MixBytes()

Lido CSM Security Audit Report

JULY 18, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.3 Project Overview

The Community Staking Module (CSM) is a permissionless staking module designed to onboard community stakers as Node Operators within the Lido on Ethereum protocol. The primary objective of CSM is to democratize the participation in staking activities by minimizing entry barriers and fostering a decentralized network of validators.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Lido
Project name	CSM
Timeline	23.07.2024 - 08.07.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
23.07.2024	a898d045b63303294752d1a60ad9dfe8d8ba69ca	Commit for the audit (Community Staking Module)
23.07.2024	fafa232a7b3522fdee5600c345b5186b4bcb7ada	Commit for the audit (Core)
23.07.2024	6d63d68f8d01668114dc46e170a24e400976d950	Commit for the audit (EasyTrack 1)
23.07.2024	13f78f1ec44436abfb1b2a55f640e2178f79d029	Commit for the audit (EasyTrack 2)
14.08.2024	638933f0e0740e049328b6e2dba616a4b93d1e64	Commit for the audit (Data Bus)
17.09.2024	0e4b33a2a0c8679319c4befd0c92c84107fa2bd1	Commit for the re-audit (Community Staking Module)
25.09.2024	1ffbb7e49e112fcac678f59bf63ba57a7e522874	Commit for the re-audit (Core)

Date	Commit Hash	Note
26.09.2024	cdc6e335ae907a5ba1160091824ce2ebe2e0842f	Commit for the re-audit 2 (Community Staking Module)
01.10.2024	9c7d014844395b37a64b3ca4ed5dc27de0af23a5	Commit for the re-audit 3 (Community Staking Module)
07.10.2024	347496df916c3b987a7f3fe8b0bd85c9b62ad730	Commit with the update (Community Staking Module)
23.01.2025	3469910c0d29a54b37d0c4de3cf527a3e7be2099	Commit with the update (Community Staking Module)
08.07.2025	d29a0bf4dac8d78550016beb82dbf16431db5ced	Commit with the update (Community Staking Module)

Project Scope

The audit covered the following files:

File name	Link
src/CSFeeDistributor.sol	CSFeeDistributor.sol
src/CSAccounting.sol	CSAccounting.sol
src/CSFeeOracle.sol	CSFeeOracle.sol
src/CSVerifier.sol	CSVerifier.sol
src/CSModule.sol	CSModule.sol
src/CSEarlyAdoption.sol	CSEarlyAdoption.sol
src/lib/base-oracle/ HashConsensus.sol	HashConsensus.sol
src/lib/base-oracle/ BaseOracle.sol	BaseOracle.sol

File name	Link
src/lib/SigningKeys.sol	SigningKeys.sol
src/lib/proxy/OssifiableProxy.sol	OssifiableProxy.sol
src/lib/TransientUintUintMapLib.sol	TransientUintUintMapLib.sol
src/lib/NOAddresses.sol	NOAddresses.sol
src/lib/ValidatorCountsReport.sol	ValidatorCountsReport.sol
src/lib/utils/PausableUntil.sol	PausableUntil.sol
src/lib/utils/Versioned.sol	Versioned.sol
src/lib/Types.sol	Types.sol
src/lib/SSZ.sol	SSZ.sol
src/lib/GIndex.sol	GIndex.sol
src/lib/QueueLib.sol	QueueLib.sol
src/lib/AssetRecovererLib.sol	AssetRecovererLib.sol
src/lib/UnstructuredStorage.sol	UnstructuredStorage.sol
src/abstract/CSBondLock.sol	CSBondLock.sol
src/abstract/CSBondCurve.sol	CSBondCurve.sol
src/abstract/CSBondCore.sol	CSBondCore.sol
src/abstract/AssetRecoverer.sol	AssetRecoverer.sol
contracts/0.4.24/nos/NodeOperatorsRegistry.sol	NodeOperatorsRegistry.sol
contracts/0.8.9/DepositSecurityModule.sol	DepositSecurityModule.sol
contracts/0.8.9/StakingRouter.sol	StakingRouter.sol

File name	Link
contracts/0.8.9/oracle/AccountingOracle.sol	AccountingOracle.sol
contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol	OracleReportSanityChecker.sol
contracts/common/lib/MinFirstAllocationStrategy.sol	MinFirstAllocationStrategy.sol
contracts/EVMScriptFactories/UpdateTargetValidatorLimits.sol	UpdateTargetValidatorLimits.sol
contracts/EVMScriptFactories/CSMSettleELStealingPenalty.sol	CSMSettleELStealingPenalty.sol
contracts/DataBus.sol	DataBus.sol

Deployments

File name	Contract deployed on mainnet	Comment
OssifiableProxy.sol	0xdA7dE2...Bf9EA83F	Proxy for CSModule.sol
CSModule.sol	0x8daEa5...7c1D895B	
CSVerifier.sol	0xec6cc1...f587b05d	
CSEarlyAdoption.sol	0x3D5148...67F90c0E	
OssifiableProxy.sol	0x4d72BF...b069e5Da	Proxy for CSAccounting.sol
CSAccounting.sol	0x71FCD2...aabf2758	
OssifiableProxy.sol	0xD99CC6...F6F618D0	Proxy for CSFeeDistributor.sol
CSFeeDistributor.sol	0x17Fc61...22E444f0	
OssifiableProxy.sol	0x4D4074...38DdF7FB	Proxy for CSFeeOracle.sol
CSFeeOracle.sol	0x919ac5...a7B0381E	

File name	Contract deployed on mainnet	Comment
HashConsensus.sol	0x71093e...C80688e4	
AssetRecovererLib.sol	0xa74528...71Cc01D9	
NOAddresses.sol	0xF8E5de...57131C72	
QueueLib.sol	0xD19B40...452f70f9	
MinFirstAllocationStrategy.sol	0x7e70De...7142d993	
StakingRouter.sol	0x89eDa9...D2eF81aA	
NodeOperatorsRegistry.sol	0x177004...c9B42135	
DepositSecurityModule.sol	0xfFA96D...8e3d72fD	
AccountingOracle.sol	0x0e6589...6Dc29bC7	
OracleReportSanityChecker.sol	0x623239...E9461E75	
UpdateTargetValidatorLimits.sol	0x161a45...ee0f399B	
CSMSettleElStealingPenalty.sol	0xF6B6E7...5fDa76f4	

1.5 Summary of findings

Severity	# of Findings
CRITICAL	0
HIGH	0
MEDIUM	10
LOW	31

1.6 Conclusion

During the audit, we thoroughly tested critical attack vectors and verified the following:

- It is impossible to initialize or compromise contract implementations. The `Versioned` contract is used in upgradeable contracts. It helps control the contract version and ensures that any upgrades correctly increase the contract version number. Each `initialize` function is properly protected by the `onlyInit` modifier or by the check inside the `Versioned` contract, which ensures that the current contract version is equal to 0 (can be initialized), and each `finalizeUpgrade` function can be called only if there has been no previous upgrade to the target contract version.

- `calldata` parameters cannot be exploited to upload invalid keys. It is possible to add invalid deposit data, but it won't be used to perform a deposit to the Beacon Chain. Also, each added set of validator data requires a bond to be deposited, so there is an economical disincentive to provide invalid data. Even if attempted, off-chain services would detect and reject them. `bytes` arrays passed as `calldata` are correctly processed. For `publicKeys` and `signatures`, when a new validator is created, the `SigningKeys` library is used to read public keys and signatures from the `bytes` arrays provided in `calldata`. In case any incorrect validator data is added, there is an unvetting functionality. It is used to mark stored keys and signatures as invalid to prevent their use as deposit data in the future.

- Tiny deposits from random external accounts to a Node Operator's bond are safely handled as donations, without affecting internal logic. It is possible to deposit ETH, stETH, or wstETH to the CSModule. This can be done by anyone, but only to an existing Node Operator. All such deposits are accounted for in the Node Operator's bond. During each deposit, an internal function `_increaseBond` is invoked to update the bond shares counters.

- The minimum bond for active validators is correctly enforced by the CSAccounting contract. There are multiple functions, such as `_getUnbondedKeysCount` and `getRequiredBondForNextKeys`, which use bond curves for Node Operators to determine how much bond is needed for the number of added validator keys. These functions also check if there is any locked bond due to reported slashings or EL rewards stealing.

- If a node operator removes keys before unvetting, the fee and nonce checks in DSM prevent misuse. There is an `unvetSigningKeys` function in the `DepositSecurityModule` contract which checks that the nonce of the staking module matches the nonce provided to DSM with the unvet message. If a user decides to call `removeKeys` right before the `decreaseVettedSigningKeysCount` is called by the Staking Router, they would be charged a fee for each removed key and the staking module nonce would be increased. These checks and charges make such an attack worthless.

- The `_updateDepositableValidatorsCount` function is secure against underflow. All calculations in this function are performed after checks that ensure there are no reverts or underflows inside unchecked blocks. The most important part is accounting for unbonded, stuck, and non-deposited validator counts. This affects the `newCount` variable, which is used to update the `depositableValidatorsCount` variable for a particular Node Operator.

- A node operator in the CSModule can remove multiple keys with minimal charges if they were slashed, which is intentional behavior. If there is a locked bond after the Node Operator penalization, and a user tries to remove not-yet-deposited validator keys, then the

keyRemovalCharge should be deducted from the Node Operator's bond. However, if the key removal charge is larger than the current Node Operator bond, the _reduceBond function in the CSBondCore contract would reduce the bond only by the current amount of shares. Despite being a highly unlikely scenario, this case was thoroughly checked and there weren't any additional attack vectors discovered.

- There is a known delay in processing reports, which may cause the availableValidatorsCount to temporarily show a lower value than expected. There is a multi-phased oracle report processing system in place. In the first phase, exited validators' counts are updated only in the Staking Router, and during the second phase, oracle reports update stuck and exited validators' counts directly to the staking modules. Additionally, stuck and exited validators' data is reported in multiple consecutive transactions, which allows for separating extra report data into chunks.

- The addNodeOperatorStETH and addNodeOperatorWstETH functions correctly process permit front-running. If anyone decides to front-run a call to either of the mentioned functions with specified permit data, it will not break the staking module logic or result in depositing on behalf of another user.

- Adding multiple Node Operators with just one validator key doesn't lead to contract DoS. In such cases, each validator key would be added to the queue as a separate batch, which theoretically could lead to out-of-gas errors if there are any loops going over the whole queue. However, each function that accesses the queue uses special arguments which limit the number of iterations. For example, there is maxItems for the queue clean function and maxDepositsPerBlock in DepositSecurityModule, which is then used as depositsCount in the obtainDepositData function.

- It is impossible to withdraw bond shares belonging to Node Operators from the CSAccounting contract using the recoverStETHShares function. If any stETH shares were mistakenly sent directly to the CSAccounting contract, they can still be recovered. The recoverable amount is calculated using the contract balance minus the totalBondShares belonging to the Node Operators' bonds.

- Reports with an incorrect refSlot are filtered in the HashConsensus contract. There is a _checkConsensus function which ensures that the submitted report is at the correct refSlot of the frame and that a report with the provided refSlot hasn't been submitted for processing earlier.

- Each operation that changes the number of validator keys leads to a staking module nonce increment. All functions that lead to a change in the number of validator keys were thoroughly checked to update the module nonce, so that off-chain services and the Deposit Security Module can check it against the tracked value.

CSVerifier contract update (presented in the commit CSVerifier.sol):
The voting script presented in PR-435 rotates the CSVerifier contract configured in the Community Staking Module (CSM) from the old implementation (0x0c345d...ffA38e8B) to the new one (0xeC6Cc1...F587b05d). The script was reviewed, and it was confirmed that the rotation is performed correctly by transferring the VERIFIER_ROLE from the old address to the new one, as seen in steps 17–18 of the script.

2. FINDINGS REPORT

2.1 Critical

Not found

2.2 High

Not found

2.3 Medium

1. Inconsistent Bond Curve Update Handling

Status
ACKNOWLEDGED
Description
The issue is identified within the function CSAccounting.sol#L168-L173 of contract CSAccounting. When updating a bond curve, there is a potential issue where the unbonded validators amount increases for a specific NO (node operator), but the corresponding depositable amount is not updated accordingly. This inconsistency can lead to unexpected behavior in the system, as the depositable amount should be forced to update, but currently, it is not possible to enforce this.
The issue is classified as **Medium** severity because it can result in inconsistencies within the contract's logic, potentially leading to deposits of NO's keys without appropriate bond.
Recommendation
We recommend conducting bond curve update in a 2-step way, when NO's curve Id is updated on any action of this NO which will guarantee that unbonded keys and depositable keys update simultaneously.
Client's commentary

Making a two-step change might be complicated. Also, it is assumed that in most cases, the curve would be changed to lower values than the current one. However, should the curve be changed to higher values, the `normalizeQueue` method can be called individually for each NO with the change curve since the `normalizeQueue` method was changed to permissionless after the start of the audits.

2. Potential Mismanagement of Updated Withdrawal Vault Addresses

Status
Fixed in 0e4b33a2
Description
The issue is identified within the function CSVerifier.sol#L308-L310 of the contract CSVerifier. The current implementation verifies that the withdrawalAddress matches WITHDRAWAL_ADDRESS. However, if the withdrawal vault address is updated through a contract upgrade, this check may fail for old deposits, leading to incorrect handling of those withdrawal processes.
The issue is classified as **Medium** severity because it can cause inconsistencies in withdrawal processing, potentially affecting users' funds and the contract's reliability.
Recommendation
We recommend implementing a mechanism to track historical withdrawal vault addresses or ensuring backward compatibility when updating the withdrawal vault address.
Client's commentary

It's possible to have multiple instances of `CSVerifier` attached via the `VERIFIER_ROLE` role to the `CSModule` with different `WITHDRAWAL_ADDRESS`'es configured. A user should just use the correct instance to bring its proof.

3. Unchecked Guardian Quorum Value

Status

ACKNOWLEDGED

Description

The issue is identified within the DepositSecurityModule.sol#L247-L254 function of the DepositSecurityModule contract. The function allows setting a guardian quorum value without checking if it exceeds the number of active guardians.

The issue is classified as **Medium** severity because it could lead to operational disruptions and governance issues if the quorum set is greater than the number of available guardians.

Recommendation

We recommend introducing a validation step within the _setGuardianQuorum function to ensure that the new quorum value does not exceed the current number of guardians.

Client's commentary

Changing the quorum can only be done through on-chain voting. All parameters are double-checked, acceptance tests are expected to be written for each vote, and any inflated quorum value can be corrected by subsequent voting. The absence of the check is intentional, and this behavior is documented in the code.

4. Reuse of Guardian Signatures for Pausing Deposits

Status

ACKNOWLEDGED

Description

The issue is identified within the DepositSecurityModule.sol#L384 function of the DepositSecurityModule contract. This function checks for guardians' signatures to pause deposits but does not utilize a nonce mechanism. Consequently, a previously used signature could potentially be replayed to initiate another pause, leading to abuse or unintended operational disruptions.

The issue is classified as **Medium** severity because it could allow an attacker to repeatedly pause the system by reusing an old, valid signature, thereby disrupting normal operations and potentially causing governance issues.

Recommendation

We recommend implementing a nonce system for signing operations. This will prevent the reuse of signatures and enhance the security and robustness of the contract.

Client's commentary

The DSM setup assumes that the unpausing of deposits can only be performed through on-chain DAO voting, the duration of which is significantly shorter than the expiration time of the pause intention.
Therefore, by the time the pause is lifted, all signed pause intentions will have expired.

5. Bond Curve is Not Reset Inside the submitInitialSlashing Function

Status

ACKNOWLEDGED

Description

The issue is identified within the CSModule.sol#L1174 function of the CSModule contract, where the initial slashing penalty is applied to the Node Operator bond. While this penalization occurs, the Bond Curve for that Node Operator is not reset to the default.

The issue is classified as **Medium** severity because it may lead to incorrect accounting of the unbonded validator keys count after the Node Operator penalization.

Recommendation
We recommend resetting the Node Operator bond curve during the mentioned penalization event via a call to `accounting.resetBondCurve(nodeOperatorId)`.
Client's commentary

The corresponding bond curve reset is done within the submitWithdrawal method CSModule.sol#L1139 It is done intentionally to prevent unnecessary unbonded validators that might occur after slashing reporting but before withdrawal reporting that will reduce totalKeysCount used to calculate required bond

6. Possible missing `_updateDepositableValidatorsCount` calls

Status
ACKNOWLEDGED
Description
The issue is identified within the CSAccounting.sol#L178 and CSAccounting.sol#L188 functions of the `CSAccounting` contract.
It is expected that the mentioned functions may be called by an external trusted entity like DAO. The functions change Node Operator bond curve which may lead to unbonded validator keys count change.
The issue is classified as **Medium** severity because it may lead to incorrect accounting of unbonded validator keys count after the Node Operator bond curve update.
Recommendation
We recommend making the calls to `setBondCurve` and `resetBondCurve` functions only from `CSModule` contract. Such calls should be followed by the `_updateDepositableValidatorsCount` calls which will update depositable validators count. In the future, when the DAO would be permitted to call the mentioned functions, the entry point for such calls would be inside the `CSModule` contract with all necessary updates.
Client's commentary

Similarly to the finding #1 it is proposed to call `normalizeQueue` after calling `setBondCurve` and `resetBondCurve` externally.

7. Griefing of `CSModule.addValidatorKeysETH()` with `CSModule.depositETH()`

Status
Fixed in `0e4b33a2`
Description
`CSModule.addValidatorKeysETH()` CSModule.sol#L388-L393 that `msg.value` is equal to `accounting.getRequiredBondForNextKeys()`. `accounting.getRequiredBondForNextKeys()` CSAccounting.sol#L527 that relies on the current node operator's bond. The `CSModule.depositETH()` CSModule.sol#L471-L481 so anyone can increase the Node Operator's bond. A griefer may front-run a call to `accounting.getRequiredBondForNextKeys()` and deposit a few weis for 1 bond share. This would cause the initial call to revert. The griefer can do this multiple times, forcing the node operator manager to use another function for adding keys.
Additionally, a potential DoS vector exists if a user deposits 0 wei of ETH. This could block deposits of new keys initiated from the DSM contract because of the nonce update.
Recommendation

We recommend making CSModule.depositETH() permissioned so that only the Node Operator manager can call it. Additionally, we recommend adding a minimum limit for ETH deposit amounts and using private pools for deposits through the DSM contract.

Client's commentary

Check for `msg.value` changed to 'less than' via b17b4eb9. Described DoS attack vector is not applicable, since 0 wei deposits doesn't lead to a nonce change if there's no new depositable keys observed.

8. Griefing of CSModule.compensateELRewardsStealingPenalty()

Status

Fixed in 0e4b33a2

Description

CSModule.compensateELRewardsStealingPenalty() CSModule.sol#L1096-L1108. At the same time CSBondLock.sol#L126-L128 to compensate no more than what was locked. A griefer may frontrun a CSModule.compensateELRewardsStealingPenalty() call and compensate 1 locked share, causing the initial call to revert. The griefer can do this multiple times, putting the Node Operator manager at risk of having to settle their lock with a reset of the bond curve.

Recommendation

We recommend making the CSModule.compensateELRewardsStealingPenalty() function permissioned so that only the Node Operator manager can call it.

Client's commentary

Fixed via 1c9532ab

9. Incorrect Check For data.length in the AccountingOracle Contract

Status

Fixed in 1ffbb7e4

Description

The issue is identified within the AccountingOracle.sol#L741 function of the contract NodeOperatorsRegistry.

There is a check whether data.length is smaller than 67 bytes, which is incorrect, because its length should be 72 bytes minimum (nextHash (32 bytes) + itemIndex (3 bytes) + itemType (2 bytes) + itemPayload (moduleId (3 bytes) + nodeOpsCount (8 bytes) + nodeOperatorIds (8 bytes * nodeOpsCount) + validatorsCounts (16 bytes * nodeOpsCount))).

The issue is classified as **Medium** severity because it could lead to processing data in an incorrect format, despite the fact that it is provided in a permissioned way.

Recommendation

We recommend changing the check to data.length < 72 to disallow incorrect data processing

Client's commentary

The check has been completely removed in commit beb6ceb23cc3b54a74e74e25f1f9aeda65d04afe03e as it looks redundant. Incorrect data is checked further in the code in _processExtraDataItem

10. Potential Misallocation of Validators in _getDepositsAllocation Function

Status

ACKNOWLEDGED

Description

The issue is identified within the StakingRouter.sol#L1406-L1434 function of the contract StakingRouter. If a deposit is made before the system updates the status of stuck validators, there is a risk that funds could be allocated to a Node Operator (NO) that has violated an exit request. This can occur because the calculation uses availableValidatorsCount, which may not reflect the true status if stuck validators have not been accounted for yet.

The issue is classified as **Medium** severity because it could lead to the misallocation of validators, resulting in staking funds being deposited into a node operator that is not operating properly or has violated an exit request. The most probable scenario is when NO loses private keys for validators and cannot exit them. In this case, the protocol may allocate new deposits for such NO if the deposit is made before the stuck keys update.

Recommendation

We recommend implementing a mechanism to ensure that the status of all validators, especially stuck validators, is updated before deposits are made. This can be done by adding a flag to the AccountingOracle contract that prevents new deposits from being made after a new report is submitted until the stuck validators update is finalized.

Client's Commentary

It is assumed that operators in the curated modules receive only half of the rewards and do not receive a new stake if they do not exit keys in the time specified in the policy. This feature is intended to incentivize operators to exit keys on time, but is not a security tool to limit the new stake on an operator. To limit the stake on a particular operator, DAO has another tool: target limit.

2.4 Low

1. Inefficient Conversion Method in Bond Calculation

Status
Fixed in 0e4b33a2
Description
The issue is identified within the CSAccounting.sol#L535-L572 view functions of the contract CSAccounting. The function currently uses WSTETH.getWstETHByStETH() to convert stEth amount to shares. However, it would be more efficient to use _sharesByEth() instead of WSTETH.getWstETHByStETH() for the conversion process.
The issue is classified as **Low** severity because it does not impact the security or core functionality but could lead to inefficiencies.
Recommendation
We recommend replacing the use of WSTETH.getWstETHByStETH() with _sharesByEth() in the functions from the description to improve efficiency and consistency in bond calculations.
Client's commentary

Fixed via aafd9984

2. Lack of Validation for curveId in Constructor

Status
ACKNOWLEDGED
Description
The issue is identified within the CSEarlyAdoption.sol#L34 of the CSEarlyAdoption contract. While the curveId parameter is required and checked for being non-zero, there is no validation to ensure that the provided curveId is valid within the system. This could lead to the assignment of an invalid or non-existent bond curve, potentially causing issues during the contract's operation.
The issue is classified as **Low** severity because it does not immediately affect the contract's security or functionality but could lead to future issues if an invalid curveId is used.
Recommendation
We recommend adding a validation step to ensure that the curveId provided during the contract's deployment corresponds to a valid bond curve in the system.
Client's commentary

Deploy parameters and deployment itself are manually validated to ensure correctness.

3. Potential Front-Running Vulnerability in Reward Claim Process

Status
Fixed in 0e4b33a2
Description
The issue is identified within the CSModule.sol#L548-L554 function of the CSModule contract. Specifically, the process might be vulnerable to a front-running attack, where an attacker

could front-run the function to claim an unfair amount of funds. This could occur in scenarios where NO has stolen EL rewards and is punished for it but manages to front-run reporting of stealing and claims an unfair amount.
The issue is classified as **Low** severity because while this vulnerability could allow an attacker to claim more rewards than intended, it requires precise timing and specific conditions, limiting its overall impact.

Recommendation

We recommend using private pools for EL rewards stealing reporting.

Client's commentary

There is also a chance to claim rewards right before the block proposal if Node Operator is planning to steal MEV. Private pools will be used by the corresponding role member.

4. Lack of Checks for Service Addresses

Status

ACKNOWLEDGED

Description

The issue is identified within the CSModule contract. Specifically, there is a potential concern if the rewardAddress or managerAddress address of a Node Operator is one of the service addresses (CSAccounting for example). This could result in the user bond being reduced, and some funds becoming stuck in the accounting. Although this situation might seem acceptable under current conditions, it would be more robust to include appropriate checks during the initialization phase to prevent such scenarios.
The issue is classified as **Low** severity because the current implementation could lead to inefficiencies in fund management, though it does not directly compromise the security or functionality of the system.

Recommendation

We recommend adding validation checks during the Node Operator initialization phase to ensure that neither the rewardAddress nor the managerAddress address is one of the service addresses.

Client's commentary

Service addresses are not the only addresses that can cause issues if used as a manager or reward address. It is the user's responsibility to set the addresses correctly.
Misdirected funds can be recovered for the majority of the CSM contracts.

5. Lack of Existence Check for Node Operator ID

Status

ACKNOWLEDGED

Description

The issue is identified within the CSModule.sol#L1447 function of the contract. The function currently does not check whether the provided nodeOperatorId corresponds to an existing node operator, potentially leading to the retrieval of incorrect or uninitialized data if an invalid ID is queried. This oversight could result in unexpected behavior or errors when interacting with the contract. The same issue is present for CSBondCurve.sol#L75-L79 and CSBondCurve.sol#L84-L88 functions.

The issue is classified as **Low** severity because, while it might lead to incorrect data being returned and potentially misleading the users or administrators of the contract, it does not directly compromise the contract's security.

Recommendation

We recommend implementing a validation step at the beginning of the `getNodeOperatorSummary`, `getBondCurve` and `getBondCurveId` functions to ensure that the `nodeOperatorId` exists in the system before proceeding with the rest of the function.

Client's commentary

Since the methods mentioned are view methods, it is proposed that inputs be verified off-chain. All on-chain calls of the methods outlined already contain Node Operator existence validation.

6. Missing Validation of Key Removal Charge

Status

Fixed in `0e4b33a2`

Description

The issue is identified within the function `CSModule.sol#L1674-L1677` of the contract. This function currently allows setting any amount for the key removal charge without enforcing a limit. Without an upper boundary, excessively high charges could be set, potentially creating barriers for node operators needing to remove keys or manage their nodes effectively.

The issue is classified as **Low** severity because while it could lead to operational inefficiencies or financial burdens for Node Operators, it does not directly compromise the contract's security.

Recommendation

We recommend implementing a cap on the key removal charge to ensure it remains within reasonable limits.

Client's commentary

Fixed via 982e236d

7. Redundant Function Definition

Status

ACKNOWLEDGED

Description

The issue is identified within the `CSBondCore.sol#L275-L279` function of the `CSBondCore` contract. This function is redundant as it is overridden in the `CSAccounting` contract, suggesting that its definition in the current context might be unnecessary. This redundancy could lead to confusion and potential maintenance issues, as developers might overlook which version of the function is being called in different contexts.

The issue is classified as **Low** severity because it primarily affects the clarity and maintainability of the code rather than its functionality or security.

Recommendation

We recommend removing the `_getClaimableBondShares` function from the current contract to streamline the codebase and avoid confusion.

Client's commentary

This function is kept in the code base to ensure no issues in case of partial code reuse in the other projects that will decide not to use the CSBondCurve abstract contract.

8. Lack of Upper Limit Validation

Status
ACKNOWLEDGED
Description
The issue is identified within the CSBondCurve.sol#L59 of the CSBondCurve contract. While there is a check to ensure the maximum curve length is not below a minimum threshold, there is no upper limit enforced for the maximum curve length. This lack of upper-limit validation could potentially lead to excessively long bond curves being created.
The issue is classified as **Low** severity because it primarily affects the potential efficiency and operability of the contract rather than posing a direct security risk.
Recommendation
We recommend introducing a reasonable upper limit for the maximum curve length during the initialization of the contract.
Client's commentary

Deploy parameters and deployment itself are manually validated to ensure correctness.

9. Inflexibility in Contract Pause Duration Extension

Status
ACKNOWLEDGED
Description
The issue is identified within the PausableUntil.sol#L59-L70 function of the PausableUntil contract. Currently, the function does not allow for the prolongation of an existing pause duration. Once set, the pause duration is fixed until it expires. This lack of flexibility could lead to poor user experience, especially in scenarios where extending a pause without resuming the contract would be beneficial.
The issue is classified as **Low** severity because it pertains to the user experience and administrative flexibility rather than the core functionality or security of the contract.
Recommendation
We recommend enhancing the _pauseFor function to allow for the extension of the current pause duration.
Client's commentary

This functionality was borrowed from the core protocol code. It is preferable to keep it consistent with the existing protocol.

10. reportELRewardsStealingPenalty can be called with amount set to 0

Status
Fixed in 0e4b33a2
Description
The issue is identified within the CSModule.sol#L1014 function of the CSModule contract. The mentioned function can be called with amount set to 0. In that case, only

EL_REWARDS_STEALING_FINE is applied, and the ELRewardsStealingPenaltyReported event with stolenAmount == 0 is emitted.
The issue is classified as **Low** severity because it requires a mistake made by the REPORT_EL_REWARDS_STEALING_PENALTY_ROLE role owner.

Recommendation

We recommend restricting calls to the reportELRewardsStealingPenalty with the amount parameter set to 0.

Client's commentary

Fixed via 7c3df56c

11. ELRewardsStealingPenaltySettled event shouldn't be emitted in some cases

Status

Fixed in 0e4b33a2

Description

The issue is identified within the CSModule.sol#L1089 function of the CSModule contract. This function emits ELRewardsStealingPenaltySettled event even if the retention period for the locked bond has passed.
The issue is classified as **Low** severity because it may lead to an incorrect event being emitted when the penalty wasn't actually settled.

Recommendation

We recommend emitting the ELRewardsStealingPenaltySettled event only if lockedBondBefore > 0 is returned from the accounting.settleLockedBondETH(nodeOperatorId) call.

Client's commentary

Fixed via 143c4fb8

12. Unused AlreadyWithdrawn error

Status

Fixed in 0e4b33a2

Description

The issue is identified at the CSModule.sol#L155 in the CSModule contract. The mentioned error is unused despite the check in the CSModule.sol#L1131 function where AlreadySubmitted error is used instead.
The issue is classified as **Low** severity because there is a potentially misused error that doesn't affect protocol functionality.

Recommendation

We recommend removing the unused error or changing AlreadySubmitted error inside the submitWithdrawal function.

Client's commentary

Fixed via fd2e869a

13. _treeCid is not checked for duplicate value

Status
Fixed in 0e4b33a2
Description
The issue is identified within the CSFeeDistributor.sol#L129 function of the CSFeeDistributor contract. _treeCid should be checked against the previously stored value as it should also be changed when the tree root changes.
The issue is classified as **Low** severity because it is possible to start processing an oracle report with an unchanged treeCID.
Recommendation
We recommend adding a revert statement for cases when _treeCid == treeCid.
Client's commentary

Fixed via dca28fd0

14. Unused function _checkProcessingDeadline()

Status
Fixed in 0e4b33a2
Description
The issue is identified at the BaseOracle.sol#L410 in the BaseOracle contract. The _checkProcessingDeadline() function, which doesn't accept parameters, is never used.
The issue is classified as **Low** severity because there is an alternative function _checkProcessingDeadline(uint256 deadlineTime) which is used inside the _startProcessing function.
Recommendation
We recommend removing the mentioned unused function.
Client's commentary

Fixed via 19ebb5d5

15. Unnecessary modifier whenPaused

Status
Fixed in 0e4b33a2
Description
The issue is identified within the CSFeeOracle.sol#L139 function of the CSFeeOracle contract. There is an unnecessary whenPaused modifier used because there is a _checkPaused check inside the _resume function at the PausableUntil.sol#L54.
The issue is classified as **Low** severity because it doesn't affect the protocol functionality, but the unnecessary modifier removal can improve its efficiency.
Recommendation
We recommend removing the mentioned unnecessary modifier from the resume function.
Client's commentary

Fixed via a0dc831e

16. Possibility of overflow inside an unchecked block in CSBondLock._lock()

Status
Fixed in 0e4b33a2
Description
If CSModule.sol#L1014 is called by mistake with amount parameter close to type(uint256).max and the node operator already has locked bond, an overflow inside an unchecked block in CSBondLock._lock() may happen. So stored node operator's lock becomes less than it should be.
Recommendation
We recommend moving

```
if ($.bondLock[nodeOperatorId].retentionUntil > block.timestamp) {
    amount += $.bondLock[nodeOperatorId].amount;
}
```

outside of the unchecked block.
Client's commentary

Fixed via 583338b3

17. Using of a constant value instead of a constant name in CSModule.initialize()

Status
Fixed in 0e4b33a2
Description
_pauseFor(type(uint256).max); CSModule.sol#L206-L207 in CSModule.initialize() for an infinite pause. However, PausableUntil.PAUSE_INFINITELY is a special constant for this purpose.
Recommendation
We recommend changing
_pauseFor(type(uint256).max);
to
_pauseFor(PAUSE_INFINITELY);
in CSModule.initialize().
Client's commentary

Fixed via 5d822028

18. Missing Validation for Withdrawal Queue Limits

Status
ACKNOWLEDGED
Description
The issue is identified within the function CSBondCore.sol#L187-L190 of the contract CSBondCore. The current implementation does not validate the minimum and maximum limits imposed by the withdrawal queue when processing withdrawal requests. This omission can lead

to unexpected behavior, as withdrawal requests that fall outside these limits will be rejected.
The issue is classified as **Low** severity because it can disrupt the withdrawal process, leading to unexpected reverts for users.

Recommendation

We recommend implementing checks to ensure that the requested withdrawal amounts adhere to the withdrawal queue's minimum and maximum limits.

Client's commentary

Client: The described checks are already present in the corresponding WQ method: WithdrawalQueue.sol#L395-L402. Given that, the revert is inevitable. Hence, it is not clear what unexpected behavior can occur.
MixBytes: Reverting inside the `CSModule` contract is more straightforward for users, rather than passing incorrect withdrawal amounts to the `WithdrawalQueue` contract and reverting there.
Client: Revert in the external contract seems more appropriate than keeping the limits in two contracts in sync.

19. Missing Nonce Increment in `addNodeOperator` Function

Status

ACKNOWLEDGED

Description

The issue is identified within the function NodeOperatorsRegistry.sol#L317-L337 of the contract NodeOperatorRegistry. The current implementation does not increment the nonce within the addNodeOperator function, which is inconsistent with the behavior in related functions such as activateNodeOperator and deactivateNodeOperator. Although this omission does not impact the obtainDepositData function directly, it may lead to future issues with state tracking.

Recommendation

We recommend adding a nonce increment within the addNodeOperator function to ensure consistency across the contract and to prevent potential issues related to state tracking.

Client's Commentary

not issue
According to the description of the `getNonce()` method, its value should change only as a result of actions that modify the state of keys in the module. Adding an operator does not fall under such actions.
See: NodeOperatorsRegistry.sol#L1386

20. Potential Issue with Signature Verification in `pauseDeposits` Function

Status

ACKNOWLEDGED

Description

The issue is identified within the DepositSecurityModule.sol#L384 function of the contract DepositSecurityModule. The current implementation of the function relies on the ECDSA.recover method to verify the signature provided by the guardian. However, there is a very low probability that the recover function can return a random address if the signature is invalid, potentially allowing the check to pass unexpectedly. This could result in unauthorized parties influencing the decision to pause deposits, which could pose a security risk.

Recommendation

We recommend adding an index to the message that is signed by the guardian. This will ensure that each guardian's approval is uniquely identifiable and prevent the possibility of an invalid signature inadvertently passing the check.

Client's Commentary

The `ECDSA.recover` method eliminates the risk of signature reuse by recalculating it from the upper range of the elliptic curve. Additionally, it ensures that invalid `s`, `r`, or `v` values can't result in a zero address being returned, as can happen with EVM's `ecrecover`, thus guaranteeing signature validity.

The method also receives the hash of the message, which includes a prefix and block number, but restricted range of valid block numbers limiting the potential for message manipulation and craft alternative signatures that recover to the some valid address. Since the recovered address must be present in the contract's internal storage, the chance of randomly guessing a valid address is reduced to the extremely low probability of guessing a guardian's private key — an acceptable risk in practice within the crypto industry.

Therefore, adding the guardian index to the hashed message seems redundant.

21. Missing Validation of depositCalldata in depositBufferedEther Function

Status

ACKNOWLEDGED

Description

The issue is identified within the DepositSecurityModule.sol#L500 function of the contract DepositSecurityModule. The depositCalldata parameter is not included in the guardian signatures' validation process. Although depositCalldata is not currently used within staking modules, this omission could lead to potential security risks or inconsistencies in future implementations where depositCalldata might be utilized. An unchecked depositCalldata could allow unintended or malicious data to be processed, compromising the integrity of the deposit process.

Recommendation

We recommend either verifying that depositCalldata is empty within the current implementation or incorporating it into the message hash that is signed by the guardians. This will ensure that any future use of depositCalldata is secure and validated by the guardian signatures.

Client's Commentary

In the current design, the depositCalldata argument is the data might be necessary to prepare the deposit data by the staking module. Depending on the staking module implementation, it might be empty (NodeOperatorsRegistry) or contain complete deposit data (hypothetical staking module implementation that uses offchain storage for the deposit data). But for any staking module, the following is ALWAYS true: passed depositCalldata MUST NOT affect the deposit data set of the staking module. This requirement is explicitly defined in the comments of IStakingModule.obtainDepositData() method. 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. For example, the offchain staking module might store a hash of the expected depositCalldata onchain to validate it and reject any call with unexpected depositCalldata.

22. Potential Rounding Errors in _getDepositsAllocation Function

Status

ACKNOWLEDGED

Description

The issue is identified within the StakingRouter.sol#L1424 function of the contract StakingRouter. The current implementation could lead to rounding errors when calculating the allocation of deposits among staking modules. Specifically, the sum of capacities minus the sum of allocations may be less than the total deposits to allocate (_depositsToAllocate), which could result in some staking modules not receiving their full allocation of validators. This could cause inefficiencies or under-utilization of available validators.

Recommendation

We recommend accumulating any remaining "dust" (i.e., small amounts resulting from rounding errors) and assigning it to one of the staking modules. This will ensure that the total number of validators allocated matches the expected amount, maximizing the use of available validators.

Client's Commentary

The stake distribution logic enforces that a module's share doesn't exceed its stakeShareLimit. Since stake allocation is always an integer, determining the number of active validator keys, accounting for fractional "dust" at the end of the cycle will also yield an integer. This leftover stake may need to be allocated to a module, potentially exceeding its stakeShareLimit, or additional logic would be required to find a module with sufficient key reserve and stake capacity. Adding such logic increases complexity and gas costs, making deposit operations more expensive.

23. Lack of Index Validation in at Function

Status

ACKNOWLEDGED

Description

The issue is identified within the QueueLib.sol#L229-L234 function of the QueueLib library. The function currently does not validate whether the provided index falls within the allowable range between head and tail. This could result in the function returning incorrect data or even accessing uninitialized memory, leading to unpredictable behavior.

Recommendation

We recommend adding a check to ensure that the index falls within the range between self.head and self.tail. This will prevent out-of-bounds access and ensure that the function only returns valid data.

Client's Commentary

The ability to get old items is crucial for the historical purposes. Items with indexes higher that head should be 0, and it is assumed that requested indexes are validated off-chain or on the integration side.

24. Missing Checks for _pauseIntentValidityPeriodBlocks and _maxOperatorsPerUnvetting Values

Status

ACKNOWLEDGED

Description

The issue is identified within the DepositSecurityModule.sol#L108. There are two parameters, _pauseIntentValidityPeriodBlocks and _maxOperatorsPerUnvetting passed. These parameters are not checked for the maximum values.

The issue is classified as **Low** severity because it may lead to incorrect contract configuration.

Recommendation
We recommend adding checks to the functions `_setPauseIntentValidityPeriodBlocks` and `_setMaxOperatorsPerUnvetting` to ensure that the mentioned parameter values don't exceed acceptable boundaries.
Client's commentary

No issue: All variables passed to the constructor will be double-checked before and after deployment manually and via acceptance tests. In case of wrong parameters passed, there is always an option to redeploy a contract with correct values.

25. Unnecessary Inheritance

Status
ACKNOWLEDGED
Description
The issue is identified within the CSFeeOracle.sol#L17. `CSFeeOracle` contract inherits both from `AssetRecoverer` contract and `IAssetRecovererLib` interface, but `AssetRecoverer` contract already implements `IAssetRecovererLib` and there is no need for `CSFeeOracle` to follow the same interface.
Recommendation
We recommend removing unnecessary inheritance from the `IAssetRecovererLib` interface.
Client's commentary

`IAssetRecovererLib` encapsulate events and errors definitions. `CSFeeOracle` inherits `IAssetRecovererLib` to make sure the ABI for the contract contains all the errors and events from the interface. Other contracts with recovery functionality use inheritance on their interfaces level.

26. Incorrect Comment

Status
Fixed in cdc6e335
Description
The issue is identified within the BaseOracle.sol#L268 of the contract `BaseOracle`. There are duplicate list numbers in the comment and info about the check for report processing deadline, but it is not implemented.
The issue is classified as **Low** severity because this missing check doesn't affect contract functionality.
Recommendation
We recommend fixing the mentioned comment and removing info about the deadline check.
Client's commentary

Fixed via PR-331

27. Possible Multiple Signature Usage in DepositSecurityModule.depositBufferedEther()

Status
ACKNOWLEDGED
Description

In certain cases, it is possible to use the same signature to invoke DepositSecurityModule.sol#L494-L523 without a revert more than once. This can happen if the following conditions are met:

1. The amount of buffered ETH in Lido was 0 during the first invocation.
2. The depositRoot did not change during the minDepositBlockDistance blocks.

DepositSecurityModule.depositBufferedEther() is permissionless (in case of the correct set of signatures), so it may be a problem.

Recommendation

We recommend reverting the DepositSecurityModule.depositBufferedEther() call if there is no buffered ETH at the moment of invocation.

Client's commentary

It is assumed that a quorum of signatures confirms that the state of the deposit contract and module is checked and it is safe to perform a deposit to the module keys. Expectedly, if the module key state or the deposit_root have not changed, the signature remains valid. In case of an empty buffer, deposits do not pass and the signature is not invalidated.
Since the first version of DSM, there has indeed been a scenario to call the deposit method for an empty buffer. We have left it unchanged to maintain backward compatibility with off-chain tooling.

28. Lack of genesisTime Check in HashConsensus and BaseOracle Constructors

Status

ACKNOWLEDGED

Description

The constructors of HashConsensus (HashConsensus.sol#L240-L265, HashConsensus.sol#L269-L299) and BaseOracle (BaseOracle.sol#L103-L107, BaseOracle.sol#L129-L133) lack a validation check to ensure that genesisTime is less than or equal to block.timestamp. This omission could allow incorrect values for the genesisTime parameter to be passed during the deployment.

Recommendation

We recommend adding checks in the HashConsensus and BaseOracle constructors to ensure that genesisTime is less than or equal to block.timestamp.

Client's commentary

genesisTime is a critical parameter checked both by off-chain oracles and post-deployment/acceptance tests. Therefore we do not see the need for an additional check of this parameter in the constructors.

29. Missing Proof Length Check in getFeesToDistribute Function

Status

Fixed in cdc6e335

Description

The issue is identified within the CSFeeDistributor.sol#L175-L179 function of the contract CSFeeDistributor. The function currently does not validate the length of the Merkle proof provided in the proof parameter. As a result, there is a possibility that an empty or malformed proof could be passed, potentially allowing a user to craft a situation where they extract stETH shares without proper verification. This could lead to an unauthorized distribution of funds.

The issue is classified as **Low** severity because it requires to find shares amount that will generate a valid Merkle root. However, ensuring that the proof length is checked adds an

extra layer of security to prevent potential exploits.

Recommendation

We recommend adding a check to validate the length of the Merkle proof before verifying it.

Client's commentary

Fixed in PR-332

30. Chain Id Value is Used in the DepositSecurityModule Constructor

Status

ACKNOWLEDGED

Description

The issue is identified within the constructor of the contract DepositSecurityModule. The immutable variables ATTEST_MESSAGE_PREFIX, PAUSE_MESSAGE_PREFIX, and UNVET_MESSAGE_PREFIX values are calculated using block.chainid. There is a possibility of the chain id value changing due to potential modifications in the Ethereum chain. This may lead to replay attacks using signatures with the mentioned message prefixes.

The issue is classified as **Low** severity because it is unlikely that the chain id would change, and it is still better to follow EIP-712 domainSeparator calculation.

Recommendation

We recommend calculating the ATTEST_MESSAGE_PREFIX, PAUSE_MESSAGE_PREFIX, and UNVET_MESSAGE_PREFIX message prefixes at the time when the message hash is calculated. This will ensure the use of the actual chain id.

Client's commentary

This is indeed an extremely unlikely event, but in case it happens, it is assumed that the dev team will have enough time to prepare a deployment of the new DSM contract and switch to it, or initiate a deposit pause to mitigate a possible attack

31. Missing Nonce Increment in _updateRefundValidatorsKeysCount Function

Status

Fixed in 1ffbb7e4

Description

The issue is identified within the NodeOperatorsRegistry.sol#L744-L766 function of the contract NodeOperatorsRegistry. The current implementation lacks a call to the _increaseValidatorsKeysNonce() function when the REFUNDED_VALIDATORS_COUNT_OFFSET variable is modified. This is particularly important because the REFUNDED_VALIDATORS_COUNT_OFFSET variable can decrease, and failing to update the nonce could lead to discrepancies in the contract's state, affecting other functionalities that depend on accurate nonce tracking of depositable validators' updates.

The issue is classified as **Low** severity because it could result in inconsistencies in the contract's state, potentially affecting the integrity of validator key management and related operations.

Recommendation

We recommend adding a call to the _increaseValidatorsKeysNonce() function after updating the REFUNDED_VALIDATORS_COUNT_OFFSET variable to ensure that all state changes are properly tracked and the nonce remains consistent.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	10
Low	31

Findings Statuses

ID	Finding	Severity	Status
M-1	Inconsistent Bond Curve Update Handling	Medium	Acknowledged
M-2	Potential Mismanagement of Updated Withdrawal Vault Addresses	Medium	Fixed
M-3	Unchecked Guardian Quorum Value	Medium	Acknowledged
M-4	Reuse of Guardian Signatures for Pausing Deposits	Medium	Acknowledged
M-5	Bond Curve is Not Reset Inside the `submitInitialSlashing` Function	Medium	Acknowledged
M-6	Possible missing `_updateDepositableValidatorsCount` calls	Medium	Acknowledged
M-7	Griefing of `CSModule.addValidatorKeysETH()` with `CSModule.depositETH()`	Medium	Fixed
M-8	Griefing of `CSModule.compensateELRewardsStealingPenalty()`	Medium	Fixed
M-9	Incorrect Check For `data.length` in the `AccountingOracle` Contract	Medium	Fixed
M-10	Potential Misallocation of Validators in `_getDepositsAllocation` Function	Medium	Acknowledged

L-1	Inefficient Conversion Method in Bond Calculation	Low	Fixed
L-2	Lack of Validation for curveId in Constructor	Low	Acknowledged
L-3	Potential Front-Running Vulnerability in Reward Claim Process	Low	Fixed
L-4	Lack of Checks for Service Addresses	Low	Acknowledged
L-5	Lack of Existence Check for Node Operator ID	Low	Acknowledged
L-6	Missing Validation of Key Removal Charge	Low	Fixed
L-7	Redundant Function Definition	Low	Acknowledged
L-8	Lack of Upper Limit Validation	Low	Acknowledged
L-9	Inflexibility in Contract Pause Duration Extension	Low	Acknowledged
L-10	reportELRewardsStealingPenalty can be called with amount set to 0	Low	Fixed
L-11	ELRewardsStealingPenaltySettled event shouldn't be emitted in some cases	Low	Fixed
L-12	Unused AlreadyWithdrawn error	Low	Fixed
L-13	_treeCid is not checked for duplicate value	Low	Fixed
L-14	Unused function _checkProcessingDeadline()	Low	Fixed
L-15	Unnecessary modifier whenPaused	Low	Fixed
L-16	Possibility of overflow inside an unchecked block in CSBondLock._lock()	Low	Fixed
L-17	Using of a constant value instead of a constant name in CSModule.initialize()	Low	Fixed
L-18	Missing Validation for Withdrawal Queue Limits	Low	Acknowledged
L-19	Missing Nonce Increment in addNodeOperator Function	Low	Acknowledged
L-20	Potential Issue with Signature Verification in pauseDeposits Function	Low	Acknowledged

L-21	Missing Validation of depositCalldata in depositBufferedEther Function	Low	Acknowledged
L-22	Potential Rounding Errors in _getDepositsAllocation Function	Low	Acknowledged
L-23	Lack of Index Validation in at Function	Low	Acknowledged
L-24	Missing Checks for _pauseIntentValidityPeriodBlocks and _maxOperatorsPerUnvetting Values	Low	Acknowledged
L-25	Unnecessary Inheritance	Low	Acknowledged
L-26	Incorrect Comment	Low	Fixed
L-27	Possible Multiple Signature Usage in DepositSecurityModule.depositBufferedEther()	Low	Acknowledged
L-28	Lack of genesisTime Check in HashConsensus and BaseOracle Constructors	Low	Acknowledged
L-29	Missing Proof Length Check in getFeesToDistribute Function	Low	Fixed
L-30	Chain Id Value is Used in the DepositSecurityModule Constructor	Low	Acknowledged
L-31	Missing Nonce Increment in _updateRefundValidatorsKeysCount Function	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Inconsistent Bond Curve Update Handling		
Severity	Medium	**Status**	Acknowledged

Description
The issue is identified within the function CSAccounting.sol#L168-L173 of contract CSAccounting. When updating a bond curve, there is a potential issue where the unbonded validators amount increases for a specific NO (node operator), but the corresponding depositable amount is not updated accordingly. This inconsistency can lead to unexpected behavior in the system, as the depositable amount should be forced to update, but currently, it is not possible to enforce this.
The issue is classified as **Medium** severity because it can result in inconsistencies within the contract's logic, potentially leading to deposits of NO's keys without appropriate bond.
Recommendation
We recommend conducting bond curve update in a 2-step way, when NO's curve Id is updated on any action of this NO which will guarantee that unbonded keys and depositable keys update simultaneously.
Client's commentary

Making a two-step change might be complicated. Also, it is assumed that in most cases, the curve would be changed to lower values than the current one. However, should the curve be changed to higher values, the `normalizeQueue` method can be called individually for each NO with the change curve since the `normalizeQueue` method was changed to permissionless after the start of the audits.

M-2	Potential Mismanagement of Updated Withdrawal Vault Addresses		
Severity	Medium	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the function CSVerifier.sol#L308-L310 of the contract CSVerifier. The current implementation verifies that the withdrawalAddress matches WITHDRAWAL_ADDRESS. However, if the withdrawal vault address is updated through a contract upgrade, this check may fail for old deposits, leading to incorrect handling of those withdrawal processes.

The issue is classified as **Medium** severity because it can cause inconsistencies in withdrawal processing, potentially affecting users' funds and the contract's reliability.

Recommendation

We recommend implementing a mechanism to track historical withdrawal vault addresses or ensuring backward compatibility when updating the withdrawal vault address.

Client's commentary

It's possible to have multiple instances of `CSVerifier` attached via the `VERIFIER_ROLE` role to the `CSModule` with different `WITHDRAWAL_ADDRESS`'es configured. A user should just use the correct instance to bring its proof.

M-3	Unchecked Guardian Quorum Value		
Severity	Medium	**Status**	Acknowledged

Description

The issue is identified within the DepositSecurityModule.sol#L247-L254 function of the DepositSecurityModule contract. The function allows setting a guardian quorum value without checking if it exceeds the number of active guardians.

The issue is classified as **Medium** severity because it could lead to operational disruptions and governance issues if the quorum set is greater than the number of available guardians.

Recommendation

We recommend introducing a validation step within the _setGuardianQuorum function to ensure that the new quorum value does not exceed the current number of guardians.

Client's commentary

Changing the quorum can only be done through on-chain voting. All parameters are double-checked, acceptance tests are expected to be written for each vote, and any inflated quorum value can be corrected by subsequent voting. The absence of the check is intentional, and this behavior is documented in the code.

M-4	Reuse of Guardian Signatures for Pausing Deposits		
Severity	Medium	**Status**	Acknowledged

Description

The issue is identified within the DepositSecurityModule.sol#L384 function of the DepositSecurityModule contract. This function checks for guardians' signatures to pause deposits but does not utilize a nonce mechanism. Consequently, a previously used signature could potentially be replayed to initiate another pause, leading to abuse or unintended operational disruptions.

The issue is classified as **Medium** severity because it could allow an attacker to repeatedly pause the system by reusing an old, valid signature, thereby disrupting normal operations and potentially causing governance issues.

Recommendation

We recommend implementing a nonce system for signing operations. This will prevent the reuse of signatures and enhance the security and robustness of the contract.

Client's commentary

The DSM setup assumes that the unpausing of deposits can only be performed through on-chain DAO voting, the duration of which is significantly shorter than the expiration time of the pause intention.
Therefore, by the time the pause is lifted, all signed pause intentions will have expired.

M-5	Bond Curve is Not Reset Inside the submitInitialSlashing Function		
Severity	Medium	**Status**	Acknowledged

Description

The issue is identified within the CSModule.sol#L1174 function of the CSModule contract, where the initial slashing penalty is applied to the Node Operator bond. While this penalization occurs, the Bond Curve for that Node Operator is not reset to the default. The issue is classified as **Medium** severity because it may lead to incorrect accounting of the unbonded validator keys count after the Node Operator penalization.

Recommendation

We recommend resetting the Node Operator bond curve during the mentioned penalization event via a call to accounting.resetBondCurve(nodeOperatorId).

Client's commentary

The corresponding bond curve reset is done within the submitWithdrawal method CSModule.sol#L1139 It is done intentionally to prevent unnecessary unbonded validators that might occur after slashing reporting but before withdrawal reporting that will reduce totalKeysCount used to calculate required bond

M-6	Possible missing _updateDepositableValidatorsCount calls		
Severity	Medium	**Status**	Acknowledged

Description

The issue is identified within the CSAccounting.sol#L178 and CSAccounting.sol#L188 functions of the CSAccounting contract.

It is expected that the mentioned functions may be called by an external trusted entity like DAO. The functions change Node Operator bond curve which may lead to unbonded validator keys count change.

The issue is classified as **Medium** severity because it may lead to incorrect accounting of unbonded validator keys count after the Node Operator bond curve update.

Recommendation

We recommend making the calls to setBondCurve and resetBondCurve functions only from CSModule contract. Such calls should be followed by the _updateDepositableValidatorsCount calls which will update depositable validators count. In the future, when the DAO would be permitted to call the mentioned functions, the entry point for such calls would be inside the CSModule contract with all necessary updates.

Client's commentary

Similarly to the finding #1 it is proposed to call `normalizeQueue` after calling `setBondCurve` and `resetBondCurve` externally.

M-7	Griefing of CSModule.addValidatorKeysETH() with CSModule.depositETH()		
Severity	Medium	Status	Fixed in 0e4b33a2

Description

CSModule.addValidatorKeysETH() CSModule.sol#L388-L393 that msg.value is equal to accounting.getRequiredBondForNextKeys(). accounting.getRequiredBondForNextKeys() CSAccounting.sol#L527 that relies on the current node operator's bond. The CSModule.depositETH() CSModule.sol#L471-L481 so anyone can increase the Node Operator's bond. A griefer may front-run a call to accounting.getRequiredBondForNextKeys() and deposit a few weis for 1 bond share. This would cause the initial call to revert. The griefer can do this multiple times, forcing the node operator manager to use another function for adding keys.

Additionally, a potential DoS vector exists if a user deposits 0 wei of ETH. This could block deposits of new keys initiated from the DSM contract because of the nonce update.

Recommendation

We recommend making CSModule.depositETH() permissioned so that only the Node Operator manager can call it. Additionally, we recommend adding a minimum limit for ETH deposit amounts and using private pools for deposits through the DSM contract.

Client's commentary

Check for `msg.value` changed to 'less than' via b17b4eb9. Described DoS attack vector is not applicable, since 0 wei deposits doesn't lead to a nonce change if there's no new depositable keys observed.

M-8	Griefing of CSModule.compensateELRewardsStealingPenalty()		
Severity	Medium	**Status**	Fixed in 0e4b33a2

Description

CSModule.compensateELRewardsStealingPenalty() CSModule.sol#L1096-L1108. At the same time CSBondLock.sol#L126-L128 to compensate no more than what was locked. A griefer may frontrun a CSModule.compensateELRewardsStealingPenalty() call and compensate 1 locked share, causing the initial call to revert. The griefer can do this multiple times, putting the Node Operator manager at risk of having to settle their lock with a reset of the bond curve.

Recommendation

We recommend making the CSModule.compensateELRewardsStealingPenalty() function permissioned so that only the Node Operator manager can call it.

Client's commentary

Fixed via 1c9532ab

M-9	Incorrect Check For data.length in the AccountingOracle Contract		
Severity	Medium	**Status**	Fixed in 1ffbb7e4

Description

The issue is identified within the AccountingOracle.sol#L741 function of the contract NodeOperatorsRegistry.

There is a check whether data.length is smaller than 67 bytes, which is incorrect, because its length should be 72 bytes minimum (nextHash (32 bytes) + itemIndex (3 bytes) + itemType (2 bytes) + itemPayload (moduleId (3 bytes) + nodeOpsCount (8 bytes) + nodeOperatorIds (8 bytes * nodeOpsCount) + validatorsCounts (16 bytes * nodeOpsCount))).

The issue is classified as **Medium** severity because it could lead to processing data in an incorrect format, despite the fact that it is provided in a permissioned way.

Recommendation

We recommend changing the check to data.length < 72 to disallow incorrect data processing

Client's commentary

The check has been completely removed in commit beb6ceb23cc3b54a74e74e25f1f9aeda65d04afe03e as it looks redundant. Incorrect data is checked further in the code in _processExtraDataItem

M-10	Potential Misallocation of Validators in _getDepositsAllocation Function		
Severity	Medium	**Status**	Acknowledged

Description

The issue is identified within the StakingRouter.sol#L1406-L1434 function of the contract StakingRouter. If a deposit is made before the system updates the status of stuck validators, there is a risk that funds could be allocated to a Node Operator (NO) that has violated an exit request. This can occur because the calculation uses availableValidatorsCount, which may not reflect the true status if stuck validators have not been accounted for yet.

The issue is classified as **Medium** severity because it could lead to the misallocation of validators, resulting in staking funds being deposited into a node operator that is not operating properly or has violated an exit request. The most probable scenario is when NO loses private keys for validators and cannot exit them. In this case, the protocol may allocate new deposits for such NO if the deposit is made before the stuck keys update.

Recommendation

We recommend implementing a mechanism to ensure that the status of all validators, especially stuck validators, is updated before deposits are made. This can be done by adding a flag to the AccountingOracle contract that prevents new deposits from being made after a new report is submitted until the stuck validators update is finalized.

Client's Commentary

It is assumed that operators in the curated modules receive only half of the rewards and do not receive a new stake if they do not exit keys in the time specified in the policy. This feature is intended to incentivize operators to exit keys on time, but is not a security tool to limit the new stake on an operator. To limit the stake on a particular operator, DAO has another tool: target limit.

2.4 Low

L-1	Inefficient Conversion Method in Bond Calculation		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the CSAccounting.sol#L535-L572 view functions of the contract CSAccounting. The function currently uses WSTETH.getWstETHByStETH() to convert stEth amount to shares. However, it would be more efficient to use _sharesByEth() instead of WSTETH.getWstETHByStETH() for the conversion process.

The issue is classified as **Low** severity because it does not impact the security or core functionality but could lead to inefficiencies.

Recommendation

We recommend replacing the use of WSTETH.getWstETHByStETH() with _sharesByEth() in the functions from the description to improve efficiency and consistency in bond calculations.

Client's commentary

Fixed via aafd9984

L-2	Lack of Validation for curveId in Constructor		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the CSEarlyAdoption.sol#L34 of the CSEarlyAdoption contract. While the curveId parameter is required and checked for being non-zero, there is no validation to ensure that the provided curveId is valid within the system. This could lead to the assignment of an invalid or non-existent bond curve, potentially causing issues during the contract's operation.

The issue is classified as **Low** severity because it does not immediately affect the contract's security or functionality but could lead to future issues if an invalid curveId is used.

Recommendation

We recommend adding a validation step to ensure that the curveId provided during the contract's deployment corresponds to a valid bond curve in the system.

Client's commentary

Deploy parameters and deployment itself are manually validated to ensure correctness.

L-3	Potential Front-Running Vulnerability in Reward Claim Process		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the CSModule.sol#L548-L554 function of the CSModule contract. Specifically, the process might be vulnerable to a front-running attack, where an attacker could front-run the function to claim an unfair amount of funds. This could occur in scenarios where NO has stolen EL rewards and is punished for it but manages to front-run reporting of stealing and claims an unfair amount.

The issue is classified as **Low** severity because while this vulnerability could allow an attacker to claim more rewards than intended, it requires precise timing and specific conditions, limiting its overall impact.

Recommendation

We recommend using private pools for EL rewards stealing reporting.

Client's commentary

There is also a chance to claim rewards right before the block proposal if Node Operator is planning to steal MEV. Private pools will be used by the corresponding role member.

L-4	Lack of Checks for Service Addresses		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the `CSModule` contract. Specifically, there is a potential concern if the `rewardAddress` or `managerAddress` address of a Node Operator is one of the service addresses (`CSAccounting` for example). This could result in the user bond being reduced, and some funds becoming stuck in the accounting. Although this situation might seem acceptable under current conditions, it would be more robust to include appropriate checks during the initialization phase to prevent such scenarios.

The issue is classified as **Low** severity because the current implementation could lead to inefficiencies in fund management, though it does not directly compromise the security or functionality of the system.

Recommendation

We recommend adding validation checks during the Node Operator initialization phase to ensure that neither the `rewardAddress` nor the `managerAddress` address is one of the service addresses.

Client's commentary

Service addresses are not the only addresses that can cause issues if used as a manager or reward address. It is the user's responsibility to set the addresses correctly.

Misdirected funds can be recovered for the majority of the CSM contracts.

L-5	Lack of Existence Check for Node Operator ID		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the CSModule.sol#L1447 function of the contract. The function currently does not check whether the provided nodeOperatorId corresponds to an existing node operator, potentially leading to the retrieval of incorrect or uninitialized data if an invalid ID is queried. This oversight could result in unexpected behavior or errors when interacting with the contract. The same issue is present for CSBondCurve.sol#L75-L79 and CSBondCurve.sol#L84-L88 functions.

The issue is classified as **Low** severity because, while it might lead to incorrect data being returned and potentially misleading the users or administrators of the contract, it does not directly compromise the contract's security.

Recommendation

We recommend implementing a validation step at the beginning of the getNodeOperatorSummary, getBondCurve and getBondCurveId functions to ensure that the nodeOperatorId exists in the system before proceeding with the rest of the function.

Client's commentary

Since the methods mentioned are view methods, it is proposed that inputs be verified off-chain. All on-chain calls of the methods outlined already contain Node Operator existence validation.

L-6	Missing Validation of Key Removal Charge		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the function CSModule.sol#L1674-L1677 of the contract. This function currently allows setting any amount for the key removal charge without enforcing a limit. Without an upper boundary, excessively high charges could be set, potentially creating barriers for node operators needing to remove keys or manage their nodes effectively.

The issue is classified as **Low** severity because while it could lead to operational inefficiencies or financial burdens for Node Operators, it does not directly compromise the contract's security.

Recommendation

We recommend implementing a cap on the key removal charge to ensure it remains within reasonable limits.

Client's commentary

Fixed via 982e236d

L-7	Redundant Function Definition		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the CSBondCore.sol#L275-L279 function of the CSBondCore contract. This function is redundant as it is overridden in the CSAccounting contract, suggesting that its definition in the current context might be unnecessary. This redundancy could lead to confusion and potential maintenance issues, as developers might overlook which version of the function is being called in different contexts.

The issue is classified as **Low** severity because it primarily affects the clarity and maintainability of the code rather than its functionality or security.

Recommendation

We recommend removing the _getClaimableBondShares function from the current contract to streamline the codebase and avoid confusion.

Client's commentary

This function is kept in the code base to ensure no issues in case of partial code reuse in the other projects that will decide not to use the CSBondCurve abstract contract.

L-8	Lack of Upper Limit Validation		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the CSBondCurve.sol#L59 of the CSBondCurve contract. While there is a check to ensure the maximum curve length is not below a minimum threshold, there is no upper limit enforced for the maximum curve length. This lack of upper-limit validation could potentially lead to excessively long bond curves being created.

The issue is classified as **Low** severity because it primarily affects the potential efficiency and operability of the contract rather than posing a direct security risk.

Recommendation

We recommend introducing a reasonable upper limit for the maximum curve length during the initialization of the contract.

Client's commentary

Deploy parameters and deployment itself are manually validated to ensure correctness.

L-9	Inflexibility in Contract Pause Duration Extension		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the PausableUntil.sol#L59-L70 function of the PausableUntil contract. Currently, the function does not allow for the prolongation of an existing pause duration. Once set, the pause duration is fixed until it expires. This lack of flexibility could lead to poor user experience, especially in scenarios where extending a pause without resuming the contract would be beneficial.

The issue is classified as **Low** severity because it pertains to the user experience and administrative flexibility rather than the core functionality or security of the contract.

Recommendation

We recommend enhancing the _pauseFor function to allow for the extension of the current pause duration.

Client's commentary

This functionality was borrowed from the core protocol code. It is preferable to keep it consistent with the existing protocol.

L-10	reportELRewardsStealingPenalty can be called with amount set to 0		
Severity	Low	Status	Fixed in 0e4b33a2

Description

The issue is identified within the CSModule.sol#L1014 function of the CSModule contract. The mentioned function can be called with amount set to 0. In that case, only EL_REWARDS_STEALING_FINE is applied, and the ELRewardsStealingPenaltyReported event with stolenAmount == 0 is emitted.

The issue is classified as **Low** severity because it requires a mistake made by the REPORT_EL_REWARDS_STEALING_PENALTY_ROLE role owner.

Recommendation

We recommend restricting calls to the reportELRewardsStealingPenalty with the amount parameter set to 0.

Client's commentary

Fixed via 7c3df56c

L-11	ELRewardsStealingPenaltySettled event shouldn't be emitted in some cases		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the CSModule.sol#L1089 function of the CSModule contract. This function emits ELRewardsStealingPenaltySettled event even if the retention period for the locked bond has passed.

The issue is classified as **Low** severity because it may lead to an incorrect event being emitted when the penalty wasn't actually settled.

Recommendation

We recommend emitting the ELRewardsStealingPenaltySettled event only if lockedBondBefore > 0 is returned from the accounting.settleLockedBondETH(nodeOperatorId) call.

Client's commentary

Fixed via 143c4fb8

L-12	Unused AlreadyWithdrawn error		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified at the CSModule.sol#L155 in the CSModule contract. The mentioned error is unused despite the check in the CSModule.sol#L1131 function where AlreadySubmitted error is used instead.

The issue is classified as **Low** severity because there is a potentially misused error that doesn't affect protocol functionality.

Recommendation

We recommend removing the unused error or changing AlreadySubmitted error inside the submitWithdrawal function.

Client's commentary

Fixed via fd2e869a

L-13	_treeCid is not checked for duplicate value		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the CSFeeDistributor.sol#L129 function of the CSFeeDistributor contract. _treeCid should be checked against the previously stored value as it should also be changed when the tree root changes.

The issue is classified as **Low** severity because it is possible to start processing an oracle report with an unchanged treeCID.

Recommendation

We recommend adding a revert statement for cases when _treeCid == treeCid.

Client's commentary

Fixed via dca28fd0

L-14	Unused function _checkProcessingDeadline()		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified at the BaseOracle.sol#L410 in the BaseOracle contract. The _checkProcessingDeadline() function, which doesn't accept parameters, is never used. The issue is classified as **Low** severity because there is an alternative function _checkProcessingDeadline(uint256 deadlineTime) which is used inside the _startProcessing function.

Recommendation

We recommend removing the mentioned unused function.

Client's commentary

Fixed via 19ebb5d5

L-15	Unnecessary modifier whenPaused		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

The issue is identified within the CSFeeOracle.sol#L139 function of the CSFeeOracle contract. There is an unnecessary whenPaused modifier used because there is a _checkPaused check inside the _resume function at the PausableUntil.sol#L54.

The issue is classified as **Low** severity because it doesn't affect the protocol functionality, but the unnecessary modifier removal can improve its efficiency.

Recommendation

We recommend removing the mentioned unnecessary modifier from the resume function.

Client's commentary

Fixed via a0dc831e

L-16	Possibility of overflow inside an unchecked block in CSBondLock._lock()		
Severity	Low	**Status**	Fixed in 0e4b33a2

Description

If CSModule.sol#L1014 is called by mistake with amount parameter close to type(uint256).max and the node operator already has locked bond, an overflow inside an unchecked block in CSBondLock._lock() may happen. So stored node operator's lock becomes less than it should be.

Recommendation

We recommend moving

```
if ($.bondLock[nodeOperatorId].retentionUntil > block.timestamp) {
    amount += $.bondLock[nodeOperatorId].amount;
}
```

outside of the unchecked block.

Client's commentary

Fixed via 583338b3

L-17	Using of a constant value instead of a constant name in CSModule.initialize()		
Severity	Low	Status	Fixed in 0e4b33a2

Description

_pauseFor(type(uint256).max); CSModule.sol#L206-L207 in CSModule.initialize() for an infinite pause. However, PausableUntil.PAUSE_INFINITELY is a special constant for this purpose.

Recommendation

We recommend changing

_pauseFor(type(uint256).max);

to

_pauseFor(PAUSE_INFINITELY);

in CSModule.initialize().

Client's commentary

Fixed via 5d822028

L-18	Missing Validation for Withdrawal Queue Limits		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the function CSBondCore.sol#L187-L190 of the contract CSBondCore. The current implementation does not validate the minimum and maximum limits imposed by the withdrawal queue when processing withdrawal requests. This omission can lead to unexpected behavior, as withdrawal requests that fall outside these limits will be rejected.

The issue is classified as **Low** severity because it can disrupt the withdrawal process, leading to unexpected reverts for users.

Recommendation

We recommend implementing checks to ensure that the requested withdrawal amounts adhere to the withdrawal queue's minimum and maximum limits.

Client's commentary

Client: The described checks are already present in the corresponding WQ method: WithdrawalQueue.sol#L395-L402. Given that, the revert is inevitable. Hence, it is not clear what unexpected behavior can occur.

MixBytes: Reverting inside the `CSModule` contract is more straightforward for users, rather than passing incorrect withdrawal amounts to the `WithdrawalQueue` contract and reverting there.

Client: Revert in the external contract seems more appropriate than keeping the limits in two contracts in sync.

L-19	Missing Nonce Increment in addNodeOperator Function		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the function NodeOperatorsRegistry.sol#L317-L337 of the contract NodeOperatorRegistry. The current implementation does not increment the nonce within the addNodeOperator function, which is inconsistent with the behavior in related functions such as activateNodeOperator and deactivateNodeOperator. Although this omission does not impact the obtainDepositData function directly, it may lead to future issues with state tracking.

Recommendation

We recommend adding a nonce increment within the addNodeOperator function to ensure consistency across the contract and to prevent potential issues related to state tracking.

Client's Commentary

not issue

According to the description of the `getNonce()` method, its value should change only as a result of actions that modify the state of keys in the module. Adding an operator does not fall under such actions.

See: NodeOperatorsRegistry.sol#L1386

L-20	Potential Issue with Signature Verification in pauseDeposits Function		
Severity	Low	Status	Acknowledged

Description

The issue is identified within the DepositSecurityModule.sol#L384 function of the contract DepositSecurityModule. The current implementation of the function relies on the ECDSA.recover method to verify the signature provided by the guardian. However, there is a very low probability that the recover function can return a random address if the signature is invalid, potentially allowing the check to pass unexpectedly. This could result in unauthorized parties influencing the decision to pause deposits, which could pose a security risk.

Recommendation

We recommend adding an index to the message that is signed by the guardian. This will ensure that each guardian's approval is uniquely identifiable and prevent the possibility of an invalid signature inadvertently passing the check.

Client's Commentary

The `ECDSA.recover` method eliminates the risk of signature reuse by recalculating it from the upper range of the elliptic curve. Additionally, it ensures that invalid `s`, `r`, or `v` values can't result in a zero address being returned, as can happen with EVM's `ecrecover`, thus guaranteeing signature validity.

The method also receives the hash of the message, which includes a prefix and block number, but restricted range of valid block numbers limiting the potential for message manipulation and craft alternative signatures that recover to the some valid address. Since the recovered address must be present in the contract's internal storage, the chance of randomly guessing a valid address is reduced to the extremely low probability of guessing a guardian's private key — an acceptable risk in practice within the crypto industry.

Therefore, adding the guardian index to the hashed message seems redundant.

L-21	Missing Validation of depositCalldata in depositBufferedEther Function		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the DepositSecurityModule.sol#L500 function of the contract DepositSecurityModule. The depositCalldata parameter is not included in the guardian signatures' validation process. Although depositCalldata is not currently used within staking modules, this omission could lead to potential security risks or inconsistencies in future implementations where depositCalldata might be utilized. An unchecked depositCalldata could allow unintended or malicious data to be processed, compromising the integrity of the deposit process.

Recommendation

We recommend either verifying that depositCalldata is empty within the current implementation or incorporating it into the message hash that is signed by the guardians. This will ensure that any future use of depositCalldata is secure and validated by the guardian signatures.

Client's Commentary

In the current design, the depositCalldata argument is the data might be necessary to prepare the deposit data by the staking module. Depending on the staking module implementation, it might be empty (NodeOperatorsRegistry) or contain complete deposit data (hypothetical staking module implementation that uses offchain storage for the deposit data). But for any staking module, the following is ALWAYS true: passed depositCalldata MUST NOT affect the deposit data set of the staking module. This requirement is explicitly defined in the comments of IStakingModule.obtainDepositData() method. 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. For example, the offchain staking module might store a hash of the expected depositCalldata onchain to validate it and reject any call with unexpected depositCalldata.

L-22	Potential Rounding Errors in _getDepositsAllocation Function		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the StakingRouter.sol#L1424 function of the contract StakingRouter. The current implementation could lead to rounding errors when calculating the allocation of deposits among staking modules. Specifically, the sum of capacities minus the sum of allocations may be less than the total deposits to allocate (_depositsToAllocate), which could result in some staking modules not receiving their full allocation of validators. This could cause inefficiencies or under-utilization of available validators.

Recommendation

We recommend accumulating any remaining "dust" (i.e., small amounts resulting from rounding errors) and assigning it to one of the staking modules. This will ensure that the total number of validators allocated matches the expected amount, maximizing the use of available validators.

Client's Commentary

The stake distribution logic enforces that a module's share doesn't exceed its stakeShareLimit. Since stake allocation is always an integer, determining the number of active validator keys, accounting for fractional "dust" at the end of the cycle will also yield an integer. This leftover stake may need to be allocated to a module, potentially exceeding its stakeShareLimit, or additional logic would be required to find a module with sufficient key reserve and stake capacity. Adding such logic increases complexity and gas costs, making deposit operations more expensive.

L-23	Lack of Index Validation in at Function		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the QueueLib.sol#L229-L234 function of the QueueLib library. The function currently does not validate whether the provided index falls within the allowable range between head and tail. This could result in the function returning incorrect data or even accessing uninitialized memory, leading to unpredictable behavior.

Recommendation

We recommend adding a check to ensure that the index falls within the range between self.head and self.tail. This will prevent out-of-bounds access and ensure that the function only returns valid data.

Client's Commentary

The ability to get old items is crucial for the historical purposes. Items with indexes higher that head should be 0, and it is assumed that requested indexes are validated off-chain or on the integration side.

L-24	Missing Checks for _pauseIntentValidityPeriodBlocks and _maxOperatorsPerUnvetting Values		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the DepositSecurityModule.sol#L108. There are two parameters, _pauseIntentValidityPeriodBlocks and _maxOperatorsPerUnvetting passed. These parameters are not checked for the maximum values.

The issue is classified as **Low** severity because it may lead to incorrect contract configuration.

Recommendation

We recommend adding checks to the functions _setPauseIntentValidityPeriodBlocks and _setMaxOperatorsPerUnvetting to ensure that the mentioned parameter values don't exceed acceptable boundaries.

Client's commentary

No issue: All variables passed to the constructor will be double-checked before and after deployment manually and via acceptance tests. In case of wrong parameters passed, there is always an option to redeploy a contract with correct values.

L-25	Unnecessary Inheritance		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the CSFeeOracle.sol#L17. CSFeeOracle contract inherits both from AssetRecoverer contract and IAssetRecovererLib interface, but AssetRecoverer contract already implements IAssetRecovererLib and there is no need for CSFeeOracle to follow the same interface.

Recommendation

We recommend removing unnecessary inheritance from the IAssetRecovererLib interface.

Client's commentary

IAssetRecovererLib encapsulate events and errors definitions. CSFeeOracle inherits IAssetRecovererLib to make sure the ABI for the contract contains all the errors and events from the interface. Other contracts with recovery functionality use inheritance on their interfaces level.

L-26	Incorrect Comment		
Severity	Low	**Status**	Fixed in cdc6e335

Description

The issue is identified within the BaseOracle.sol#L268 of the contract BaseOracle. There are duplicate list numbers in the comment and info about the check for report processing deadline, but it is not implemented.

The issue is classified as **Low** severity because this missing check doesn't affect contract functionality.

Recommendation

We recommend fixing the mentioned comment and removing info about the deadline check.

Client's commentary

Fixed via PR-331

L-27	Possible Multiple Signature Usage in DepositSecurityModule.depositBufferedEther()		
Severity	Low	**Status**	Acknowledged

Description

In certain cases, it is possible to use the same signature to invoke DepositSecurityModule.sol#L494-L523 without a revert more than once. This can happen if the following conditions are met:

1. The amount of buffered ETH in Lido was 0 during the first invocation.
2. The depositRoot did not change during the minDepositBlockDistance blocks.

DepositSecurityModule.depositBufferedEther() is permissionless (in case of the correct set of signatures), so it may be a problem.

Recommendation

We recommend reverting the DepositSecurityModule.depositBufferedEther() call if there is no buffered ETH at the moment of invocation.

Client's commentary

It is assumed that a quorum of signatures confirms that the state of the deposit contract and module is checked and it is safe to perform a deposit to the module keys. Expectedly, if the module key state or the deposit_root have not changed, the signature remains valid. In case of an empty buffer, deposits do not pass and the signature is not invalidated.
Since the first version of DSM, there has indeed been a scenario to call the deposit method for an empty buffer. We have left it unchanged to maintain backward compatibility with off-chain tooling.

L-28	Lack of genesisTime Check in HashConsensus and BaseOracle Constructors		
Severity	Low	**Status**	Acknowledged

Description

The constructors of HashConsensus (HashConsensus.sol#L240-L265, HashConsensus.sol#L269-L299) and BaseOracle (BaseOracle.sol#L103-L107, BaseOracle.sol#L129-L133) lack a validation check to ensure that genesisTime is less than or equal to block.timestamp. This omission could allow incorrect values for the genesisTime parameter to be passed during the deployment.

Recommendation

We recommend adding checks in the HashConsensus and BaseOracle constructors to ensure that genesisTime is less than or equal to block.timestamp.

Client's commentary

genesisTime is a critical parameter checked both by off-chain oracles and post-deployment/acceptance tests. Therefore we do not see the need for an additional check of this parameter in the constructors.

L-29	Missing Proof Length Check in getFeesToDistribute Function		
Severity	Low	**Status**	Fixed in cdc6e335

Description

The issue is identified within the CSFeeDistributor.sol#L175-L179 function of the contract CSFeeDistributor. The function currently does not validate the length of the Merkle proof provided in the proof parameter. As a result, there is a possibility that an empty or malformed proof could be passed, potentially allowing a user to craft a situation where they extract stETH shares without proper verification. This could lead to an unauthorized distribution of funds.

The issue is classified as **Low** severity because it requires to find shares amount that will generate a valid Merkle root. However, ensuring that the proof length is checked adds an extra layer of security to prevent potential exploits.

Recommendation

We recommend adding a check to validate the length of the Merkle proof before verifying it.

Client's commentary

Fixed in PR-332

L-30	Chain Id Value is Used in the DepositSecurityModule Constructor		
Severity	Low	**Status**	Acknowledged

Description

The issue is identified within the constructor of the contract DepositSecurityModule. The immutable variables ATTEST_MESSAGE_PREFIX, PAUSE_MESSAGE_PREFIX, and UNVET_MESSAGE_PREFIX values are calculated using block.chainid. There is a possibility of the chain id value changing due to potential modifications in the Ethereum chain. This may lead to replay attacks using signatures with the mentioned message prefixes.

The issue is classified as **Low** severity because it is unlikely that the chain id would change, and it is still better to follow EIP-712 domainSeparator calculation.

Recommendation

We recommend calculating the ATTEST_MESSAGE_PREFIX, PAUSE_MESSAGE_PREFIX, and UNVET_MESSAGE_PREFIX message prefixes at the time when the message hash is calculated. This will ensure the use of the actual chain id.

Client's commentary

This is indeed an extremely unlikely event, but in case it happens, it is assumed that the dev team will have enough time to prepare a deployment of the new DSM contract and switch to it, or initiate a deposit pause to mitigate a possible attack

L-31	Missing Nonce Increment in _updateRefundValidatorsKeysCount Function		
Severity	Low	**Status**	Fixed in 1ffbb7e4

Description

The issue is identified within the NodeOperatorsRegistry.sol#L744-L766 function of the contract NodeOperatorsRegistry. The current implementation lacks a call to the _increaseValidatorsKeysNonce() function when the REFUNDED_VALIDATORS_COUNT_OFFSET variable is modified. This is particularly important because the REFUNDED_VALIDATORS_COUNT_OFFSET variable can decrease, and failing to update the nonce could lead to discrepancies in the contract's state, affecting other functionalities that depend on accurate nonce tracking of depositable validators' updates.

The issue is classified as **Low** severity because it could result in inconsistencies in the contract's state, potentially affecting the integrity of validator key management and related operations.

Recommendation

We recommend adding a call to the _increaseValidatorsKeysNonce() function after updating the REFUNDED_VALIDATORS_COUNT_OFFSET variable to ensure that all state changes are properly tracked and the nonce remains consistent.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

 https://mixbytes.io/

 https://github.com/mixbytes/audits_public

 hello@mixbytes.io

 https://x.com/mixbytes