `WithdrawalQueueERC721.sol#L233`, `request.owner` is set to `payable(_to)`. In the `WithdrawalQueueBase` the **owner** is not **payable**, and recipient is **payable** during claiming. So it is unnecessary to set **owner** as **payable** when transferring.

Recommendation

We recommend removing **payable**.

INFORMATIONAL-29	Inconsistent input string for role hash	Fixed at e2fad5
------------------	---	---------------------------------

Description

At the line `OracleReportSanityChecker.sol#L104`, the name of the role `ALL_LIMITS_MANAGER_ROLE` doesn't match the input string for hash `LIMITS_MANAGER_ROLE`. However, for the other roles these match.

Recommendation

We recommend using `keccak256("ALL_LIMITS_MANAGER_ROLE")` for the sake of consistency.

INFORMATIONAL-30	Incorrect check	Fixed at f7680b
------------------	-----------------	---------------------------------

Description

In the function `SigningKeys.saveKeysSigs()`, the sum `_startIndex + _keysCount` can be `UINT64_MAX + 1`. In that case the function will return new total keys count `UINT64_MAX + 1`, but total keys count should be limited to `UINT64_MAX`.

Recommendation

We recommend removing `- 1` at the line `SigningKeys.sol#L44`.

INFORMATIONAL-31	No zero check for <code>_maxPositiveTokenRebase</code>	Fixed at df51ab
<p>Description</p> <p>There is no zero value check for parameter <code>_maxPositiveTokenRebase</code> in the function <code>setMaxPositiveTokenRebase()</code>. The comment says that passing zero value is prohibited.</p> <p>Recommendation</p> <p>We recommend adding a zero value check.</p>		

INFORMATIONAL-32	Unsafe casting	Fixed at b3eb25
<p>Description</p> <p>There is unsafe casting from <code>uint64</code> to <code>int64</code> at the lines:</p> <ul style="list-style-type: none"> • NodeOperatorsRegistry.sol#L566 • NodeOperatorsRegistry.sol#L801 <p>There is unsafe casting from <code>int64</code> to <code>uint64</code> at the lines:</p> <ul style="list-style-type: none"> • NodeOperatorsRegistry.sol#L582 • NodeOperatorsRegistry.sol#L672 <p>Recommendation</p> <p>We recommend using safe casting.</p>		

INFORMATIONAL-33	Unsafe math	Fixed at PR#719
<p>Description</p> <p>There is unsafe math at the lines:</p> <ul style="list-style-type: none"> • NodeOperatorsRegistry.sol#L582 • NodeOperatorsRegistry.sol#L620 • NodeOperatorsRegistry.sol#L672 • NodeOperatorsRegistry.sol#L706 • NodeOperatorsRegistry.sol#L725 • NodeOperatorsRegistry.sol#L828-L829 • NodeOperatorsRegistry.sol#L867 • NodeOperatorsRegistry.sol#L946 • NodeOperatorsRegistry.sol#L1109 • NodeOperatorsRegistry.sol#L1168 • NodeOperatorsRegistry.sol#L1207 <p>Recommendation</p> <p>We recommend using safe math for sanity checks.</p>		

INFORMATIONAL-34	Unused return value	Fixed at 7a93c4
------------------	---------------------	---------------------------------

Description

The function `ValidatorsExitBusOracle._processExitRequestsList()` returns value `uint256`, but the return value is unused at the line `ValidatorsExitBusOracle.sol#L344`.

Recommendation

We recommend removing the return value.

INFORMATIONAL-35	Exited validators count might be bigger than total deposited	Fixed at 8a9bd6
------------------	--	---------------------------------

Description

In the function `StakingRouter.updateExitedValidatorsCountByStakingModule()`, it is possible to set `_exitedValidatorsCounts[i]` to be bigger than `totalDepositedValidators`. In that case, the oracle report will revert due to underflow at the line `StakingRouter.sol#L1088-L1090`.

Recommendation

We recommend adding a check to avoid unexpected reverts.

INFORMATIONAL-36	Missing checks for _requestId out of range	Fixed at 58e160
------------------	--	---------------------------------

Description

In the functions `WithdrawalQueueBase.findCheckpointHint()` and `WithdrawalQueueBase._claimWithdrawalTo()`. If `_requestId > getLastRequestId()`, the functions revert with `RequestNotFinalized()` even though the request doesn't exist.

Recommendation

We recommend checking if `_requestId > getLastRequestId()` and revert with `InvalidRequestId()` like in the function `WithdrawalQueueBase.getWithdrawalRequestStatus()`.

INFORMATIONAL-37	Unused errors	Fixed at 97f112
------------------	---------------	---------------------------------

Description

The errors `Burner.NotEnoughExcessStETH()`, `WithdrawalQueue.Unimplemented()`, `WithdrawalQueue.LengthsMismatch()` are unused.

Recommendation

We recommend removing unused errors.

INFORMATIONAL-38	Unclear comments	Fixed at 55c2b0
------------------	------------------	---------------------------------

Description

At the lines:

- `OracleReportSanityChecker.sol#L446`, assigned to burn or already burnt?

Recommendation

We recommend checking if the comments are correct.

INFORMATIONAL-39	Gas optimization	Fixed at 64765b
------------------	------------------	---------------------------------

Description

At the line [StakingRouter.sol#L295](#), it is possible to use `_stakingModuleIds[i]` instead of `stakingModule.id` which reads from storage.

Recommendation

We recommend using `_stakingModuleIds[i]`.

INFORMATIONAL-40	Missing zero check when reporting minted rewards	Fixed at PR#656
------------------	--	---------------------------------

Description

In the function [reportRewardsMinted\(\)](#), `_totalShares[i]` can be `0` if the staking module is stopped and no shares were minted for it. The function [onRewardsMinted\(\)](#) should be called if the rewards were minted for the module.

Recommendation

We recommend adding a zero check for `_totalShares[i]`.

INFORMATIONAL-41	Possible underflow in trimUint256Array()	Fixed at 893d0a
------------------	--	---------------------------------

Description

In the function [MemUtils.trimUint256Array\(\)](#), there is no check if `_trimBy` is not bigger than `_arr.length`.

Recommendation

We recommend adding a check to avoid possible underflow.

Client's comments

The function is not used anymore and was completely removed

INFORMATIONAL-42	Typo in comment	Fixed at 14c355
------------------	-----------------	---------------------------------

Description

At the line [NodeOperatorsRegistry.sol#L325](#), it should be `activate` instead of `deactivate`.

Recommendation

We recommend fixing the typo.

INFORMATIONAL-43	Typo in _addSigningKeys()	Fixed at 8bb30e
------------------	---------------------------	---------------------------------

Description

At the line [NodeOperatorsRegistry.sol#L997](#), it should be `SUMMARY_TOTAL_KEYS_COUNT_OFFSET` instead of `TOTAL_KEYS_COUNT_OFFSET`.

Recommendation

We recommend fixing the typo.

INFORMATIONAL-44	Unable to get a consensus state if the quorum is unreachable	Fixed at e73508
------------------	--	---------------------------------

Description

```
function _getFastLaneSubset(uint256 frameIndex, uint256 totalMembers)
    internal view returns (uint256 startIndex, uint256 pastEndIndex)
{
    if (totalMembers != 0) {
        startIndex = frameIndex % totalMembers;
        pastEndIndex = startIndex + _quorum;
        //           ^
        //    may overflow ----|
    }
}
```

`_getFastLaneSubset` method at the line [HashConsensus.sol#L683](#) calculates `pastEndIndex` and it will overflow if `_quorum` is `UNREACHABLE_QUORUM` and `startIndex` is not zero. So all the methods using this one will also revert. As an example, `getConsensusStateForMember` method will revert at the [HashConsensus.sol#L472](#):

```
result.canReport = slot <= frame.reportProcessingDeadlineSlot &&
    frame.refSlot > _getLastProcessingRefSlot();

result.isFastLane = _isFastLaneMember(index, frame.index); // #L472

if (!result.isFastLane && result.canReport) {
    result.canReport = slot > frame.refSlot + _frameConfig.fastLaneLengthSlots;
}
```

Recommendation

We recommend reconsidering the logic of `_getFastLaneSubset` method for the situations when `_quorum` is greater than `totalMembers`.

INFORMATIONAL-45	Insufficient if statement in WithdrawalQueueBase contract	Fixed at d0b48c
------------------	---	---------------------------------

Description

```
if (_batches.length == 0) revert EmptyBatches();
uint256 lastRequestIdToBeFinalized = _batches[_batches.length - 1];
if (lastRequestIdToBeFinalized > getLastRequestId()) revert InvalidRequestId(lastRequestIdToBeFinalized);
uint256 lastFinalizedRequestId = getLastFinalizedRequestId();
if (lastRequestIdToBeFinalized <= lastFinalizedRequestId) revert InvalidRequestId(lastRequestIdToBeFinalized);
```

Condition `lastRequestIdToBeFinalized <= lastFinalizedRequestId` doesn't guarantee that all the requests are not finalized yet.

Recommendation

We recommend checking the same condition not for the last but for the first request in finalization batches, as it's made in `prefinalize` function.

```
if (_batches[0] <= getLastFinalizedRequestId()) revert InvalidRequestId(_batches[0]);
```

INFORMATIONAL-46	Lack of zero address check	Fixed at ddf756
------------------	----------------------------	---------------------------------

Description

In `AccountingOracle.constructor()` there's no check, that lido can't be set as zero address, but similar checks are provided here.

Recommendation

We recommend adding this check.

INFORMATIONAL-47	State mutability can be restricted to view	Fixed at f93b9f
------------------	--	---------------------------------

Description

Function `AccountingOracle.checkCanSubmitExtraData()` can have `view` modifier.

Recommendation

We recommend restricting its mutability.

INFORMATIONAL-48	Gas optimization	Fixed at c36635
------------------	------------------	---------------------------------

Description

Function `HashConsensus._getCurrentFrame(FrameConfig memory config)` can be deleted, cause it is used only in the function above. Instead, function `_getCurrentFrame()` can return `_getFrameAtTimestamp(...)` result itself, making one less internal call and reducing copying of memory in several sections of code.

Moreover, function `_getCurrentFrame()` could be replaced with `_getFramAtTimestamp(...)`.

Recommendation

We recommend removing this/ese unnecessary function/s.

INFORMATIONAL-49	Wrong comment in DepositSecurityModule	Fixed at b209c0
------------------	--	---------------------------------

Description

Function `DepositSecurityModule.depositBufferedEther()` has invalid comment. Signatures must be sorted in ascending order by address, not by index, due to this [line](#).

Recommendation

We recommend fixing this comment.

INFORMATIONAL-50	Gas optimization	Fixed at 6887ec
------------------	------------------	---------------------------------

Description

In `finalizeUpgrade_v2()` function, while we're reading structs from current storage, there's no value in `operatorTargetStats` slot. So gas consumption can be reduced, if we initialize an empty struct, instead of reading it from storage.

Recommendation

We recommend fixing this issue.

INFORMATIONAL-51	State mutability	Fixed at f93b9f
<p>Description</p> <p>The function <code>StakingRouter._checkValidatorsByNodeOperatorReportData()</code> state mutability can be restricted to pure.</p> <p>Recommendation</p> <p>We recommend changing it.</p>		
INFORMATIONAL-52	Modifier doesn't guarantee anything, default enum status	Fixed at ca4d8a
<p>Description</p> <p>The modifier <code>StakingRouter.validStakingModuleId()</code> doesn't guarantee that this staking module is really valid. Instead, it only restricts the area of access to possible slots to <code>UINT24_MAX</code>. The only check that reverts if stakingModule is exist provided in <code>StakingRouter._getStakingModuleIndexById</code>.</p> <p>Furthermore, without this check default status of the non-existed StakingModule is Active (which can be seen in deposit function checks), cause in the enum <code>StakingModuleStatus</code> this value corresponds to 0. This can lead to errors in future updates of protocol or using the same part of code in other places.</p> <p>As a result, more than half of several functions could be workable before the first revert.</p> <p>Recommendation</p> <p>We recommend replacing the check from <code>_getStakingModuleIndexById</code> to modifier, to be sure, that stakingModule is exist. Adding <code>StakingModuleStatus.None</code> at first place is also will be useful.</p>		
INFORMATIONAL-53	Check arrays length	Fixed at 31b99f
<p>Description</p> <p>In function <code>updateExitedValidatorsCountByStakingModule</code> there's no check that both <code>uint256[]</code> arrays are the same length.</p> <p>Recommendation</p> <p>We recommend adding this check.</p>		
INFORMATIONAL-54	Binary search boundary	Fixed at cb9049
<p>Description</p> <p><code>_end</code> is already checked in if clauses. link</p> <p>Recommendation</p> <p>Use <code>_end - 1</code> to reduce calculations.</p>		
INFORMATIONAL-55	Unnecessary variable	Fixed at fa03bf
<p>Description</p> <p><code>amountOfWstETH</code> is used only once. link</p> <p>Recommendation</p> <p><code>amounts[i]</code> can be inlined.</p>		

INFORMATIONAL-56	Penalty delay type	Fixed at PR#656
<p>Description</p> <p>Stuck penalty delay stores and return in <code>getStuckPenaltyDelay</code> as uint256. However, it is converted to uint64 when the penalty end timestamp is <u>calculated</u> and stored. In the case where this is set intentionally large, there will be an overflow for the penalty end timestamp.</p> <p>Recommendation</p> <p>We recommend storing penalty delay in uint64.</p> <p>Client's comments</p> <div> <p>The value of the stuckPenaltyDelay was limited by the constant MAX_STUCK_PENALTY_DELAY</p> </div>		

INFORMATIONAL-57	Unclear purpose of limits in Sanity Checker	Fixed at f01637
<p>Description</p> <ul style="list-style-type: none"> <code>churnValidatorsPerDayLimit</code> is described in comments as a max possible number of validators that might appear or exit. However, it <u>limits</u> only appeared validators. Also, for exited validator there is <code>maxValidatorExitRequestsPerReport</code> <code>shareRateDeviationBPLimit</code> is described as max deviation of the share rate. But for positive rebase there is <code>maxPositiveTokenRebase</code>. So it's needed only for negative rebases and for <code>checkSimulatedShareRate</code>can be added checks for it to prevent extra calculations. <p>Recommendation</p> <p>We recommend clarifying the purpose of described limits.</p> <p>Client's comments</p> <div> <p>bullet 1: churnValudatorsPerDayLimit participates in the exited checks also: https://github.com/lidofinance/lido-dao/blob/feature/shapella-upgrade/contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol#L424 (it was here for the prev commit). maxValudatorExitRequestsPerReport is not about exitED keys; it's only about requests to sign voluntary exits later.</p> <p>bullet 2: Documentation fixed in https://github.com/lidofinance/lido-dao/releases/tag/v2.0.0-beta.2</p> </div>		

INFORMATIONAL-58	Type mismatch	Fixed at 6d62b2
<p>Description</p> <p>Accounting oracle accepts stuckValidatorsCount / exitedValidatorsCount as a <u>16-byte number</u>, but the staking module uses <u>8 bytes</u> to store it.</p> <p>Recommendation</p> <p>We recommend using the same type in both cases to prevent incorrect type cast.</p>		

INFORMATIONAL-59	Incorrect information in the comment	Fixed at c5ba9a
<p>Description</p> <p>In the comments it <u>says</u> "Extra data — the oracle information that can be processed asynchronously in chunks after the main data is processed.". However, it can't be processed asynchronously in chunks.</p> <p>Recommendation</p> <p>We recommend updating comments.</p> <p>Client's comments</p> <p>The comment was more related to the nature of the data (that should allow async processing to be eligible for the extra data stage) than to the current implementation of the processing. It's already processed asynchronously after the main data and in a separate tx; support for chunked processing will be added in a future upgrade by implementing <code>EXTRA_DATA_FORMAT_TREE=2</code>.</p>		

INFORMATIONAL-60	Consecutively emitting identical events	Acknowledged
<p>Description</p> <p><code>StakingRouters</code> function <code>updateStakingModule</code> emits <u>events</u> <code>StakingModuleTargetShareSet</code> <code>StakingModuleFeesSet</code> regardless if values are changed. Via this function invocation, we may see two consecutive and identical events.</p> <p>Recommendation</p> <p>We recommend emitting an event only if the corresponding value is changed to save extra gas.</p> <p>Client's comments</p> <p>This method is planned to be called rarely, and gas costs are not critical. The decision to emit both events was chosen to simplify logic and reduce contract size.</p>		

INFORMATIONAL-61	"Bunker mode" withdrawal implementation	Acknowledged
<p>Description</p> <p>Under "Bunker mode" lido should not make new deposits and finalize withdrawal requests associated with slashing frames. The current implementation has on-chain <u>checks</u> for deposit flow but not for withdrawals.</p> <p>Recommendation</p> <p>We recommend adding a new require check inside <code>WithdrawalQueues</code> <u>finalize</u> function, restricting finalization for requests with a timestamp more than bunker mode timestamp (or other close value depending on sophisticated user action mitigation). In the case of continuous "Bunker mode", an off-chain oracle could replace the timestamp (or create a new state variable to signify a finalizable stamp within "Bunker mode") to have the ability to finalize unaffected to current slashings requests.</p> <p>Client's comments</p> <p>Unfortunately, there is no single and representative rule to enforce for timestamps if bunker mode is activated.</p>		

INFORMATIONAL-62	Uncertain reward withdrawal	Acknowledged
------------------	-----------------------------	--------------

Description

The `LidoExecutionLayerRewardsVault` contract has the `withdrawRewards` function which returns all the ETH it has if `balance > _maxAmount`. Although `withdrawRewards` method is only reachable from `_collectRewardsAndProcessWithdrawals`, right deafter `IOracleReportSanityChecker.smoothenTokenRebase` (passing `_maxAmount` value will always be less or equal than current `LidoExecutionLayerRewardsVault` balance), design of ETH transfer should be more strict.

Recommendation

We recommend using the same reverting approach as in the `WithdrawalVault::withdrawWithdrawals`:

```
uint256 balance = address(this).balance;
if (_maxAmount > balance) {
    revert NotEnoughEther(_amount, balance);
}
```

Client's comments

The suggested check is already implemented in the `OracleReportSanityChecker.checkAccountingOracleReport()` method. Moving it directly into the `LidoExecutionLayerRewardsVault` will require redeployment of the contract, which is widely used by the Lido-participating node operators. The redeployment of this contract is justifiable only in case of critical or high-severity issues.

INFORMATIONAL-63	Prefinalize SLOAD reduction	Acknowledged
------------------	-----------------------------	--------------

Description

The `prefinalize` function reads the same `LAST_FINALIZED_REQUEST_ID_POSITION` slot twice:

1. In the `InvalidRequestId` check.
2. `prevBatchEndRequestId` memory variable.

Recommendation

We recommend saving `getLastFinalizedRequestId` before the `InvalidRequestId` check.

```
if (_maxShareRate == 0) revert ZeroShareRate();
if (_batches.length == 0) revert EmptyBatches();

uint256 prevBatchEndRequestId = getLastFinalizedRequestId(); // <-- move up
if (_batches[0] <= prevBatchEndRequestId) revert InvalidRequestId(_batches[0]);
if (_batches[_batches.length - 1] > getLastRequestId()) revert InvalidRequestId(_batches[_batches.length - 1]);

uint256 currentBatchIndex;
WithdrawalRequest memory prevBatchEnd = _getQueue()[prevBatchEndRequestId];
while (currentBatchIndex < _batches.length) {
    ...
}
```

Client's comments

The proposed gas optimization will be considered for implementation in future protocol versions.

INFORMATIONAL-64	Redundant type casting	Acknowledged
------------------	------------------------	--------------

Description

The `StakingModuleCache` struct is used in `getStakingRewardsDistribution` function where the struct's `stakingModuleAddress address` field is getting stored in `recipients address` array with `address()` type casting.

```
recipients = new address[](stakingModulesCount);
```

```
struct StakingModuleCache {
    address stakingModuleAddress;
    uint24 stakingModuleId;
    uint16 stakingModuleFee;
    uint16 treasuryFee;
    uint16 targetShare;
    StakingModuleStatus status;
    uint256 activeValidatorsCount;
    uint256 availableValidatorsCount;
}
```

```
for (uint256 i; i < stakingModulesCount; ) {
    ...
    recipients[rewardedStakingModulesCount] = address(stakingModulesCache[i].stakingModuleAddress);
    ...
}
```

Recommendation

We recommend removing redundant type casting.

Client's comments

The described finding doesn’t add any risks or overheads. However, the code will be improved for better clarity in future protocol versions.

INFORMATIONAL-65	Redundant checkpoint check	Acknowledged
------------------	----------------------------	--------------

Description

The internal `_findCheckpointHint` checks that:

- `0 < _requestId <= getLastRequestId (revert InvalidRequestId);`
- `_start > 0 (revert InvalidRequestIdRange);`
- `_end <= lastCheckpointIndex (revert InvalidRequestIdRange);`
- `lastCheckpointIndex > 0 (return NOT_FOUND);`
- `_requestId <= getLastFinalizedRequestId() (return NOT_FOUND);`
- `_start <= _end (return NOT_FOUND).`

Checks 2, 3, and 6 would include such cases, where `lastCheckpointIndex > 0`. Furthermore, check 5 has the same effect as `lastCheckpointIndex > 0`, because we would have `getLastFinalizedRequestId == 0` before the first finalization.

Recommendation

We recommend removing the `lastCheckpointIndex > 0` check.

Client's comments

It was intentionally decided to enforce checkpoint invariants explicitly while the cost of double-checking is tolerable.

INFORMATIONAL-66	Claim SLOAD reduction	Acknowledged
------------------	-----------------------	--------------

Description

The internal `_claim` uses `storage` reference to access `WithdrawalRequest` from the queue and makes multiple reads from the same slot (`WithdrawalRequest` occupies 2 slots).

```
WithdrawalRequest storage request = _getQueue()[_requestId];

if (request.claimed) revert RequestAlreadyClaimed(_requestId); // <-- SLOAD
if (request.owner != msg.sender) revert NotOwner(msg.sender, request.owner); // <-- 2x SLOAD

request.claimed = true; // <-- SSTORE
assert(_getRequestsByOwner()[request.owner].remove(_requestId)); // <-- SLOAD

uint256 ethWithDiscount = _calculateClaimableEther(request, _requestId, _hint); // <-- 2x SLOAD
// uses storage reference --^
```

It calls `_calculateClaimableEther(WithdrawalRequest storage request, uint256 _requestId, uint256 _hint)` function, which passes `_request` reference to the `_calcBatch(WithdrawalRequest memory _preStartRequest, WithdrawalRequest memory _endRequest)`, copying the whole storage to the memory (2 SLOADS).

Recommendation

We recommend saving the `WithdrawalRequest` struct to the memory in the `_claim` function.

```
WithdrawalRequest memory request = _getQueue()[_requestId];

if (request.claimed) revert RequestAlreadyClaimed(_requestId);
if (request.owner != msg.sender) revert NotOwner(msg.sender, request.owner);

_getQueue()[_requestId].claimed = true; // <-- Changing storage value
assert(_getRequestsByOwner()[request.owner].remove(_requestId));

uint256 ethWithDiscount = _calculateClaimableEther(request, _requestId, _hint);
```

This would require `_calculateClaimableEther(WithdrawalRequest storage, uint256, uint256)`'s input type to change from `storage` to `memory`. Which is called from `_getClaimableEther` as well.

Client's comments

The proposed gas optimization will be considered for implementation in future protocol versions.

INFORMATIONAL-67	Avoiding heavy computations	Acknowledged
------------------	-----------------------------	--------------

Description

Function `onRewardsMinted()` is called in modules' contracts involved in rewards distribution. If heavy computations are done in this function, the whole oracle report can run out of gas.

Recommendation

It is recommended to avoid doing heavy computations in the `inRewardsMinted()` function.

Client's comments

Each staking module will have to pass the review phase before adding into the `StakingRouter`, and modules that might break the Oracle report at any stage because of being out of gas will be rejected.

INFORMATIONAL-68	Affecting on burn request	Acknowledged
<p>Description</p> <p>At Line 1286 if the operator is penalized, a burn request is submitted to the Burner contract. sharesToDistribute is calculated based on the current smart-contract balance. Anyone can transfer stETH to NodeOperatorsRegistry contract and increase rewards. In case of penalized operators, this can lead to burning more shares than expected.</p> <p>Recommendation</p> <p>It is recommended to have a state variable to store distributed rewards value.</p> <p>Client's comments</p> <div> <p>The burning of extra stETH shares can't adversely affect the protocol. stETH burning happens only on oracle's report handling and is limited by the sanity checks applied during the processing. While Burner has permissioned interface to request shares burning, it doesn't pose security risks for the protocol and is made for UX purposes (prevent users from accidentally burning their stETH)</p> </div>		

INFORMATIONAL-69	Range restriction	Acknowledged
<p>Description</p> <p>At Line 599 _targetLimit is set, but there are no limits checked for it.</p> <p>Recommendation</p> <p>It is recommended to add limit checking for _targetLimit variable (e.g. exited_keys + _targetLimit <= total_keys).</p> <p>Client's comments</p> <div> <p>The targetLimit should be set by the DAO voting which lasts for 72h. So it's hard to predict the contract's state at the voting enactment.</p> </div>		

INFORMATIONAL-70	Narrow range	Acknowledged
<p>Description</p> <p>At Line 640 _refundedValidatorsCount should be less than deposited_keys. This range could be narrowed, because _refundedValidatorsCount should be equal to or less than _stuckValidatorsCount.</p> <p>Recommendation</p> <p>It is recommended to narrow the limits of _refundedValidatorsCount variable.</p> <p>Client's comments</p> <div> <p>There is a hypothetical situation when a node operator decides to refund before stuck keys are reported. Moreover, refunded validators can be delivered concurrently with the stuck validators updated by oracle. It's better to disentangle them.</p> </div>		

INFORMATIONAL-71	Depositable ether zero check	Acknowledged
<p>Description</p> <p>At Line 706 function getDepositableEther() is called and then its value passed as argument, but there is no check for zero.</p> <p>Recommendation</p> <p>It is recommended to add zero check for return value of getDepositableEther() to prevent extra function calls.</p> <p>Client's comments</p> <div>NB: the actual implementation now explicitly allows zero deposits https://github.com/lidofinance/lido-dao/commit/8bfb8edcd43f959610a6060a94f0a53a0460073a which is used on testnet setups</div>		

INFORMATIONAL-72	Active node operator check	Acknowledged
<p>Description</p> <p>At Line 803 if depositedSigningKeysCount equals maxSigningKeysCount then this node operator is considered as inactive, but there is no check if its state is active or not.</p> <p>Recommendation</p> <p>It is recommended to add a check for the operator's activity status.</p> <p>Client's comments</p> <div>An additional check is excessive here because, on validator deactivation, its available keys count will be set to zero.</div>		

INFORMATIONAL-73	Limits for max deposits	Acknowledged
<p>Description</p> <p>In function <u>_setMaxDeposits()</u> there are no restrictions for value.</p> <p>Recommendation</p> <p>It is recommended to add limits to max deposits in block based on block gas limit and average gas cost of deposit.</p> <p>Client's comments</p> <div>Limiting it there leads to the coupling of the system. The block gas limit or gas costs of the deposit method might be changed in the future, which might require redeploying the deposit security module to bypass the limit restrictions.</div>		

INFORMATIONAL-74	No appropriate event	Acknowledged
<p>Description</p> <p>In the contract StETH in the function <u>_mintShares</u>, there is no appropriate event like SharesBurnt in the function <u>_burnShares</u>.</p> <p>Recommendation</p> <p>Add a new event SharesMint in the function <u>_mintShares</u>.</p> <p>Client's comments</p> <div>The state changes are still covered with events when <u>_mintShares</u> called.</div>		

INFORMATIONAL-75	Possible overflow	Acknowledged
<p>Description</p> <p>In library SigningKeys in the function saveKeysSigs return value can be bigger than max value of uint64 and because of that an overflow can occur in the function _addSigningKeys of the contract NodeOperatorsRegistry on L991.</p> <p>Recommendation</p> <p>In the library SigningKeys in the function saveKeysSigs on L44 check that _startIndex.add(_keysCount) <= UINT64_MAX.</p> <p>Client's comments</p> <div> <p>The returned value of the saveKeySigs method is equal to totalSigningKeysCount + _keysCount, which value is validated later: _requireValidRange(totalSigningKeysCount.add(_keysCount) <= UINT64_MAX);</p> </div>		

INFORMATIONAL-76	No post deposit root check	Acknowledged
<p>Description</p> <p>The contract StakingRouter deposits ETH into public validators' keys. Guardians must check these keys off-chain that they have no withdrawal credentials. Next, guardians submit the last confirmed deposit root to the DepositSecurityModule contract that confirms that there weren't any unchecked deposits.</p> <p>But in the depositBufferedEther function of the DepositSecurityModule contract, there is no check on deposit_root of the contract DEPOSIT_CONTRACT after the deposits that will confirm that the StakingRouter contract used good keys in the depositing.</p> <p>If something goes wrong and the StakingRouter uses a bad public validator's key, this transaction will revert because of the deposit_root check after deposits.</p> <p>Recommendation</p> <p>All staking modules implement the function obtainDepositData(uint256 _depositsCount, bytes _calldata), used in the StakingRouter contract to fetch data about public validators' keys. For example, this function could be called (using eth_call of JSON-RPC API) by guardians to receive a list of public validators' keys, check their withdrawal credentials, simulate depositing, calculate deposit_root of the DepositContract, and then pass this value into the depositBufferedEther function of the DepositSecurityModule contract, which will check the results of deposits.</p> <p>Client's comments</p> <div> <p>The recommended fix dramatically complicates the code and the signing process (guardians' daemon has to have deposit logic, supports a variety of _depositCalldata for different staking modules, and each staking module has to have its own set of signatures cause the final deposit_root will differ depends on staking module).</p> </div>		

INFORMATIONAL-77	Stuck validators update	Acknowledged
------------------	-------------------------	--------------

Description

In the contract **NodeOperatorsRegistry** in the function **_updateStuckValidatorsCount** on L615, there is a check that the value **STUCK_VALIDATORS_COUNT** variable is less than **DEPOSITED_KEYS_COUNT - EXITED_KEYS_COUNT** and this should be invariant for a node operator. Still, the contract breaks this invariant in the function **_updateExitedValidatorsCount**. So there should be a check that **DEPOSITED_KEYS_COUNT - EXITED_KEYS_COUNT** value is not less than the **STUCK_VALIDATORS_COUNT** value.

Recommendation

In the function **_updateExitedValidatorsCount**, should be added this code:

```

if (totalExitedValidatorsDelta != 0) {
    ...
    uint64 activeValidators = signingKeysStats.get(DEPOSITED_KEYS_COUNT_OFFSET) - _exitedValidatorsKeysCount;

    Packed64x4.Packed memory stuckPenaltyStats = _loadOperatorStuckPenaltyStats(_nodeOperatorId);
    if (stuckPenaltyStats.get(STUCK_VALIDATORS_COUNT_OFFSET) > activeValidators) {
        _updateStuckValidatorsCount(_nodeOperatorId, activeValidators);
    }
    ...
}

```

Client's comments

This invariant may be violated only when the extra data report includes data about exited validators and miss an item with the stuck validators update for the same node operator. But this invariant will be recovered in future reports when the data about stuck validators of the node operator will be delivered.

Description

At the line [OracleReportSanityChecker.sol#L422](#), `_checkAnnualBalancesIncrease()` doesn't include withdrawn ether like `_checkOneOffCLBalanceDecrease()`.

Consider a case:

```
// Initial state
preCLBalance = x;
withdrawalVaultBalance = 0;

// The validators receive 16 Ether rewards
// No validators exited or deposited
// During the next oracle report call:
preCLBalance = x;
withdrawalVaultBalance = 0;
postCLBalance = x + 16;

// the check is made for 16 Ether

// The validators receive 16 Ether rewards
// 16 Ether is automatically skimmed to withdrawal vault
// No validators exited or deposited
// During the next oracle report call:
preCLBalance = x + 16;
withdrawalVaultBalance = 16;
postCLBalance = x + 16;

// there is no check even though CL balance increased by 16 Ether

// The validators receive 16 Ether rewards
// No Ether is automatically skimmed to withdrawal vault
// 1 validator exited, no validators deposited
// During the next oracle report call:
preCLBalance = x + 16;
withdrawalVaultBalance = 48;
postCLBalance = x + 16 + 16 - 32 = x;

// there is no check even though CL balance increased by 16 Ether
```

Due to partial rewards skimming and validator exits there is inconsistency when the check is applied. One approach would be to save the withdrawal vault balance at the end of the previous report and during the next report add to **postCLBalance** only the ether which arrived between the reports. In that case, for the above example, every check would be for 16 ether.

Recommendation

We recommend checking if this is intended behavior.

Client's comments

The current behavior is intended. Withdrawals vault balance has its check separate from the annual CL balance increase. Counting it in the CL balance check introduces the risk of DOS via ETH sending into WithdrawalsValue

INFORMATIONAL-79	SanityChecker includes withdrawn ether from past reports	Acknowledged
------------------	--	--------------

Description

The function `OracleReportSanityChecker._checkOneOffCLBalanceDecrease()` checks if the CL balance can only decrease by a limited amount since the last report. The variable `_unifiedPostCLBalance` includes withdrawal vault balance to account for withdrawals that have happened since the last oracle report. However, it also includes withdrawn ether from validators which exited before the latest oracle report. This can lead to a situation where when the CL balance does decrease, the check passes when it should not pass.

Consider a case:

```
// Initial state with one validator and some ether from withdrawals and partial rewards
preCLBalance = 32;
withdrawalVaultBalance = 16;

// The validator is slashed/penalized for 16 Ether
// During oracle report call:
preCLBalance = 32;
withdrawalVaultBalance = 16;
postCLBalance = 16;
unifiedPostCLBalance = 32;

// _checkOneOffCLBalanceDecrease() passes since preCLBalance == unifiedPostCLBalance
// even though CL balance decreased
```

One approach would be to save the withdrawal vault balance at the end of the previous report and during the next report add to `postCLBalance` only the ether which arrived between the reports.

Recommendation

We recommend checking if this is intended behavior.

Client's comments

The current behavior is intended. The withdrawal vault balance is zeroed most of the time (with an exception of positive rebase smoothening, however, the withdrawals vault balance is applied first for the rebase limiter).

INFORMATIONAL-80	Unused function parameters	Acknowledged
------------------	----------------------------	--------------

Description

In the function `BaseOracle._handleConsensusReport()`, the parameters `report` and `prevSubmittedRefSlot` are unused in `AccountingOracle.sol` and `ValidatorsExitBusOracle.sol`.

Recommendation

We recommend removing unused parameters.

Client's comments

We'll consider removing these unused parameters in a future upgrade.

INFORMATIONAL-81	No checks for node operator id and validator index	Acknowledged
------------------	--	--------------

Description

In the function `ValidatorsExitBusOracle._processExitRequestsList()`, there are no checks if the node operator exists `nodeOpId < IStakingModule.getNodeOperatorsCount()`, and if the node operator validator index is valid `valIndex < totalDepositedValidators`.

Recommendation

We recommend considering adding sanity checks for these values.

Client's comments

Checking for a validator index is impossible since the total number of deposited validators is currently unavailable on the execution layer, so the only way to check it is to require the oracle to provide it which would make the check useless. We'll consider adding the check for a valid node operator id in the future. That said, even if the oracle submits a non-existent node operator id, this won't lead to any immediate incorrectness since exit requests are events interpreted offchain by affected node operators, and in the case of a non-existent node operator id there would just be no one to interpret the erroneous event. However, if an operator with this id is added later, its last requested validator index would be initially set to an incorrect value instead of zero, and recovering from this would require upgrading the contract—so the proposed check is still desired.

INFORMATIONAL-82	No check for <code>_etherToLockOnWithdrawalQueue</code>	Acknowledged
------------------	---	--------------

Description

In the function `_collectRewardsAndProcessWithdrawals()`, there is no check if `_etherToLockOnWithdrawalQueue <= address(this).balance`. The oracle can send the `_lastFinalizableRequestId` parameter such that there is not enough ether in the **Lido** contract balance to finalize withdrawals.

Recommendation

We recommend checking if `_etherToLockOnWithdrawalQueue <= address(this).balance`.

Client's comments

The call, anyway, will be reverted in an attempt to transfer the ether to the withdrawal queue.

INFORMATIONAL-83	Missing length check in <code>loadKeysSigs()</code>	Acknowledged
------------------	---	--------------

Description

In the function `loadKeysSigs()`, there is no length check for `_pubkeys` and `_signatures`. These variables should be of length to fit `_keysCount` number of keys and signatures respectively starting from `_bufOffset`.

Recommendation

We recommend adding a sanity check to ensure the loaded keys fit.

Client's comments

This check was skipped for gas savings because `loadKeysSigs` is used in pair with `initKeysSigsBuf` which reserves correct space for keys and signatures.

INFORMATIONAL-84	No checks for a different values in the setter functions	Acknowledged
<p>Description</p> <p>In the contract DepositSecurityModule, the functions <u>_setOwner()</u>, <u>_setPauseIntentValidityPeriodBlocks()</u>, <u>_setMaxDeposits()</u>, there are no checks if the new value is different than the current value. In the functions <u>_setMinDepositBlockDistance()</u>, <u>_setGuardianQuorum()</u> there are checks if the new value is different.</p> <p>Recommendation</p> <p>We recommend adding checks in the functions <u>_setOwner()</u>, <u>_setPauseIntentValidityPeriodBlocks()</u>, <u>_setMaxDeposits()</u>.</p> <p>Client's comments</p> <div> <p>For code simplicity, checks on setting the same values were omitted. These methods are supposed to be called rarely, so gas extra costs are not so important in this case.</p> </div>		

INFORMATIONAL-85	StakingRouter can return exited validators that hasn't been updated	Acknowledged
<p>Description</p> <p>The function <u>StakingRouter.getStakingModuleSummary()</u> returns summary.totalExitedValidators taken from staking module which can differ from the total exited stored in the staking router.</p> <p>Recommendation</p> <p>We recommend returning the maximum of both values.</p> <p>Client's comments</p> <div> <p>This method is supposed to return the same values as IStakingModule.getStakingModuleSummary().</p> </div>		

Description

The function `OracleReportSanityChecker._checkAnnualBalancesIncrease()` checks that the consensus layer balance of Lido validators doesn't increase beyond a certain annual limit. This can lead to DOS attack to block oracle reports. An attacker can send a deposit to the Ethereum 2.0 deposit contract with a public key of a Lido validator to increase that validator's balance on the consensus layer such that **annualBalanceIncrease** is bigger than the limit. The function `apply_deposit()` from [beacon-chain.md#](#):

```
def apply_deposit(state: BeaconState,
                 pubkey: BLS Pubkey,
                 withdrawal_credentials: Bytes32,
                 amount: uint64,
                 signature: BLSSignature) -> None:
    validator_pubkeys = [v.pubkey for v in state.validators]
    if pubkey not in validator_pubkeys:
        # Verify the deposit signature (proof of possession) which is not checked by the deposit contract
        deposit_message = DepositMessage(
            pubkey=pubkey,
            withdrawal_credentials=withdrawal_credentials,
            amount=amount,
        )
        domain = compute_domain(DOMAIN_DEPOSIT) # Fork-agnostic domain since deposits are valid across forks
        ...

        # Add validator and balance entries
        state.validators.append(get_validator_from_deposit(pubkey, withdrawal_credentials, amount))
        state.balances.append(amount)
    else:
        # Increase balance by deposit amount
        index = ValidatorIndex(validator_pubkeys.index(pubkey))
        increase_balance(state, index, amount)
```

The attacker can send a deposit message without having the Lido validator's private key and increase the balance. The accounting oracle calculates the CL balance as the sum of current balances [accounting.py#L204](#):

```
def _get_consensus_lido_state(self, blockstamp: ReferenceBlockStamp) -> tuple[int, Gwei]:
    lido_validators = self.w3.lido_validators.get_lido_validators(blockstamp)

    count = len(lido_validators)
    total_balance = Gwei(sum(int validator.balance for validator in lido_validators))

    logger.info({'msg': 'Calculate consensus lido state.', 'value': (count, total_balance)})
    return count, total_balance
```

However, this attack is unlikely to have financial incentive for the attacker. Depending on the **annualBalanceIncreaseBPLimit** it will be increasingly expensive to block oracle reports for a long time.

Recommendation

We recommend assessing the risk of such an attack.

Client's comments

Considering the current TVL of the protocol and the expected value for **annualBalanceIncreaseBPLimit** equal to 10.00%, the attack will be insanely expensive for an attacker to DOS multiple reports in a row still with a tolerable impact on the protocol. Moreover, Lido DAO can raise the **annualBalanceIncreaseBPLimit** in case of such an attack.

INFORMATIONAL-87	Possibility to deposit the same validator address	Acknowledged
<p>Description</p> <p><u><code>_makeBeaconChainDeposits32ETH</code></u> method doesn't check that all the public keys are unique, so it is possible to deposit the same address multiple times.</p> <p>Recommendation</p> <p>We recommend checking if all public keys are unique.</p> <p>Client's comments</p> <p>All keys used for the deposits pass the checks off-chain before using them in deposits. Adding such a check into the on-chain code will considerably increase the gas costs and complexity of the <code>_makeBeaconChainDeposits32ETH()</code> method.</p>		

INFORMATIONAL-88	Extra if statement in <code>_checkConsensus</code> method	Acknowledged
<p>Description</p> <pre>if (_computeSlotAtTimestamp(timestamp) > frame.reportProcessingDeadlineSlot) { // reference slot is not reportable anymore return; }</pre> <p><u><code>_checkConsensus</code></u> method at the line HashConsensus.sol#L861 performs an extra check, the condition</p> <pre>_computeSlotAtTimestamp(timestamp) > frame.reportProcessingDeadlineSlot</pre> <p>in the if statement is always false. Moreover, the comment in the if body does not match the actual logic of the condition.</p> <p>Recommendation</p> <p>We recommend removing the if statement.</p> <p>Client's comments</p> <p>This finding is correct; however, it's only correct due to the reporting deadline being set to the last slot of the frame (i.e. the reference slot of the next frame) which is a detail that may change in the future. The offset of the reference slot was moved to a constant in https://github.com/lidofinance/lido-dao/commit/b91ad87f6706085247c5f4de39738ebf186ef1c7 to express this more clearly.</p>		

INFORMATIONAL-89	Impossible to set reportProcessor if the initial epoch is in the future	Acknowledged
<p>Description</p> <p>It is impossible to execute the method <code>setReportProcessor</code> of <code>HashConcensus</code> contract if <code>_frameConfig.initialEpoch</code> is in the future. The execution will revert after the line <code>HashConcensus#L1061</code> cause of <code>_getCurrentFrame()</code> method.</p> <p>Recommendation</p> <p>We recommend reconsidering the logic of setting Report Processor if the initial epoch is in the future.</p> <p>Client's comments</p> <p>The recommended solution might be used only when Transfer events are emitted after all shares were minted (in other cases <code>Transfer.value</code> parameter will have an incorrect value). It, in turn, will require a second for loop to emit an event for each staking module. Such change will complicate the logic and almost eliminate the gas savings from the <code>_mintShares</code> method usage.</p>		

INFORMATIONAL-90	DEADLINE_SLOT_OFFSET	Acknowledged
<p>Description</p> <p><code>DEADLINE_SLOT_OFFSET</code> is a constant field of <code>HashConcensus</code>. Cause <code>DEADLINE_SLOT_OFFSET</code> value may change in the future as mentioned in the comments it will require deploying the whole contract again rather than changing the value in the current instance.</p> <p>Recommendation</p> <p>We recommend adding the possibility to change this field.</p> <p>Client's comments</p> <p>It is an intended decision. Change of this constant might happen in the extremely rare cases, so for gas savings, this value is declared as constant. Also, a change in this variable most likely leads to changes in the logic and redeployment of related contracts.</p>		

INFORMATIONAL-91	Soft requirement for transfer	Acknowledged
<p>Description</p> <p>Function <code>LidoExecutionLayerRewardsVault.withdrawRewards()</code> is used only inside <code>Lido._handleOracleReport()</code>. Both balances from <code>LidoExecutuinLayerRewardsVault</code> and <code>WithdrawalsVault</code> are checked, during <code>_handleOracleReport</code> but <code>WithdrawalVault.withdrawRewards()</code> reverts, if there's not enough balance. So, in case, that will change checks about balances, <code>ELRewardsVault.withdrawRewards()</code> doesn't guarantee, that the required amount will be transferred, which can lead to math errors.</p> <p>Recommendation</p> <p>We recommend changing soft check inside <code>LidoExecutuinLayerRewardsVault.withdrawRewards()</code> function to similar one, which is used inside <code>WithdrawalVault.withdrawRewards()</code>.</p> <p>Client's comments</p> <p>The <code>LidoExecutionLayerRewardsVault</code> contract was deployed earlier, and its address is widely used by the Lido-participating node operators. The redeployment of this contract is justifiable only in case of critical or high-severity issues.</p>		

INFORMATIONAL-92	Gas optimization	Acknowledged
------------------	------------------	--------------

Description

In function `Lido._distributeFee()`([Lido.sol#L994](#) Lido protocol `mintShares()` to `address(this)` and then transfer them in `_transferTreasuryRewards()` and `transferModuleRewards()`. But the transfer function consumes more gas, than mint, and a lot of checks are not needed in our case, cause we already know, that we have enough balance.

Recommendation

We recommend removing `_mintShares(address(this), sharesMintedAsFees)` and make `_mintShares(recipient, amount)` instead of `transferShares(address(this), recipient, amount)` in both functions, cause it will save some gas.

Client's comments

The recommended solution might be used only when **Transfer** events are emitted after all shares were minted (in other cases **Transfer.value** parameter will have an incorrect value). It, in turn, will require a second for loop to emit an event for each staking module. Such change will complicate the logic and almost eliminate the gas savings from the `_mintShares` method usage.

INFORMATIONAL-93	Gas optimization: MinAllocation	Acknowledged
------------------	---------------------------------	--------------

Description

[MinFirstAllocationStrategy](#) in function `allocate` calls `allocateToBestCandidate` in loop until all `allocationSize` candidates will be allocated. In function `allocateToBestCandidate` there are two loops:

- to find min in **buckets** and the number of min elements.
- to find the upper bound of min. And since the array **buckets** remain the same, it doesn't have to be done every iteration, only when the min changes.

Recommendation

We recommend optimizing the allocation algorithm. Return min, the number of mins, the upper bound of min, and the number of upper bounds in `allocateToBestCandidate` and reuse them in the next call.

Client's comments

Improvements to the allocation algorithm will be considered in future releases

STATE MIND