# LIDO V2. ON-CHAIN AUDIT REPORT

0XORIO

MAY 10, 2023

# CONTENTS

# 1 INTRO

# 1.1 DISCLAIMER

The audit makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

# 1.2 ABOUT OXORIO

Oxorio is a young but rapidly growing audit and consulting company in the field of the blockchain industry, providing consulting and security audits for organizations from all over the world. Oxorio has participated in multiple blockchain projects during which smart contract systems were designed and deployed by the company.

Oxorio is the creator, maintainer, and major contributor of several blockchain projects and employs more than 5 blockchain specialists to analyze and develop smart contracts.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

# 1.3 SECURITY ASSESSMENT METHODOLOGY

A group of auditors is involved in the work on this audit. Each of them checks the provided source code independently of each other in accordance with the security assessment methodology described below:

**1. Project architecture review**

Study the source code manually to find errors and bugs.

**2. Check the code for known vulnerabilities from the list**

Conduct a verification process of the code against the constantly updated list of already known vulnerabilities maintained by the company.

**3. Architecture and structure check of the security model**

Study the project documentation and its comparison against the code including the study of the comments and other technical papers.

**4. Result's cross-check by different auditors**

Normally the research of the project is done by more than two auditors. This is followed by a step of mutual cross-check process of the audit results between different task performers.

**5. Report consolidation**

Consolidation of the audited report from multiple auditors.

**6. Reaudit of new editions**

After the provided review and fixes from the client, the found issues are being double-checked. The results are provided in the new version of the audit.

**7. Final audit report publication**

The final audit version is provided to the client and also published on the official website of the company.

# 1.4 FINDINGS CLASSIFICATION

## 1.4.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

- **CRITICAL**: A bug leading to assets theft, locked fund access, or any other loss of funds due to transfer to unauthorized parties.
- **MAJOR**: A bug that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
- **WARNING**: A bug that can break the intended contract logic or expose it to DDoS attacks.
- **INFO**: Minor issue or recommendation reported to / acknowledged by the client's team.

## 1.4.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

- **NEW**: Waiting for the project team's feedback.
- **FIXED**: Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
- **NO ISSUE**: Finding does not affect the overall security of the project and does not violate the logic of its work.
- **DISMISSED**: The issue or recommendation was dismissed by the client.

# 1.5 PROJECT OVERVIEW

Lido is a liquid staking solution for ETH backed by industry-leading staking providers. Lido lets users stake their ETH - without locking tokens or maintaining infrastructure - whilst participating in on-chain activities, e.g. lending.
Lido attempts to solve the problems associated with initial ETH staking - illiquidity, immovability and accessibility - making staked ETH liquid and allowing for participation with any amount of ETH to improve performance of the Ethereum network.

# 1.6 AUDIT SCOPE

- contracts/0.4.24/lib/Packed64x4.sol
- contracts/0.4.24/lib/SigningKeys.sol
- contracts/0.4.24/lib/StakeLimitUtils.sol
- contracts/0.4.24/nos/NodeOperatorsRegistry.sol
- contracts/0.4.24/oracle/LegacyOracle.sol
- contracts/0.4.24/utils/Pausable.sol
- contracts/0.4.24/utils/Versioned.sol
- contracts/0.4.24/Lido.sol
- contracts/0.4.24/StETH.sol
- contracts/0.4.24/StETHPermit.sol
- contracts/0.6.11/deposit_contract.sol
- contracts/0.6.12/interfaces/IStETH.sol
- contracts/0.6.12/WstETH.sol
- contracts/0.8.9/interfaces/IStakingModule.sol
- contracts/0.8.9/lib/Math.sol
- contracts/0.8.9/lib/PositiveTokenRebaseLimiter.sol
- contracts/0.8.9/lib/UnstructuredStorage.sol
- contracts/0.8.9/lib/UnstructuredRefStorage.sol
- contracts/0.8.9/oracle/AccountingOracle.sol
- contracts/0.8.9/oracle/BaseOracle.sol
- contracts/0.8.9/oracle/HashConsensus.sol
- contracts/0.8.9/oracle/ValidatorsExitBusOracle.sol
- contracts/0.8.9/proxy/OssifiableProxy.sol
- contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol
- contracts/0.8.9/utils/access/AccessControl.sol
- contracts/0.8.9/utils/access/AccessControlEnumerable.sol
- contracts/0.8.9/utils/PausableUntil.sol
- contracts/0.8.9/utils/Versioned.sol
- contracts/0.8.9/BeaconChainDepositor.sol
- contracts/0.8.9/Burner.sol
- contracts/0.8.9/DepositSecurityModule.sol
- contracts/0.8.9/EIP712StETH.sol
- contracts/0.8.9/LidoExecutionLayerRewardsVault.sol
- contracts/0.8.9/LidoLocator.sol
- contracts/0.8.9/OracleDaemonConfig.sol
- contracts/0.8.9/StakingRouter.sol
- contracts/0.8.9/WithdrawalQueueERC721.sol
- contracts/0.8.9/WithdrawalQueue.sol
- contracts/0.8.9/WithdrawalQueueBase.sol
- contracts/0.8.9/WithdrawalVault.sol
- contracts/common/interfaces/IEIP712StETH.sol
- contracts/common/interfaces/ILidoLocator.sol

- contracts/common/interfaces/IBurner.sol
- contracts/common/lib/ECDSA.sol
- contracts/common/lib/Math256.sol
- contracts/common/lib/MemUtils.sol
- contracts/common/lib/MinFirstAllocationStrategy.sol
- contracts/common/SignatureUtils.sol

The final commits to audit are:

- 2023-02-21: [ `e57517730c3e11a41e9cbc32ce018726722335b7` ] initial commit (Lido 2.0 beta2)
- 2023-03-14: [ `2bce10d4f0cb10cde11bead4719a5bcde76b93f9` ] updated Lido 2.0 beta3
- 2023-03-23: [ `ac06171909b752124069671e9676507c1f733a72` ] updated Lido 2.0 rc0-hotfix release
- 2023-04-04: [ `feafec437669a131a9e3c33ca680618d490c4fef` ] updated Lido 2.0 rc1 release
- 2023-04-13: [ `e45c4d6fb8120fd29426b8d969c19d8a798ca974` ] updated Lido 2.0 rc2 release

# 2 FINDINGS REPORT

OXORIO

## 2.1 CRITICAL

No issues found

## 2.2.1 Missing validation for `_withdrawalCredentials` in `StakingRouter`

| SEVERITY | MAJOR |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the functions `initialize` and `setWithdrawalCredentials` in the `StakingRouter` contract there is no validation for the variable `_withdrawalCredentials`.
So, if the `_withdrawalCredentials` variable is passed as an empty value, it will not be possible to call the `deposit` function.

### Recommendation

We recommend adding validation that `_withdrawalCredentials` is not empty and is of 0x01-type credentials (which support withdrawals).

### Update

LIDO's response

For the purpose of Lido V2 upgrade, a dedicated template contract that contains all necessary variables was developed to perform all necessary operations - such as creation, initialization, and configuration in an atomic way. In particular, for the `initialize` method in `StakingRouter`, the constant `_withdrawalCredentials` is preconfigured and does not change after the template is deployed.
The `setWithdrawalCredentials` method still can be used later on behalf of the `Lido DAO Agent` contract that has a granted role (which is a part of the whole protocol ACL setup). Therefore, the change requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks of changing `_withdrawalCredentials` if support the vote.

Oxorio's response

There is still a possibility of issues with data in the set up script as it happened with crvUSD.

## 2.2.2 Missing `_publicKeys` and `_signatures` validation in `NodeOperatorsRegistry`

| | |
|---|---|
| SEVERITY | **MAJOR** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `addSigningKeys` in the `NodeOperatorsRegistry` contract the node operator adds the new validator's public key with a signature. This function checks the length of the `_publicKeys` and `_signatures` arrays and that public key is not empty. But it does not check the validity of public key and signature. So if by any chance the operator sends an incorrect public key or signature with the right length, the data will be stored in the contract storage without any error.

In the `deposit` function of `StakingRouter` contract the deposit committee deposits buffered ETH to this module. It calls the `obtainDepositData` function of `NodeOperatorsRegistry` contract to receive public keys and signatures. After that it calls the `deposit` function of BeaconChain `DepositContract` contract with these public keys and signatures. If a public key or signature are incorrect but do have the right length, the `DepositContract` will still receive and store it without an error. The deposited funds are lost in this case, because they were deposited to the incorrect public key.

## Recommendation

We recommend implementing additional validation for `_publicKeys` and `_signatures` on the deposit committee side. If the deposit committee detects some incorrect keys or signatures it must remove it from the module storage.

## Update

LIDO's response

All keys used for the deposits pass an extensive set of the offchain checks to be used for deposits. Adding such a check into the on-chain code will considerably increase the gas costs still not allowing completely getting rid of the off-chain checks due to unavailability Consensus Layer state on Execution Layer.

## 2.2.3 There is no check for equal constructor variables in `DepositSecurityModule`

| | |
|---|---|
| SEVERITY | MAJOR |
| STATUS | ACKNOWLEDGED |

### Description

In the constructor in the `DepositSecurityModule` contract the variables `_lido`, `_depositContract` and `_stakingRouter` are validated for zero value, but these addresses are not validated that they are not equal to each other and there is no sanity checks that these addresses actually support interfaces. If the addresses are set incorrectly, they cannot be changed since they are `immutable` and the `DepositSecurityModule` contract must be redeployed.

### Recommendation

We recommend adding validation for interface support with `ERC165Checker` and check that addresses are not the same.

### Update

LIDO's response

The Lido governance token holders accept associated risks to verify the input for the `_lido`, `_depositContract` and `_stakingRouter` addresses correctness upon the Lido V2 upgrade (or the new `DepositSecurityModule` instance activation) if support the vote.

## 2.2.4 `checkAccountingOracleReport` may revert in case of skipped frames in `OracleReportSanityChecker`

| | |
|---|---|
| SEVERITY | MAJOR |
| STATUS | ACKNOWLEDGED |

## Description

In the functions `_checkOneOffCLBalanceDecrease` and `_checkSimulatedShareRate` in `OracleReportSanityChecker` the revert condition verifies the variables `oneOffCLBalanceDecreaseBP` and `simulatedShareDeviation` in terms of a single rebase report. Several skipped frames can produce the accumulated balance decrease and the share rate deviation that will not pass the checks, while it will not trigger the revert in case when frames were not skipped and balance changes were not accumulated.

## Recommendation

We recommend to clarify the workflow of the `oneOffCLBalanceDecreaseBPLimit` and `simulatedShareRateDeviationBPLimit` parameters for the case of several skipped frames and consider the time passed when checking the change against the limits. The revert of the report due to the mentioned limits will require manual adjustment of the limits in order to let the report pass the check.

## Update

LIDO's response

The risk is acknowledged and mitigated by the Lido DAO intervention vote that changes the limits if several skipped oracle frames lead to limits violation. Moreover, the `OracleReportSanityChecked` contracts enables the limits tuning by assigning the restrictive roles subset to a dedicated DAO committee if it had been gathered for this purpose.

## 2.2.5 Missing validation in `StakingRouter`

| | |
|---|---|
| SEVERITY | **MAJOR** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `initialize` in the `StakingRouter` contract there is no validation that `_lido` and `_admin` are the same address. If the same one is installed, it can lead to a complete block of the contract.

In the `constructor` in the `Burner` contract, there is no validation that `admin`, `treasury` and `_stETH` are the same address. If the same one is installed, it can lead to a complete

block of the contract because `treasury` and `_stETH` are `immutable` and cannot be updated.

## Recommendation

We recommend adding validation that the `_lido` and `_admin` addresses are different in `StakingRouter` and adding validation that the `admin`, `treasury` and `_stETH` addresses are different in `Burner`.

## Update

LIDO's response

The Lido governance token holders accept associated risks to verify the input for the both `_lido` and `_admin` addresses of StakingRouter contract and `_admin`, `_treasury`, `_stETH` addresses of Burner contract correctness upon the Lido V2 upgrade if support the vote.

# 2.2.6 Lack of validation of `_stakingModuleAddress` in `StakingRouter`

| | |
|---|---|
| SEVERITY | **MAJOR** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `addStakingModule` in the `StakingRouter` contract there is no validation that the `_stakingModuleAddress` is a contract and supports the `IStakingModule` interface. If the module `_stakingModuleAddress` is set as address, the staking module will be unusable. For example the `deposit` function will revert because of the missing interface, or if the `_stakingModuleAddress` is a contract, but is missing the `onWithdrawalCredentialsChanged` hook, the function `setWithdrawalCredentials` will always revert of `try` block, because of the missing interface. So, if the `_stakingModuleAddress` address is set incorrectly, the module will be lost forever without the possibility of deleting it or changing the `stakingModuleAddress` to a new one.

## Recommendation

We recommend validating the address for interface support with the `ERC165Checker` contract using the `supportsInterface` call.

## Update

LIDO's response

The Lido governance token holders accept associated risks to verify the input for the `_stakingModuleAddress` address correctness upon the Lido V2 upgrade if support the vote.

## 2.2.7 `REQUEST_BURN_SHARES_ROLE` can withdraw `stETH` for burning at any time in `Burner`

| | |
|---|---|
| SEVERITY | MAJOR |
| STATUS | ACKNOWLEDGED |

## Description

In the functions of `Burner` contract:

- requestBurnShares
- requestBurnSharesForCover

`REQUEST_BURN_SHARES_ROLE` can withdraw `stETH` for burning at any time from contract `WithdrawalQueue` and `NodeOperatorRegistry`. At the same time if this role gets assigned to a third party that is different from the verified contracts they can request burning shares in uncontrolled way, thus affecting economic mechanisms of the protocol in a critical way.

## Recommendation

We recommend limiting the set of callers of the functions `requestBurnShares` and `requestBurnSharesForCover` to verified contracts.

## Update

LIDO's response

The Lido governance token holders accept associated risks to verify the correctness of the assignment of the `REQUEST_BURN_SHARES_ROLE` role upon the Lido V2 upgrade if support the vote.

However, the finding is worth further consideration from the side of the Lido DAO contributors. The currently chosen design relies on the `stETH` token approvals for the

`Burner` contract functiioning, which might be changed in future releases taking into account the outlined risk of the ACL misconfiguration.

# 2.3 WARNING

## 2.3.1 Possibility of overflow in `Burner`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the function `commitSharesToBurn` in the `Burner` contract the variables `totalCoverSharesBurnt` and `totalNonCoverSharesBurnt` always get bigger. After some time the `Burner` contract will become unusable since these 2 variables will extend the limits of the `uint256` type and the `commitSharesToBurn` function will revert with overflow.

### Recommendation

We recommend adding setter functions which will reinitialize these 2 variables.

### Update

LIDO's response

These variables use `uint256` precision while deal with amounts comparable with ETH total supply. The risk of overflows is unrealizable in practice. Even though, it's possible to change the contract instance later via an on-chain vote if supported by token holders.

## 2.3.2 Allowance cannot be reset in `Lido`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the `Lido` contract there is no function for resetting allowance from the `WithdrawalQueue` contract to the `Burner` contract. If all of the allowance is used, the requests can no longer be finalized and all the system will be paused.

## Recommendation

We recommend adding function for resetting allowance to the `Burner` contract from the `WithdrawalQueue` in case when all of the allowance is used.

## Update

LIDO's response

Allowance is set as max `uint256` while `stETH` TVL is comparable with ETH total supply. The risk of underflows is unrealizable in practice. However, it was decided to implement a special case for infinite allowance to save gas and follow the latest ERC20 implementation in OpenZeppelin. Commit with changes: https://github.com/lidofinance/lido-dao/commit/e9509d77f010fec76899e25ccde785c8de47bd42. Therefore, the described finding is not applicable for the new code version.

# 2.3.3 Interface support in `LidoLocator`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the `constructor` in the `LidoLocator` contract there are no sanity checks for addresses. Passed parameters can be equal to each other, and, if the passed variables are missing the correct interface, all the system might be stopped. For example, if the `withdrawalQueue` is set incorrectly and is missing `IWithdrawalQueue`, the `deposit` function of the `lido` contract will always revert. Since all variables are `immutable`, passed variables cannot be reset.

## Recommendation

We recommend checking if the addresses are not equal to each other and that the passed contract implements the appropriate interface.

## Update

LIDO's response

The risks are mitigated with an extensive set of deployment checks, tests and alerting tools. In the worst case, the `LidoLocator` contract is upgradable through the on-chain Lido DAO vote.

## 2.3.4 All balance is used for rewards in `LidoExecutionLayerRewardsVault`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the function `withdrawRewards` in the `LidoExecutionLayerRewardsVault` contract the balance of the contract is used as an execution layer rewards. This means that any user can send value to the `LidoExecutionLayerRewardsVault` contract and this value will be used as a reward. The problem is that value sent to the contract can be sent by mistake, or it can be sent by the hacker of other protocol in order to mix funds and receive at least a part of the funds in a legal way. Since the `LidoExecutionLayerRewardsVault` contract is missing blacklist or return of the native ether logic, all received tokens will be used for rewards.

## Recommendation

We recommend to track the source of the native tokens and adding a function that can be called by DAO to return all the suspicious funds.

## Update

LIDO's response

This is the expected behavior. As this contract is used as `feeRecipient` by the Lido-participating validators which use the `mev-boost` middleware, it's expected that funds (native ether) may come from different type of transactions and even without transactions at all. The current design makes no assumptions about funds sources and balance top-up approaches which is required for the protocol operation.

## 2.3.5 All balance is used for withdrawals in `WithdrawalVault`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | NO ISSUE |

## Description

In the function `withdrawWithdrawals` of the `WithdrawalVault` contract the balance of the contract is used as withdrawals. This means that any user can send value to the `WithdrawalVault` contract and this value will be used as a withdrawal. The problem is that the value sent to the contract can be sent by mistake, or it can be sent by the hacker of other protocol in order to mix funds and receive at least a part of the funds in a legal way. Since the `WithdrawalVault` contract is missing blacklist or return of the native ether logic, all received tokens will be used for rewards.

## Recommendation

We recommend to track the source of the native tokens and adding a function that can be called by DAO to return all the suspicious funds.

## Update

LIDO's response

The `WithdrawalVault` contract has no payable `receive` or `fallback` functions to accept ether via transactions, as it serves as the `0x01`-type withdrawal credentials corresponding address to handle withdrawals of the Lido-participating validators. It's still possible to top-up the contract's balance (e.g., via the `selfdestrtuct` call) yet without invocation of the contract's on-chain code. Hence, tracking these funds can't be done on-chain.

# 2.3.6 Missing validations in `unsafeChangeDepositedValidators` in `Lido`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `unsafeChangeDepositedValidators` in the `Lido` contract there is not enough checks of the `_newDepositedValidators` variable and validations when the function can be invoked or not. Even considering that the function is operated via `UNSAFE_CHANGE_DEPOSITED_VALIDATORS_ROLE` which belongs to Lido DAO, the risk of using it incorrectly is very high, the following risks are:

- In the `_getTransientBalance` function in the Lido contract there is an `assert` that validates if `depositedValidators` >= `clValidators`, but due to the lack of validation of

input variable in `unsafeChangeDepositedValidators` function the variable in `DEPOSITED_VALIDATORS_POSITION` can be changed so that the `assert` will be met, while the `assert` should be used only for invariants, and the fail of the `assert` will lead to the failed report, which is critical point for Lido. Furthermore the incorrect setting for smaller value of the deposited validators with the `unsafeChangeDepositedValidators` function will revert `require` in `_processClStateUpdate` function and lead to incorrect accounting.

- If the `deposit` function is executed right before the invocation of the `unsafeChangeDepositedValidators` function, the `unsafeChangeDepositedValidators` will overwrite the value in `DEPOSITED_VALIDATORS_POSITION` which was set during the `deposit` function to the other one since the `unsafeChangeDepositedValidators` is setting the new value instead of adding value to the existing one. This leads to incorrect accounting of the deposited validators.
- In the `unsafeChangeDepositedValidators` function in the Lido contract the `canDeposit` `modifier` is missing, so the amount of the deposited validators can change during bunker mode, or when all staking modules are stopped or paused.
- After the `unsafeChangeDepositedValidators` function in the Lido contract the amount of deposited validators is changed, but these validators do not associate with any of the existing staking modules. Thus it is possible to deposit more than `_maxDepositsCount` of any staking module, there is no execution of the `obtainDepositData` hook of the staking module address, there is no update of the `lastDepositBlock` variable in DSM and of `stakingModule.lastDepositAt`, `stakingModule.lastDepositBlock` variables in the `StakingRouter` contract.

## Recommendation

We recommend refactoring the `unsafeChangeDepositedValidators` function with more validations for bunker mode, introducing interactions with the `StakingRouter` contract, adding value to the existing amount of valitators instead of setting it.

## Update

### LIDO's response

The `unsafeChangeDepositedValidators` method calls require an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input for the `_newDepositedValidators` variable correctness if support the vote.

The method was introduced to support the onboarding of the already deposited validators with 0x00 credentials to the Lido protocol by rotating their withdrawal credentials to the type-0x01 ones used by Lido. It is prefixed with `unsafe_` and restricted by a role with `UNSAFELY_` prefix to raise additional attention before ever being used. The Lido V2 deployment template doesn't assign this role.

## 2.3.7 Members of the deposit committee can collude with node operators in `DepositSecurityModule`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

To mitigate the deposit front-running vulnerability, Deposit Security Committee was established in **LIP-5**. The ability of collusion between the deposit committee and node operators is considered in this LIP:

> Members of the committee can collude with node operators and steal money by signing bad data that contains malicious pre-deposits. To mitigate this we propose to allow a single committee member to stop deposits and also enforce space deposits in time (e.g. no more than 150 deposits with 150 blocks in between them), to provide the single honest participant an ability to stop further deposits even if the supermajority colludes.
>
> This design ensures the protocol's robustness even with just a single honest committee member. The impact of majority-malicious committee is limited to 4800 ETH at most (150 keys allowed within a time window, 32 ETH deposited to every key).

The possible amount of funds under risk (4800 ETH) is a very big amount which equals 150 slashings. The closed set of Deposit Security Committee members increase the risk (in case of no honest participant).

## Recommendation

To minimize possible risk we recommend implementing additional functionality for Deposit Security Committee:

1. Staking/slashing mechanism for the committee members with amount comparable to the amount of funds at risk.
2. Making committee members set open for external participants willing to make a stake. It increases the probability that there will be at least a single honest participant.

## Update

LIDO's response

As it's described in docs.lido.fi, the members of the deposit committee consist of six node operators and a Lido dev contributors subteam while we want to extend this set.

The mentioned LIP-5 and its associated risks were accepted by the Lido governance token holders (for example, the loss of 4800ETH corresponds to less than 0.1% of the current protocol's TVL).

## 2.3.8 Missing sanity check that `_stETH` is a `stETH` contract in `Burner`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the `constructor` in the `Burner` contract there is no sanity check that `_stETH` is a `stETH` contract. Given that the `_lido` address cannot be updated, this can lead to a contract lock if a different address than `stETH` is set.

### Recommendation

We recommend to validate `_stETH` address for interface support with the `ERC165Checker` contract using the `supportsInterface` call.

### Update

LIDO's response

The Lido governance token holders accept associated risks to verify the input for the `_stETH` address correctness upon the Lido V2 upgrade if support the vote.

## 2.3.9 Missing validation for duplication of staking module names in `StakingRouter`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the function addStakingModule in the `StakingRouter` contract there is no validation for duplication of the `_name` parameter of staking module names.

## Recommendation

We recommend adding validation for duplication of staking module names.

## Update

LIDO's response

The addition of a module to StakingRouter requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input for the staking module name correctness if support the vote.

## 2.3.10 Missing logic for updating staking module name in `StakingRouter`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function updateStakingModule there is no logic for updating the `name` parameter of staking module. In case of an error in the title, it will not be possible to update it.

## Recommendation

We recommend adding a logic for updating the `name` parameter of staking module in `StakingRouter`.

## Update

LIDO's response

The adding module to the `StakingRouter` requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input for the staking module name correctness if support the vote.

Moreover, it's still possible to update the staking module name later on behalf of the `StakingRouter` upgrade procedure since the contract is upgradable via the Lido DAO Aragon vote.

## 2.3.11 Missing validation for `treasuryFee` and `stakingModuleFee` in `StakingRouter`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the functions addStakingModule and updateStakingModule there is no `treasuryFee` and `stakingModuleFee`. It is checked that the sum of `stakingModuleFee` and `treasuryFee` does not exceed `TOTAL_BASIS_POINTS`, but `stakingModuleFee` and `treasuryFee` are not validated or limited in any way. This means you can set `stakingModuleFee = 0` and `treasuryFee=100%`, for example.

### Recommendation

We recommend adding validation for the `treasuryFee` and `stakingModuleFee` parameters.

### Update

LIDO's response

This is the expected behavior. The addition or update of a staking module requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input for the `treasuryFee` and `stakingModuleFee` correctness if support the vote.

## 2.3.12 Missing error handling logic when calling `stakingModule` in `StakingRouter`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the following lines there is no error handling or changing status of staking module to `DepositPaused` or `Stopped`:

- StakingRouter#L273
- StakingRouter#L288
- StakingRouter#L303
- StakingRouter#L435
- StakingRouter#L515
- StakingRouter#L546
- StakingRouter#L569
- StakingRouter#L1120

## Recommendation

We recommend to add error handing logic with `try/catch` block and set the status of the contract to `DepositsPaused` or `Stopped` in cases of errors in function calls.

## Update

LIDO's response

This is the intended design decision. The `DepositPaused` status will be set to the staking module only in two cases:

- an error occurred while trying to change `withdrawalCredentials`
- `DepositSecurityModule` stoped the deposit for specific module

The updating module to the `Stopped` status requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input status correctness if support the vote.

## 2.3.13 Try catch can revert in `StakingRouter`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the following functions of the `StakingRouter` contract:

- `setWithdrawalCredentials`

- `reportRewardsMinted`
- `onValidatorsCountsByNodeOperatorReportingFinished`

`try` call can revert all transactions without executing `catch`. `Try/catch` reverts transaction:

- If an EOA address is called (e.g. `address(0)`).
- If the target contract does not have the called method.
- If the target contract returns a wrong number of arguments.

For example, in the `setWithdrawalCredentials` function `stakingModuleAddress` can revert in the `try` block when `onWithdrawalCredentialsChanged` is called, if `stakingModuleAddress` does not support the `IStakingModule` interface properly. In this case withdrawal credentials of staking module cannot be changed.

## Recommendation

We recommend adding validation that the `stakingModuleAddress` has a correct interface before calling it.

## Update

LIDO's response

The adding module to the StakingRouter requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify the input for the `stakingModuleAddress` correctness if support the vote.

The compatibility of a new staking module is have to be checked following the established Lido DAO development process: internal peer-reviews, extensive integration and regression tests, external audits, and the design guidelines and docs.

## 2.3.14 Underflow validation in `Packed64x4`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the functions `get` and `set` of `Packed64x4` library there is no underflow protection of the `n` variable. For example, the following expression `(_self.v >> (64 * n))` can be translated into `_self.v / 2^(64*n)`. If the `n` variable is passed greater than 3, it will

underflow and return the result of the `(n % 4)` calculation, which is incorrect, since `_self.v / 2^(64*n)` and `_self.v / 2^(64*(n % 4))` are not equal expressions.

## Recommendation

We recommend to enforce the value of `n` to be less than 4 or make sure that `n` cannot be passed with user input, or, if the constants are used, they are not updated with contract upgrades.

## Update

LIDO's response

The risk is accepted because the library's client contract never violates the desctibed invariant by design.

## 2.3.15 Total `targetShare` can be higher than 100% in `StakingRouter`

| | |
|---|---|
| SEVERITY | **WARNING** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the functions `addStakingModule` and `updateStakingModule` of the `StakingRouter` contract the `_targetShare` variable is not validated properly. Target share is expressed as the percentage of active validators in staking module to those in total across all modules and is regulated by DAO. It is possible to set 2 staking modules with total target share more than 100%.

## Recommendation

We recommend to store total target share of all modules and validate the `_targetShare` variable explicitly.

## Update

LIDO's response

The `addStakingModule` and `updateStakingModule` methods calls on StakingRouter requires an on-chain Aragon vote to enact and the Lido governance token holders accept associated risks to verify `_targetShare` correctness for each module if support the vote.

## 2.3.16 Missing remove module logic in `StakingRouter`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the `StakingRouter` contract there is no option to delete unused, or incorrectly set staking modules. If the staking module was set with incorrect `name` or `stakingModuleAddress`, or if the staking module is stopped and is not used in the future, it cannot be deleted and will always be used in functions, that iterate over all staking modules id.

### Recommendation

We recommend to add a function to delete the deprecated modules.

### Update

LIDO's response

The implementation of the module removal logic is planned in future.

## 2.3.17 Number of staking modules cannot be changed in `StakingRouter`

| SEVERITY | WARNING |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

The `MAX_STAKING_MODULES_COUNT` variable in the `StakingRouter` contract is constant and the amount of staking modules can be changed with setter functions, but at the same time the `MAX_STAKING_MODULES_COUNT` variable is stored in the bytecode of implementation and can be changed with implementation upgrade.

## Recommendation

We recommend to make `MAX_STAKING_MODULES_COUNT` a variable and add setter function for modifying it in order to simplify `MAX_STAKING_MODULES_COUNT` upgrade. If the amount of staking modules is designed to be not changeable, we recommend to use `immutable` instead of `constant`.

## Update

### LIDO's response

Any change to a contract or variable will go through a DAO vote, so there is no need to add a redundant method to update this variable.

## 2.4 INFO

### 2.4.1 `MANAGE_NODE_OPERATOR_ROLE` is overpowered in `NodeOperatorsRegistry`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

#### Description

In the function `setNodeOperatorRewardAddress` in the `NodeOperatorsRegistry` contract only the `MANAGE_NODE_OPERATOR_ROLE` can change the operator reward address. This way the operator cannot change his reward address himself.

#### Recommendation

We recommend to add an option to change reward address for the operator.

### 2.4.2 Guardians are not stored in sorted array in `DepositSecurityModule`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **NO ISSUE** |

#### Description

In the function `_addGuardian` in the `DepositSecurityModule` contract guardians are not stored in the sorted array and the `getGuardians` function will return unsorted array of the guardians. While in the `_verifySignatures` function all the guardian signatures must be passed as sorted array of guardians.

#### Recommendation

We recommend sorting guardians in the `_addGuardian` function.

## Update

LIDO's response

The mentioned method `_verifySignatures` uses the internal state of the guardians subset only to check whether the address is included in the subset or not while the sorting itself happens off-chain. Therefore, there is no need to store the guardians sorted.

## 2.4.3 `require` should be removed in `Burner`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **FIXED** |

## Description

In the functions:

- `requestBurnMyStETHForCover`
- `requestBurnMyStETH`
- `recoverExcessStETH`

the `require` with the `transfer` and `transferFrom` call is redundant since these two functions in the `stETH` contract will always return `true`, except in situations when they revert. But if the `transfer` or `transferFrom` functions revert the execution will not reach the `require` statement.

## Recommendation

We recommend removing the `require` check from these functions in order to keep the codebase clean and save gas.

## Update

LIDO's response

Fixed in commit: https://github.com/lidofinance/lido-dao/pull/735/commits/e3ee224f547f6dd9224d57f684bf7c80c35e49f4

## 2.4.4 `key` can be updated with the same value in `OracleDaemonConfig`

| SEVERITY | INFO |
| --- | --- |
| STATUS | FIXED |

### Description

In the `update` function in the `OracleDaemonConfig` contract there is no validation that the new `_key` variable equals the existing `key`.

### Recommendation

We recommend validating that the new `_key` is not equal to the existing `key`.

### Update

LIDO's response

Fixed in commit: https://github.com/lidofinance/lido-dao/pull/737/commits/4c617c636c6a3d62766d04972a9ce98cd525c3a3

## 2.4.5 Int type initialization to zero is redundant

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

There are a few places in the code where variables are initialized to zero:

- NodeOperatorsRegistry.sol#L858
- NodeOperatorsRegistry.sol#L938
- NodeOperatorsRegistry.sol#L940
- MinFirstAllocationStrategy.sol#L35
- MinFirstAllocationStrategy.sol#L69
- StakingRouter.sol#L295
- StakingRouter.sol#L371
- StakingRouter.sol#L780

- StakingRouter.sol#L831
- StakingRouter.sol#L998

These initializations are redundant because zero is the default value of int/uint type variable in Solidity.

## Recommendation

We recommend removing redundant initialization to zero.

## Update

LIDO's response

While declaring variables in this case is redundant, the readability and clarity of code is better.

## 2.4.6 `STAKING_MODULE_INDICES_MAPPING` logic is redundant in `StakingRouter`

| | |
|---|---|
| SEVERITY | INFO |
| STATUS | ACKNOWLEDGED |

## Description

In the function `_setStakingModuleIndexById` in the `StakingRouter` contract the `STAKING_MODULE_INDICES_MAPPING` is redundant because the key of this mapping always equals to value (`id == index + 1`).

## Recommendation

We recommend removing redundant logic to safe gas and improve code readability.

## Update

LIDO's response

Since it's planned to add a module removal feature in the next upgrades `STAKING_MODULE_INDICES_MAPPING` is needed to allow proper modules enumeration and access.

## 2.4.7 Unclear use of the `moduleAddr` variable in `StakingRouter`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **FIXED** |

### Description

In the `unsafeSetExitedValidatorsCount` function of the `StakingRouter` contract the `stakingModule.stakingModuleAddress` variable is used while the `moduleAddr` variable is already declared and used the same way in the `unsafeSetExitedValidatorsCount` function.

### Recommendation

We recommend to use the `moduleAddr` variable instead of `stakingModule.stakingModuleAddress` in this place of code.

### Update

LIDO's response

Fixed in commit: https://github.com/lidofinance/lido-dao/pull/741/commits/2fb7299145f650d3dba02716a1c825ed925766ec

## 2.4.8 Typos in contracts

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **FIXED** |

### Description

In the function `getTotalFeeE4Precision` of the contract `StakingRouter` there is a typo `total fee total fee` instead of `total fee`.

In the contract `StakingRouter` there is a typo `panlty` instead of `penalty`.

In the contract `NodeOperatorsRegistry` there is a typo `timastamp` instead of `timestamp`.

In the contract `NodeOperatorsRegistry` there is a typo `reawards` instead of `rewards`.

In the function `_removeUnusedSigningKeys` in the contract `NodeOperatorsRegistry` there is a typo `comapring` instead of `comparing`.

In the contract `WithdrawalQueueBase` there is a typo `int the queue` instead of `in the queue`.

In the contract `WithdrawalQueueBase` there is a typo `the resukt later` instead of `the result later` and the phrase is repeated two times.

In the contract `WithdrawalQueueBase` there is a typo `intemediate` instead of `intermediate`.

In the contract `WithdrawalQueueBase` there are multiple typos `invokations` instead of `invocations`.

In the contract `WithdrawalQueueBase` there is a typo `better to me` instead of `better to be`.

In the contract `NodeOperatorsRegistry` there is a typo `for this operator` instead of `for all operators`.

In the contract `NodeOperatorsRegistry` there is a typo `TYPE_POSITION` instead of `STUCK_PENALTY_DELAY_POSITION`.

In the contract `NodeOperatorsRegistry` there is a typo `to set reward address for` instead of `to set staking limit for`.

## Recommendation

We recommend fixing the typos to keep the codebase clean.

## Update

### LIDO's response

Fixed in commit: https://github.com/lidofinance/lido-dao/pull/736/commits/7a92c9c38faa70b26237ede5187acf4754e35506

## 2.4.9 Out-of-gas validation in `StakingRouter`

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the `reportRewardsMinted` function of `StakingRouter` contract there is a `try/catch` block for validation of the `out-of-gas` error, but the error `UnrecoverableModuleError` is not used in the `_estimate_gas` function for recalculating of gas if the Ethereum nodes proposed incorrect amount of gas.

### Recommendation

We recommend recalculating neccessary amount of gas in `_estimate_gas` function if the `UnrecoverableModuleError` occurs.

### Update

LIDO's response

This revert was introduced for the Ethereum node to be able to predict gas properly (see the comment to the code). Without this revert, gas estimation calculation was wrong and the node believed that less gas is enough to send a transaction, since `onRewardsMinted` may fall (including out-of-gas).

## 2.4.10 No logic for manual reward distribution in `StakingRouter`

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

### Description

In the `getStakingRewardsDistribution` function of the `StakingRouter` contract if the module is stopped, the reward for that module will go to the Lido treasury, but there is no logic for distributing the rewards to the staking module with the `onRewardsMinted` call. If the module is unstopped and Lido DAO decides to transfer this rewards to the staking

module, the `onRewardsMinted` call of the staking module will not be executed, since it is very likely that this hook will be authorized from the `StakingRouter` contract.

## Recommendation

We recommend adding a function for transferring rewards in this case with the `onRewardsMinted` call in the `StakingRouter` contract.

## Update

LIDO's response

Rewards distribution happens only on oracle report and cannot be called manually currently.
However, the fee distribution model will be re-visited in future `StakingRouter` upgrades when plugging-in new modules that use pull-based fee distribution approaches.

## 2.4.11 Missing on-chain validation in the function `requestWithdrawals` in `WithdrawalQueue` during the bunker mode

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `requestWithdrawals` of the `WithdrawalQueue` contract stETH holders can create withdrawal requests during the bunker mode.
Existing withdrawal requests are finalized by the oracle report in the function `_handleOracleReport` of the `Lido` contract. By the protocol architecture withdrawal requests created after bunker mode activation cannot be finalized prior to bunker mode deactivation but no on-chain workflow ensures it.

## Recommendation

We recommend implementing on-chain verification that the new withdrawal requests cannot be finalized prior the bunker mode deactivation. Introducing a flag that marks the requests created during the bunker mode may allow to ensure the condition on-chain.

## Update

LIDO's response

It's the expected behavior by the protocol withdrawals design since the bunker mode activation can't be validated on the Execution Layer side in a trustless way.

## 2.4.12 Frontrun `deposit_root` for pausing deposits in `DepositSecurityModule`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `depositBufferedEther` of the `DepositSecurityModule` contract there is a check that the `onchainDepositRoot` on the `deposit_contract` has not changed and equals the `depositRoot` provided from the `calldata`. According to the EIP-6110 there is a probability for DoS vector with the minimal deposit amount as 1 ETH, even though it is not related with the `depositRoot`, the risks related to the DoS of minimal deposit amount should be considered. Thus, there is a risk of frontrunning every single transaction of the `depositBufferedEther` and creating deposit directly through `deposit_contract` with minimal stake amount in order to change the `deposit_root` and revert the execution of the `depositBufferedEther` function which will lead to the pausing of deposits to all staking modules.

In total there is a `24 * 60 * 60 / 12 = 7200` blocks on Ethereum network per day, according to the beacon chain statistics the max day deposits is around `200.000 ETH` per day, which can be up to `200.000` validators taking in consideration that minimum deposit amount is `1 ETH`. Anyone can frontrun deposits signed by guardians in order to revert the function execution and prevent Lido from staking. This attack can be executed with at least `7200 ETH` in order to block `Lido` deposits for all day long, which is a very big amount of ETH, but at the same time this ETH can be withdrawn from the beacon chain after adding additional ETH up to `32 ETH`. There is a probability, that this attack can take place, since the attacker will not lose any funds, but at the same time there is no immediate profit. This attack can be executed by other protocols that work with consensus layer staking during the time when there are a lot of people willing to make a deposit to the beacon chain and by pausing the deposits in Lido they will push users to try other liquid staking protocol. The attack may be more harmful during the time of large MEV, preventing the LIDO protocol from making deposits that otherwise would result in higher amounts of MEV captured. The

cost of the attack is not significant for the major competitors of LIDO and is profitable when matched against the values of MEV seen on the Beacon chain so far.

## Recommendation

We recommend refactoring the deposit logic taking into consideration the possibility of the DoS attack.

## Update

LIDO's response

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

# 2.4.13 Mass slashing of non-Lido validators increases the potential damage from malicious behavior of Lido node operators

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

Node operators actually control all users' staking ETH, because they control validators. A node operator can slash its validators and the protocol will bear the losses. At the same time, the node operator risks only income from working with Lido. A simplified formula for such losses looks like this:

$$3 * B * S/T$$

where:
$B$ - balance of the validator.
$S$ - sum of the balances of slashed validators in the last 36 days.
$T$ - the total active balance of all validators.

In other words, under normal conditions, such a node operator will not cause significant losses for Lido, but if mass slashing occurs in the network, then the node operator can thus reduce the balance of validators down to 0.

It is worth noting that the node operator can also benefit from slashing Lido validators by providing evidence of the slashing behalf of its non-Lido validators. So, the node operator may collude with a Lido competitor with a large pool of validators:

- Firstly, because it is profitable for competitors to arrange slashes on Lido validators in order to significantly reduce TVL.
- Secondly, a Lido competitor with a large pool of validators can use them during block proposals to register an evidence of slashings of the Lido validators and be rewarded for it.

## Recommendation

We recommend reducing the impact of each node operator on the entire protocol, for example, by increasing their number relative to the total number of validators.

## Update

LIDO's response

The risk is accepted and mitigated by implementing a more diversified validators set. To lower the impact of node operators drastically, Lido would adopt withdrawal credentials triggerable exits once and if they got implemented on Ethereum. Worth noting that validator exits ordering is implemented with the same effort to make node operators stake allocation more uniform (see the recently ratified exits policy: https://snapshot.org/#/lido-snapshot.eth/proposal/0xa4eb1220a15d46a1825d5a0f44de1b34644d4aa6bb95f910b86b29bb7654e330.

## 2.4.14 Large deposits and withdrawals during the limiter-capped rebases in `OracleReportSanityChecker`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the function `smoothenTokenRebase` of the `OracleReportSanityChecker` contract a large single-frame or multi-frame rebase (e.g. at the moment of high volatility) can be capped several times producing large APR across several frames (~27% according to current `maxPositiveTokenRebase` value of `0.075%`, can be changed by the governance). Rebase transaction can be front-runned by a large deposit in a protocol (daily staking limit is `150 000 ETH`, `~2.5%` of current TVL). Under current conditions rebase is limited at `4500 ETH`. At 11 march, 2023 MEV paid to proposers amounted to `6113 ETH`. It makes it profitable to

surround rebase transaction with deposit and withdraw transactions, making a profit of `112 ETH` (`150 000 ETH * 0.075%`) per frame of maximal rebase. Several frames of maximal rebase produce profit of `112 ETH` multiplied by the number of frames. Similar situation is observed with skipped frames, though under current conditions it takes 8 days with no oracle report whatsoever to reach rebase limit. Skipped frames during high volatility time produce even higher output.
While protocol benefits from deposits overall, sandwich type deposits over single frames do not produce any positive effect other than diluting profits and exhausting available validators key set. Deposits held over multi-frame capped rebases during periods of high volatility are more profitable for the protocol, as additional liquidity allows to capture more MEV. Positive feedback loop can be observed, as higher profits during those frames may produce an influx of liquidity to the protocol, noticeably increasing its TVL, ending up with a sharp withdrawals shortly after, likely with the effect of a massive flooding of the withdrawal queue with significant amount of large withdrawals.
Current Beacon chain activation rate is around `2000` validators per day (`~64000 ETH` per day). This is meaningful upper limit for deposits over single frame, any deposits over that amount that ended up with a withdrawal in the next frame will produce nothing but a profit dilution.
The current solution uses `PositiveTokenRebaseLimiter` to limit the rebase value over the single frame. The protocol limits the staking amount per day, while it is capped at a value that is significantly higher than beacon chain processing capacity. The deposits can be withdrawn in the next frame without producing any profit on the consensus layer, while such deposits will participate in reward distribution together with all other stakers. This allows to capture profit without producing any benefit for the protocol.

## Recommendation

Several suggestion may be offered to optimize the protocol workflow during the capped rebases.
1) Dynamic staking limit will give more flexibility over large deposits during the time of high income. Strategies based on floating staking limit may help to encourage deposits that help to generate more income, while discouraging those that are made by the high APR hunters.
2) Distribute the income solely over the deposits that arrived to beacon chain, thus participated in producing protocol income. This option may be activated just at the time of high rebase, encouraging the stakers for prolonged deposits.
3) Make a withdrawal finalization longer for the stakes made during the period of the capped rebase activation.
4) Consider the current activation rate and activation queue length in adjusting the staking limit, which will allow to limit the dilution of profit only to stakes that arrived to beacon chain and thus participated in the production of the income.

## Update

LIDO's response

This risk is accepted since the protocol doesn't have total exposure to market conditions, but a set of measures to mitigate already known and possible attack vectors. Therefore, it's could be theoretically possible to build sophisticated market strategies that lead to short-term profits when outstanding network events happen.

However, the proposed vector has different possible mitigations to prevent attack become sustainable: lowering the token rebase limit, extending withdrawal finalization time, and changing the daily staking limit by the DAO governance vote once suspicious behavior observed.

Worth noting that withdrawals are backed by buffered ether whenever it's possible, thus if an attacker decides to exit, they either effectively get their funds back from buffer or wait till enough exited validators appear (if request was placed after the real deposit to Beacon Chain). Therefore, in the first case, massive validators activation/exit process isn't a case, while for the second case the attacker short-term rewards will be dilluted by the waiting time in a queue.

# 2.4.15 "Memory Array Creation Overflow" compiler bug

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the functions:

- `getNodeOperatorIds`
- `_getSigningKeysAllocationData`
- `getRewardsDistribution`
- `getSigningKeys`
- `_transferModuleRewards`

in the `NodeOperatorsRegistry` and `Lido` contracts there is a possibility of "Memory Array Creation Overflow" Solidity compiler bug. The creation of very large memory arrays can result in overlapping memory regions and thus memory corruption. In cases when memory size of an array is in bytes, i.e. the array length times 32, is larger than 2^256-1, the memory allocation will overflow, potentially resulting in overlapping memory areas. The length of the array is still stored correctly, so copying or iterating over such an array will result in out-of-gas. The functions above are protected from the array overflow by the out-of-gas error, and

the overflow may take place only in the function `_getSigningKeysAllocationData`, in case `activeNodeOperatorsCount` is bigger than `nodeOperatorsCount`, which in theory may happen during incorrect implementation upgrade.

## Recommendation

We recommend to pay more attention to contract upgrades and take into consideration that memory arrays must be limited in length up to `type(uint64).max`.

## Update

LIDO's response

The risk of incorrect implementation upgrade is accepted by the Lido DAO since the upgrade requires an on-chain Aragon vote to enact supported by the governance token holders.

The compatibility of a new implementation is have to be checked following the established Lido DAO development process: internal peer-reviews, extensive integration and regression tests, external audits, and the design guidelines and docs.

## 2.4.16 ECDSA signature malleability in the OpenZeppelin library in `EIP712StETH`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the contract `EIP712StETH` there is an import from the ECDSA library, which has signature malleability with the `recover` and `tryRecover` functions. In this affected version, user may take a signature that has already been submitted, submit it again in a different form, and replay the signature.
In other contracts custom ECDSA library is used for the `recover` function without malleability. During the contract development/upgrade of implementation the vulnerable contract version may be used by mistake creating a big security risk.

## Recommendation

We recommend to remove import from the OpenZeppelin ECDSA library from the contract and migrate `toTypedDataHash` to the custom ECDSA library.

## Update

LIDO's response

The proposed recommendation will be considered for implementation in future versions.

## 2.4.17 Explicit cast to address in `StakingRouter`

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the function `getStakingRewardsDistribution` there is an explicit cast of the `stakingModuleAddress` variable to address, when it is already of an address type.

## Recommendation

We recommend to remove the cast to address in order to keep the codebase clean.

## Update

LIDO's response

The proposed recommendation will be considered for implementation in future versions.

## 2.4.18 `UINT64_MAX` explicitly declared in `NodeOperatorsRegistry`

| SEVERITY | INFO |
| --- | --- |
| STATUS | ACKNOWLEDGED |

## Description

In the contract `NodeOperatorsRegistry` the constant `UINT64_MAX` is explicitly declared, since it already exists in the `Packed64x4` library.

## Recommendation

We recommend to remove the `UINT64_MAX` variable from the `NodeOperatorsRegistry` contract and, instead, use the same variable from `Packed64x4` library in order to save bytecode and keep the codebase clean.

## Update

LIDO's response

The proposed recommendation will be considered for implementation in future versions.

## 2.4.19 Link does not exist in `StETH`

| | |
|---|---|
| SEVERITY | **INFO** |
| STATUS | **ACKNOWLEDGED** |

## Description

In the `StETH` contract the link leads to the page, which was deleted and was not saved on the Wayback Machine website.

## Recommendation

We recommend to remove the link in order to keep the codebase clean.

## Update

LIDO's response

The comment will be fixed in future releases.

# 3 AUDITED INCIDENTS REPORT

0XORIO

## 3.1.1 DSM can initiate a deposit between `ref_slot` and the oracle report execution block

### Description

The `deposit` function of the `Lido` contract can be called by DSM just before the oracle report transaction thus reducing the buffered ETH value. It can lead to the situation that buffered ETH is not enough for withdrawal requests finalization in the `_handleOracleReport` function of the `Lido` contract. Because oracle prepares a set of withdrawal requests for finalization (`withdrawal_batches`) using buffered ETH value in `ref_slot` block.

### Solution

In the function `getDepositableEther` in the `Lido` contract there is a calculation that the buffered ETH value is not less than the sum of unfinalized withdrawal requests after deposit execution in the `deposit` function of the `Lido` contract. Thus the existing check is enough and this potential incident is handled carefully by the protocol.

## 3.1.2 StETH/ETH stability during bunker mode

### Description

In the `get_finalization_batches` function in the `withdrawal.py` file withdrawal requests are not finalized during bunker mode, which can become a reason of moving the price in stETH/ETH pool from 1/1 to other values, leading to money loss of the stETH holders, liquidations on the lending platforms, increasing the posibility of the "bunkrun" from Lido staking.

### Solution

The `_get_associated_slashings_border_epoch` function in the `safe_border.py` file represents the latest epoch before associated slashings started. During the bunker mode, withdrawal requests are accepted and user can exit Lido staking in order to stop taking risks of further slashings. Since slashing is associated with the withdrawal request, after the slashing is covered the withdrawal request will pass the associated slashing border and it

can be finalized, user will receive his funds no matter how much slashings will occur later during the covering process. Taking into the consideration that before the Lido update there was no withdrawals and users continue to hold stETH tokens in the liquidity provider pools for collecting fees, the price stability in the pool will more likely remain as it is, even though the withdrawal process will take longer time.
We would like to draw extra attention to the case when the informed actor that monitors the LIDO performance on the CL layer and knows in advance that the bunker mode will get activated in the next frame, will prefer to swap large amount of stETH in the Curve pool to avoid socialized losses and lack of the withdrawal possibility until the bunker mode gets deactivated. This is profitable to the margin when the swap slippage becomes larger than the potential loss of socialized slashings. This may result in unbalanced Curve pool, depeg of stETH from ETH with all the possible consequences for the third parties.

### 3.1.3 `getStakingRewardsDistribution` returns empty values for stopped modules in `StakingRouter`

## Description

In the function `handleOracleReport` in the `Lido` contract during the `_processRewards` function call the fee is distributed only to the profitable reports. If the report was profitable the Lido contract will call `StakingRouter` with the `getStakingRewardsDistribution` call for calculations. If the module is stopped, but it has active validators, the module will be present in the return arrays of this function, but the `stakingModuleFee` of this module will be zero.

## Solution

The module with empty fees will be present in all `for` loops in the `_transferModuleRewards` function in the `Lido` contract and `reportRewardsMinted` in the `StakingRouter` contract, but the calculations will not be proceeded with the module and there will be no `onRewardsMinted` call, which is the correct logic that does not create any problems when returning empty values from arrays, but it still costs gas for iterating over the loop for the stopped modules with active validators, which should be refactored.

## 3.1.4 Length of `_depositCalldata` does not equal amount of deposits in `StakingRouter`

### Description

In the `deposit` function of the `StakingRouter` contract the `_depositCalldata` variables' length is not validated with the `_depositsCount`. For example, it is possible to pass 2 deposits with 1 key, which will lead to revert, or with 3 keys which will lead to 1 unused key, because the `_depositsCount` variable is passed to `_makeBeaconChainDeposits32ETH` function. A call to the `deposit_contract` contract with incorrect `pubkey` to `signature` variables can lead to corrupt deposits where the `signature` cannot verify its `pubkey`.

### Solution

In the `obtainDepositData` function call to the `stakingModuleAddress` contract returned `publicKeysBatch` and `signaturesBatch` must return the exact number of requested keys according to the pull request. Even if the staking module does not return correct amount of public keys and signatures, there is a check `_makeBeaconChainDeposits32ETH` function preventing incorrect handling of these variables. Futhermore, in the `deposit` function of the `deposit_contract` there is a sanity check for the provided data length.

## 3.1.5 False-positive stop accepting deposits in `DepositSecurityModule`

### Description

In the function `pauseDeposits` of the `DepositSecurityModule` contract it is enough to have one vote of a committee member to pause accepting deposits for a certain module for days, until the module is unpaused by the DAO.

### Solution

The `DepositEvent` in the `deposit_contract` is monitored by the committee members. They verify the signatures of the deposit, and also check that the `withdrawal_credentials` of the deposit matches the `withdrawal_credentials` obtained from the function `getWithdrawalCredentials` of the `StakingRouter` contract. If `withdrawal_credentials` does not match, the module of this public key is paused. In other words, only the node operator of this module can deceive the monitoring system and pause the module.

False-positive stop on behalf of the committee members themselves is not a big problem, since such members could be excluded in case of malicious intent.

### 3.1.6 Oracle accounting report flow over skipped frames

## Description

Accounting oracle report is delivered by the group of off-chain oracles, which deliver report based on predefined time intervals and employ consensus mechanism to agree on the matching report data. After consensus is reached, the report is checked for the data correctness and matched against the limits set up by the protocol governance. Then the on-chain state of the protocol is updated and matched against the share rate limit.
The current workflow allows the case when report will not be accepted, which will result in the skipped frame. This may happen due to lack of the consensus, incorrectness of the report, or by incorrect data supplied by beacon chain nodes.

## Solution

By the logic of the protocol, all the changes that were not accounted in the current frame will be accounted in the next one, all the changes on the consensus layer whether positive or negative will be delievered with the next successful report.

Detailed look into oracle report flow when one or several frames were skipped, produced the following picture.

The off-chain oracle will:
1) In the function `_get_consensus_lido_state` of the `Accounting` class calculate the validators count and balances for the current ref slot.
2) In the function `_get_newly_exited_validators_by_modules` of the `Accounting` class count newly exited validators for all skipped frames.
3) In the functions `get_lido_newly_stuck_validators` and `get_lido_newly_exited_validators` of the `LidoValidatorStateService` class count all the newly exited and stuck validators.
4) In the function `simulate_full_rebase` of the `Accounting` class simulate the rebase accounting for the skipped frames.
5) In the function `_is_bunker` of the `Accounting` class calculate the bunker mode condition based on average values over skipped frames.
6) In the function `_get_finalization_data` of the `Accounting` class calculate the simulated share rate based on the current amount of ETH and shares.
7) In the function `_get_finalization_data` of the `Accounting` class calculate finalization batches based on the simulated share rate and current balances of vaults.
8) In the function `_get_finalization_data` of the `Accounting` class report current balances of EL and Withdrawal vaults, and current amount of shares to burn.

The on-chain oracle contract will:

1) In the function `_submitReport` of the `HashConsensus` contract produce the correct consensus for the current ref slot.
2) In the function `checkAccountingOracleReport` of the `OracleReportSanityChecker` contract check the report with sanity checker.
3) In the function `_handleOracleReport` of the `LIDO` contract calculate withdrawals and request shares to burn.
4) In the function `smoothenTokenRebase` of the `OracleReportSanityChecker` contract smoothen the token rebase.
5) In the function `_handleOracleReport` of the `LIDO` contract finalize rewards and withdrawals.
6) In the function `_handleOracleReport` of the `LIDO` contract burn shares.
7) In the function `_handleOracleReport` of the `LIDO` contract process rewards based on smoothed value of the rebase.
8) In the function `checkSimulatedShareRate` of the `OracleReportSanityChecker` contract check simulated share rate.

### 3.1.7 Consequences of reverts in `OracleReportSanityChecker`

## Description

The `OracleReportSanityChecker` performs several kind of checks leading to reverts:

- Check for monotonic increase in vaults balances and shares requested to burn: `_checkWithdrawalVaultBalance`, `_checkELRewardsVaultBalance`, `_checkSharesRequestedToBurn`.
- Check for correctness of data from CL layer: `_checkAppearedValidatorsChurnLimit`, `checkExitedValidatorsRatePerDay`.
- Check for correctness of data delivered by the oracle: `checkExitBusOracleReport`, `checkNodeOperatorsPerExtraDataItemCount`, `checkAccountingExtraDataListItemsCount`, `_checkLastFinalizableId`.
- Check against the limits set as protocol parameters: `_checkOneOffCLBalanceDecrease`, `_checkAnnualBalancesIncrease`, `_checkSimulatedShareRate`.

In case when any of the mentioned checks fail, it will result in a skipped report. While the protocol will not be supplied with the incorrect data, the condition leading to the revert should be addressed.

## Solution

Conditions of monotonic increase of EL and Withdrawal vaults balances and shares requested to burn are intra-frame checks of the correctness of supplied data. Report supplying the incorrect data will be skipped. The report in the next frame will report the actual data of the vault balances and shares and is expected to be correct, unless there is a factor that leads to draining of the vaults balances or an issue with the contracts or the oracle.

All conditions checked against limits, including the checks for correctness of CL and oracle data, are leading to the revert of the report and are fixed by updating the limits with an attempt to submit the report once again. If not done within the report deadline the report frame will be skipped.

# 3.1.8 Malicious DAO proposal

## Description

LDO is the governance token of Lido protocol. LDO holders can create DAO proposals and vote for them. The current LDO circulating supply is 876,809,361 LDO. And clear circulating supply (excluding DAO treasury) is about 765,351,649 LDO. It is about $1.5b at the current moment.
Also, there are the next parameters of Lido DAO proposals:

> For a DAO vote to pass, it's currently required that at least 5% of total voting power (common mistake is to think it's 5% of circulating token supply) supports the outcome AND more voting power supports the outcome than objects to it.

So if the attacker plans to accept a malicious DAO proposal, $0.75b in LDO tokens is enough to accept DAO proposal regardless of how other LDO holders vote. In practice, this value may be even less because not all holders will vote against it. As Lido protocol TVL is more than $11b at the current moment, this attack can be profitable.
For example, as most Lido contracts are upgradable, the attacker can implement a DAO proposal that upgrades some of the contracts to a malicious implementation.
The main problem for the attacker here is to accumulate the needed amount of LDO tokens. There is no such huge amount of LDO tokens on the open market. So even if the attacker has $0.75b, they cannot buy the needed amount of LDO tokens immediately. There are two options for the attacker:

1. They can accumulate tokens for a long time, which is a very complex and not predictable task for the attacker.
2. There is the next LDO initial token distribution:
    - DAO treasury - 36.32%

- Investors - 22.18%
- Validators and signature holders - 6.5%
- Initial Lido developers - 20%
- Founders and future employees - 15%

Thus limited set of vesting holders already have the needed amount of LDO tokens. Moreover, insufficient liquidity on the open market (for such huge volumes) works in the opposite direction in this case. It is not possible to swap $0.75b in LDO tokens for $0.75b in ETH, but in case of a successful attack TVL in ETH is under risk. This way Lido protocol security depends on the vesting holders' custody services.

Both of these options are highly unlikely, but worth to be taken into account.

# 4. CONCLUSION

The following table contains the total number of issues that were found during audit:

| Severity | FIXED | ACKNOWLEDGED | NO ISSUE | Total |
|---|---|---|---|---|
| CRITICAL | 0 | 0 | 0 | 0 |
| MAJOR | 0 | 7 | 0 | 7 |
| WARNING | 0 | 16 | 1 | 17 |
| INFO | 4 | 14 | 1 | 19 |
| TOTAL | 4 | 37 | 2 | 43 |

THANK YOU FOR CHOOSING

OXORIO