LIDO V2. OFF-CHAIN AUDIT REPORT

0X()RIO

MAY 10, 2023

CONTENTS

1 INTRO

OXORIO

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 - liquidity, 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

The following source files are included in the off-chain scope:

```
./src/services/withdrawal.py
./src/services/exit_order_iterator_state.py
./src/services/validator_state.py
./src/services/bunker_cases/abnormal_cl_rebase.py
./src/services/bunker_cases/midterm_slashing_penalty.py
./src/services/bunker_cases/typings.py
./src/services/bunker.py
./src/services/prediction.py
./src/services/exit_order_iterator.py
./src/services/safe_border.py
./src/variables.py
./src/providers/consistency.py
./src/providers/consensus/client.py
./src/providers/consensus/typings.py
./src/providers/keys/client.py
./src/providers/keys/typings.py
./src/providers/http_provider.py
./src/modules/ejector/ejector.py
./src/modules/ejector/typings.py
./src/modules/ejector/data_encode.py
./src/modules/submodules/oracle_module.py
./src/modules/submodules/typings.py
./src/modules/submodules/consensus.py
./src/modules/submodules/exceptions.py
./src/modules/accounting/extra_data.py
./src/modules/accounting/typings.py
./src/modules/accounting/accounting.py
./src/modules/checks/checks_module.py
./src/modules/checks/pytest.ini
./src/modules/checks/suites
./src/modules/checks/suites/keys_api.py
./src/modules/checks/suites/consensus_node.py
./src/modules/checks/suites/execution_node.py
./src/modules/checks/suites/conftest.py
./src/modules/checks/suites/common.py
./src/metrics/prometheus/duration_meter.py
./src/metrics/prometheus/ejector.py
./src/metrics/prometheus/basic.py
```

```
./src/metrics/prometheus/business.py
./src/metrics/prometheus/validators.py
./src/metrics/prometheus/accounting.py
./src/metrics/logging.py
./src/metrics/healthcheck_server.py
./src/utils/validator_state.py
./src/utils/build.py
./src/utils/events.py
./src/utils/abi.py
./src/utils/cache.py
./src/utils/types.py
./src/utils/slot.py
./src/utils/web3converter.py
./src/utils/dataclass.py
./src/utils/input.py
./src/utils/blockstamp.py
./src/typings.py
./src/web3py/typings.py
./src/web3py/extensions/keys_api.py
./src/web3py/extensions/fallback.py
./src/web3py/extensions/contracts.py
./src/web3py/extensions/lido_validators.py
./src/web3py/extensions/consensus.py
./src/web3py/extensions/tx_utils.py
./src/web3py/contract_tweak.py
./src/web3py/middleware.py
./src/constants.py
./src/main.py
```

The final commits to audit are:

- 2023-03-01: initial commit `20bc575f4fae7b3139b210b92f6d05a28215a9fb`
- 2023-03-20: `f3b314c31d9823f8c68b8ab3458f4c24a0eef004` updated Oracle 3.0.0-beta.1
- 2023-04-01: `9fd3a5dd8404a8c84d5664d059396f5649635208` updated Oracle 3.0.0-beta.2
- 2023-04-20: `73a655c66702ee45e6d08602b708e3441ac5f8c3` updated Oracle 3.0.0-rc.1 release
- 2023-05-05: `e50088b0cc51d3ae8954f5651348fb1405bdf61f` updated Oracle 3.0.0-rc.2 release.
- 2023-05-17: `44678954915b8291c949904c63de5e4e4983b427` updated Oracle 3.0.0 release.
- The docker image `sha256-d2ee5ecc78f8b991fcd2327e1d1bc84b8015aa7b8fde73e5ec0e702e6bec6c86` was verified

to match the codebase of the release commit `44678954915b8291c949904c63de5e4e4983b427`.

2 FINDINGS REPORT

OXORIO

2.1 CRITICAL

No issues found

2.2 MAJOR

2.2.1 Abnormal CL rebase condition of the bunker mode depends on the manipulatable value of the `WithdrawalVault` balance in `abnormal_cl_rebase.py`

SEVERITY	**MAJOR**
STATUS	**ACKNOWLEDGED**

Description

The function `_calculate_cl_rebase_between_blocks` in the `AbnormalClRebase` class uses the `WithdrawalVault` balance to calculate the difference in the rebase between blocks. `WithdrawalVault` is a contract, which balance can be manipulated by sending ether directly to the contract. This will increase the expected rebase value at the end of the frame and allow the attacker to prevent bunker mode from activation.

```
def _calculate_cl_rebase_between_blocks(
    self, prev_blockstamp: BlockStamp, ref_blockstamp: ReferenceBlockStamp
) -> Gwei:

    ref_lido_balance_with_vault = self._get_lido_validators_balance_with_vault(
        ref_blockstamp, self.lido_validators
    )
    prev_lido_balance_with_vault = self._get_lido_validators_balance_with_vault(
        prev_blockstamp, prev_lido_validators
    )


def _get_lido_validators_balance_with_vault(
    self, blockstamp: BlockStamp, lido_validators: list[LidoValidator]
) -> Gwei:

    real_cl_balance = AbnormalClRebase.calculate_validators_balance_sum(lido_validators)
    withdrawals_vault_balance = int(
        self.w3.from_wei(self.w3.lido_contracts.get_withdrawal_balance_no_cache(blockstamp),
"gwei")
```

```
    )
    return Gwei(real_cl_balance + withdrawals_vault_balance)
```

The scenario of the attack may be as follows:

1. Malicious actor has a withdrawal request for a large amount of ETH in the withdrawal queue sent in the current frame.
2. The CL conditions show abnormal situation that provokes a bunker mode activation based on the abnormal CL rebase condition.
3. The difference in the inequality of `frame_cl_rebase` < `normal_cl_rebase` check is insignificant.
4. By sending this difference to the `WithdrawalVault` the attacker may obtain 24 more hours for withdrawing the funds from the protocol.

Recommendation

We recommend to take the `WithdrawalVault` balance changes into account accurately, so that an unexpected increase in the balance can be matched against a calculated invariant.

Update

Lido's response

A risk's realization requires, **on top** of highly unlikely "bunker mode" premises, **additional preconditions with a low probability** (high amount of ETH in buffer and low cost for performing attack **simultaneously**) whereas the assessed impact on `stETH` holders is on par with the risk of APR decreasing due to new stake (when funds idle temporary) and **less than the standard deviation of APR**. Even though, considered mitigations either decrease user experience drastically or amplify the complexity of the protocol.

2.2.2 Abnormal CL rebase condition of the bunker mode depends on the manipulatable value of the validators real balances in `abnormal_cl_rebase.py`

SEVERITY	**MAJOR**
STATUS	**ACKNOWLEDGED**

Description

The function [`_calculate_cl_rebase_between_blocks`]() in the [`AbnormalClRebase`]() class uses real balances of Lido validators to calculate the condition of the bunker mode activation. Validator's balance can be increased by depositing more ETH, thereby affecting the value of the specific CL rebase, which is proportional to the mean total Lido real balance taken from the `REBASE_CHECK_NEAREST_EPOCH_DISTANCE` or `REBASE_CHECK_FAR_EPOCH_DISTANCE` and the current epoch of the Oracle frame. By adding funds to validators balance, it is possible to prevent bunker mode at the condition 3 from activation, allowing the withdrawal queue to process the withdrawal requests for one more Oracle frame.

```
def _calculate_cl_rebase_between_blocks(
    self, prev_blockstamp: BlockStamp, ref_blockstamp: ReferenceBlockStamp
) -> Gwei:

    ref_lido_balance_with_vault = self._get_lido_validators_balance_with_vault(
        ref_blockstamp, self.lido_validators
    )
    prev_lido_balance_with_vault = self._get_lido_validators_balance_with_vault(
        prev_blockstamp, prev_lido_validators
    )


def _get_lido_validators_balance_with_vault(
        self, blockstamp: BlockStamp, lido_validators: list[LidoValidator]
) -> Gwei:
    """
    Get Lido validator balance with withdrawals vault balance
    """
    real_cl_balance = AbnormalClRebase.calculate_validators_balance_sum(lido_validators)


def calculate_validators_balance_sum(validators: Sequence[Validator]) -> Gwei:
    return Gwei(sum(int(v.balance) for v in validators))
```

The scenario may be as follows:

1. Malicious actor has a withdrawal request for a large amount of ETH in the withdrawal queue sent in the current frame.
2. The CL conditions show abnormal situation that provokes bunker mode activation based on the condition 3.
3. The `nearest_cl_rebase` or `distant_cl_rebase` variables in the `nearest_cl_rebase` < `0` and `distant_cl_rebase` < `0` inequalities are only slightly lesser than `0`.
4. By depositing funds to Lido validators' balances the attacker may obtain 24 more hours for withdrawing the funds from the protocol.

Recommendation

We recommend taking into account changes in validators balances that may happen outside of the protocol, so that an unexpected increase in balances can be matched against a calculated invariant.

Update

Lido's response

A risk’s realization requires, **on top** of highly unlikely “bunker mode” premises, **additional preconditions with a low probability** (high amount of ETH in buffer and low cost for performing attack **simultaneously**) whereas the assessed impact on `stETH` holders is on par with the risk of APR decreasing due to new stake (when funds idle temporary) and **less than the standard deviation of APR**. Even though, considered mitigations either decrease user experience drastically or amplify the complexity of the protocol.

2.2.3 Normal CL rebase calculation depends on the manipulatable value of validators' effective balances in `abnormal_cl_rebase.py`

SEVERITY	**MAJOR**
STATUS	**ACKNOWLEDGED**

Description

The function `_calculate_lido_normal_cl_rebase` in the `AbnormalClRebase` class uses effective balances of Lido validators to calculate the condition of the bunker mode activation.

```
def _calculate_lido_normal_cl_rebase(self, blockstamp: ReferenceBlockStamp) -> Gwei:

    mean_sum_of_all_effective_balance = AbnormalClRebase.get_mean_sum_of_effective_balance(
        last_report_blockstamp, blockstamp, last_report_all_validators, self.all_validators
    )
    mean_sum_of_lido_effective_balance = AbnormalClRebase.get_mean_sum_of_effective_balance(
        last_report_blockstamp, blockstamp, last_report_lido_validators, self.lido_validators
    )
```

Method `calculate_active_effective_balance_sum`:

```
def calculate_active_effective_balance_sum(validators: Sequence[Validator], ref_epoch:
EpochNumber) -> Gwei:
    """
    Return the combined effective balance of the active validators from the given list
    """
    effective_balance_sum = 0

    for validator in validators:
        if is_active_validator(validator, ref_epoch):
            effective_balance_sum += int(validator.validator.effective_balance)

    return Gwei(effective_balance_sum)
```

Validator's balance can be increased by depositing more ETH, thereby affecting the value of the normal CL rebase, which is proportional to the mean total Lido real balance taken from the previous and the current epochs of the Oracle frame. By adding funds to validators balance, it is possible to prevent the bunker mode at the condition 3 from activation, allowing the withdrawal queue to process the withdrawal requests for one more Oracle frame.

The attack scenario may be as follows:

1. Malicious actor has a withdrawal request for a large amount of ETH in the withdrawal queue sent in the current frame.
2. The CL conditions show abnormal situation that provokes a bunker mode activation based on the condition 3.
3. The difference in the following inequality `diff_current_with_normal` > `self.b_conf.normalized_cl_reward_mistake_rate` of [`is_abnormal_cl_rebase`]() function is insignificant.
4. By depositing funds to Lido validators balances the attacker may obtain 24 more hours for withdrawing the funds from the protocol.

Recommendation

We recommend to take into account the changes in validators balances that may happen outside of the protocol, so that an unexpected increase in balances can be matched against a calculated invariant.

Update

Lido's response

A risk's realization requires, **on top** of highly unlikely "bunker mode" premises, **additional preconditions with a low probability** (high amount of ETH in buffer and low cost for performing attack **simultaneously**) whereas the assessed impact on `stETH` holders is on par with the risk of APR decreasing due to new stake (when funds idle temporary) and **less than the standard deviation of APR**. Even though, considered mitigations either decrease user experience drastically or amplify the complexity of the protocol.

2.2.4 The `finalization_share_rate` value can be artificially inflated in `accounting.py`

SEVERITY	**MAJOR**
STATUS	**ACKNOWLEDGED**

Description

In the function `_calculate_report` in `accounting.py` the value of `finalization_share_rate` is obtained through the call to the `simulate_full_rebase` function, which accumulates the `WithdrawalVault` and `LidoExecutionLayerRewardsVault` balances and balances of Lido validators, obtained by calling the `_get_consensus_lido_state` function:

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

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

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

By sending ETH to any of `WithdrawalVault`, `LidoExecutionLayerRewardsVault`, or validators' balances (real balances, which are not capped), the share rate obtained after the rebase simulation can be inflated.

While the funds used for the attack will be counted towards the protocol income, the scheme allows to predictably manipulate the value of the rebase, which can be used in complex attacks with other protocols that use stETH as a collateral or keep it in liquidity pools. Predictability of the Oracle report frames combined with the possibility to manipulate the reported rebase values opens a wide range of opportunities of malicious value extraction.

Recommendation

We recommend to take into account the changes of the `WithdrawalVault`, `LidoExecutionLayerRewardsVault`, and validators' balance accurately, so that an unexpected increase in balances can be matched against a calculated invariant. If possible, the effective balance of validators should be used for calculations, the real balance should be accounted separately and matched against an invariant to account for abrupt changes.

Update

Lido's response

The possible attack vectors can be divided into the following groups:

1. Top-up validators balance on the side of Consensus Layer
2. Top-up vaults balances on the side of Execution Layer.

There is a sanity check exists that prevents CL balance increase above the allowed level expressed in APR (and takes into account the elapsed time since the previous report). To address the second vector, there is a positive token rebase limiter which prevents Oracle report sandwiching and also suitable for this case (the limiter just smoothes rewards over multiple Oracle reports instead of step-wise addition).

Worth noting that the Lido-participating validators collect MEV which has outliers during the network congestion and market instability periods. It means that the provided scenario of extremely high EL rewards can be not an attack at all.

See also relevant GitHub issues #405 and #428 in the Lido DAO repo.

2.2.5 Bunker mode condition calculation is based on average values over skipped frames in `bunker.py`

SEVERITY	MAJOR
STATUS	ACKNOWLEDGED

Description

The function `get_cl_rebase_for_current_report` in the file `bunker.py` calculates conditions of the bunker mode activation based on the change in total pooled ether value. In case of skipped report frame bunker mode condition in the function `is_bunker_mode` is calculated in the function `get_cl_rebase_for_current_report` based on the change in total pooled ether between the last report slot and current ref slot. Large positive rebase

value in the skipped report frame may offset the negative rebase in the next frame, bunker mode will not get activated.
Skipped frames during bunker condition may be critical for the protocol and affect the bunker mode activation.

```
def get_cl_rebase_for_current_report(self, blockstamp: BlockStamp, simulated_cl_rebase:
LidoReportRebase) -> Gwei:
    """
    Get simulated Cl rebase and subtract total supply before report
    """
    logger.info({"msg": "Getting CL rebase for frame"})
    before_report_total_pooled_ether = self._get_total_supply(blockstamp)
    """
    Can't use from_wei - because rebase can be negative
    """
    frame_cl_rebase = (simulated_cl_rebase.post_total_pooled_ether -
before_report_total_pooled_ether) // GWEI_TO_WEI
    logger.info({"msg": f"Simulated CL rebase for frame: {frame_cl_rebase} Gwei"})
    return Gwei(frame_cl_rebase)

def _get_total_supply(self, blockstamp: BlockStamp) -> Gwei:
    return
self.w3.lido_contracts.lido.functions.totalSupply().call(block_identifier=blockstamp.block_hash)
```

Recommendation

We recommend to take the skipped frames into account when calculating the bunker mode condition. Another mitigation strategy can be the sequential Oracle report delivery, so that frame skipping will not be possible.

Update

Lido's response

In case of report frame is skipped, possible risk realization is not activating bunker mode in conditions when, if the report were not skipped, “bunker mode” would’ve been activated.
The conditions could be:

- **Negative rebase in the last frame (24 hours) – that risk is mitigated**. The previous skipped frame could compensate for the last 24 hours, but “bunker mode” would still trigger due to the condition of negative rebase in the last 23 or 1 epoch (as the last frame is negative) and abnormal CL rebase (as trigger condition is 10% and negative rebase observed through half of the window between last report slot and current ref slot).
- **Negative rebase in last 23 or 1 epochs and abnormal rebase in the last 24 hours – that risk is acknowledged**. As the report frame is wider than skipped report, abnormal CL rebase calculation would consider skipped frame, and stable good performance there

could lead to the condition not triggering when in the last 24 hours problems were observed. For actual risk realization (not triggering “bunker mode”) on top of the low probability of skipping the Oracle report (probability less than 0.5%), the decrease in CL rebase in the last 24 hours should be not more than 20% (or skipped frame couldn’t compensate that) - with such an extreme combination of conditions that risk could be accepted on Oracle level and could be mitigated with Gateseal, as missing report incident would trigger additional focus with monitoring.

2.3 WARNING

2.3.1 Withdrawal queue finalization sequence can be manipulated in `withdrawal.py`

SEVERITY	**WARNING**
STATUS	**ACKNOWLEDGED**

Description

The function `get_finalization_batches` in the file `withdrawal.py` uses the `WithdrawalVault` and `LidoExecutionLayerRewardsVault` balances to calculate the last finalizable requests through the `WithdrawalQueue` contract.

The function `simulate_full_rebase` obtains the value of EL rewards vault balance:

```
def simulate_full_rebase(self, blockstamp: ReferenceBlockStamp) -> LidoReportRebase:
    el_rewards = self.w3.lido_contracts.get_el_vault_balance(blockstamp)
    return self.simulate_rebase_after_report(blockstamp, el_rewards=el_rewards)
```

which gets passed to the `get_finalization_batches` function and then is used to calculate ETH available to finalize withdrawal requests.

```
def get_finalization_batches(
    self,
    is_bunker_mode: bool,
    share_rate: int,
    withdrawal_vault_balance: Wei,
    el_rewards_vault_balance: Wei
) -> list[int]:

    available_eth = self._get_available_eth(withdrawal_vault_balance, el_rewards_vault_balance)

    return self._calculate_finalization_batches(share_rate, available_eth,
withdrawable_until_timestamp)
```

This allows the attacker to advance finalization queue by sending funds directly to the `LidoExecutionLayerRewardsVault` contract.

The same scheme applies to the `WithdrawalVault` contract.

The scenario may be as follows:

1. All the available funds (Withdrawal vault, EL rewards vault, buffered ether) allow to process $100M withdrawals in the withdrawal queue.
2. Queue contains withdrawals requests for the amount of $50M for the next frame.
3. Malicious actor may want to withdraw $50,000,001 within the next frame, which is more than the total available balance in the protocol. In the best case, they will have to wait 24 hours more for another Oracle report.
4. By sending $1 to the `WithdrawalVault` or `LidoExecutionLayerRewardsVault` they can make their request be processed within the next frame.

Recommendation

We recommend taking into account the `WithdrawalVault` and the `LidoExecutionLayerRewardsVault` balance changes accurately, so that an unexpected increase in balance can be matched against a calculated invariant.

Update

Lido's response

An instant withdrawal request finalization is prevented by requiring a withdrawal request placed not before the report’s collecting timeframe with an additional time margin (also known as a safe border across the Oracle codebase).

Apart from this, the proposed top-up scenario follows the intended design because, for example, these extra funds have no significant differences from MEV which is collected by the Lido-participating validators. To finalize withdrawals, the protocol is allowed to use all available funds from vaults and buffers unless bunker mode has been activated.

Moreover, the withdrawing staker can do the new `submit` method call to enrich the buffer with native ether while getting minted `stETH` in return. By doing this, one can achieve an effect similar to a partial withdrawal request fulfillment which doesn’t pose risks to the protocol or its stakers.

2.3.2 `latest_to_exit_validators_count` takes a non-zero value when there are no validators to exit in `ejector.py`

SEVERITY	WARNING
STATUS	FIXED

Description

In the function `_get_predicted_withdrawable_epoch` in the file `ejector.py` the maximum value of `exit_epoch` among validators (`max_exit_epoch_number`) can be less than `blockstamp.ref_epoch + 1 + MAX_SEED_LOOKAHEAD` (according to `compute_activation_exit_epoch` function). This can happen when all validators have reached their `exit_epochs` and none of the validators have initiated an exit yet.

In this case `max_exit_epoch_number` sets to `blockstamp.ref_epoch + 1 + MAX_SEED_LOOKAHEAD`, but at the same time `latest_to_exit_validators_count` is left untouched with a non-zero value:

```
max_exit_epoch_number, latest_to_exit_validators_count =
self._get_latest_exit_epoch(blockstamp)

max_exit_epoch_number = max(
    max_exit_epoch_number,
    self.compute_activation_exit_epoch(blockstamp),
)

churn_limit = self._get_churn_limit(blockstamp)

remain_exits_capacity_for_epoch = churn_limit - latest_to_exit_validators_count
```

It leads to the reduction of the capacity of the exit queue.

Recommendation

We recommend setting the `latest_to_exit_validators_count` to zero when there are no validators in the exit queue:

```
activation_exit_epoch = self.compute_activation_exit_epoch(blockstamp)
if max_exit_epoch_number < activation_exit_epoch:
    max_exit_epoch_number = activation_exit_epoch
    latest_to_exit_validators_count = 0
```

Update

Lido's response

Fixed: https://github.com/lidofinance/lido-oracle/pull/376

2.4 INFO

2.4.1 Repetitive deposits and withdrawals are able to exhaust available validators' keys.

SEVERITY	**INFO**
STATUS	**ACKNOWLEDGED**

Description

Each deposit of 32 ETH requires an activation of a new validator and each withdrawal of 32 ETH requires an exit of a validator unless there is enough ETH in the buffer and EL contracts that collect the rewards. Repetitive deposits of large amount of ETH will steadily exhaust the validator sets managed by node operators, requiring them to add more validators with new keys to the protocol. If all the set of available validators will be exhausted, deposits to the protocol will become impossible until new set of public keys will be added.

Possible scenario:

1. Lido has a single node operator with 10K validators and 5K of them are active.
2. The attacker deposits 32K ETH activating 1000 validators then immediately sends a withdraw request.
3. Repeating the step 2 five times will exhaust the whole validator set and will require adding more new validator public keys to the protocol.

Recommendation

We recommend introducing a delay between deposit and withdrawal in proportion to the deposit amount.

Update

Lido's response

The overall scenario is coherent with Ethereum staking approaches in general. Nevertheless, the following considerations make the attack surface tolerable:

- The Lido protocol has a daily stake limit (it’s 150k ETH (4687 keys) per day) and there is an alerting system for large stake and available keys amount; moreover, the Node Operator Management team keeps track of keys with a reasonable reserve (40k keys at the time of response) for several days of max stake in advance

- The attacker have to block their ETH for an undetermenistic time period since the buffer is one of the sources of finalizing requests, hence the attacker must wait further till real deposit of the keys and only then issue a withdrawal request
- Waiting for funds isn't free for the attacker (opportunity costs due to discontinued token rebases for them, exposure to additional risks)

2.4.2 Malicious activity of a node operator may affect the normal workflow of the protocol or result in a financial loss

SEVERITY	**INFO**
STATUS	**ACKNOWLEDGED**

Description

Node operators have total control of the validators infrastructure, being incentivized for a good behavior only by the reward they get from the protocol. As node operators do not take any risks related to any malicious activity performed by their validators on the consensus layer, they may use their validators to create adverse conditions on the consensus layer, that, for example, may lead to a mass slashing. This will allow node operator to activate the bunker mode, blocking the withdrawals from the protocol and leading to a financial loss.

While Lido protocol introduces decentralization mechanics so that no node operator concentrates more than 1% of the staking power of the protocol, even in that case node operator operates a significant amount of financial power which is under a risk of being slashed with a node operator not taking any material responsibility for the malicious activity it may perform, other than the loss of the node operator reward received from the protocol.

There is a significant degree of trust to the node operator legitimacy and the process of adding a node operator to the protocol hardly guarantees the node operator long-term trustworthiness.

Recommendation

We recommend introducing a stake of a node operator, analogous to the stake on the consensus layer, so that any malicious activity of the node operator will result in financial loss that raises the cost of the attack.

Update

Lido's response

Node operators are trusted entities inside the Lido ecosystem. They are responsible for correctly running the validators as well as following the established Lido DAO policies (e.g., accruing MEV rewards to the dedicated contract using mev-boost infrastructure).

The risk of node operator misbehavior is mitigated by maintaining diversified validators set and incentives structure (protocol fee) with monitoring and DAO governance processes. There is a set of policies and management actions:

- onboarding new NOs to decentralize further;
- limiting the stake amount per a single NO, especially during the onboarding period;
- maintaining dashboards and tools to monitor network participation performance, MEV and priority fee distribution (open-sourced https://github.com/lidofinance/ethereum-validators-monitoring)
- protocol fee is proved to be a good-behavior incentive in a long-run;
- there is a set of on-chain levers can be used by the Lido governance to penalize the misbehaving entities: excluding new stake, disabling fee distribution, prioritized exiting of validators.

The later option will be reinforced once withdrawal credentials-initiated exits would be implemented in Ethereum.

2.4.3 A protocol user and a node operator can be a single entity performing coordinated actions in the protocol

SEVERITY	**INFO**
STATUS	**ACKNOWLEDGED**

Description

A node operator that also has a large deposit in the protocol can coordinate its actions between the consensus and execution layer leading to a complex scenarios that do not follow the regular workflow of the protocol.

For example, a node operator can exit a large number of validators prior to the Oracle report, so that it will allow to withdraw large amount of funds within a short time frame, which otherwise would require waiting for at least one Oracle frame while Oracle will process Lido validators exits.

Recommendation

We recommend to introduce a mode of the protocol that blocks deposits and withdrawals in case of an unusual activity of node operators on the consensus layer.

Update

Lido's response

It’s practically possible that a node operator on the one hand can be a withdrawing whale staker on another.

There are two considerations in general that discourage this approach of self-initiated exits:

- an instant withdrawal request finalization is prevented by requiring a withdrawal request placed not before the report’s collecting timeframe with an additional time margin (also known as a safe border across the Oracle codebase).
- the exit bus Oracle reports several times a day while it takes 1-7 days for a validator to be fully withdrawn depending on the sweep in the chain. Hence, having a few Oracle frames early or later makes lesser impact than the Ethereum sweep current queue state (validator index cursor) and its length.

Nevertheless, the incentive to cover up the withdrawing amount with their own exited validators is not straightforward since the overall intent to withdraw a significant amount of funds is legit by design and can be performed using exits of other node operators as well.

2.4.4 Prediction of `withdrawal_epoch` without taking `activation_epoch` into account in `ejector.py`

SEVERITY	INFO
STATUS	ACKNOWLEDGED

Description

In the `_get_predicted_withdrawable_epoch` function in the `ejector.py` file among the `validators_to_eject_count` number of validators there may be those which have the `activation_epoch` set very far in the future, but the function tries to predict the withdrawal epoch without taking the `activation_epoch` into account.

Consider an example. We want to predict the withdrawal epoch when ejecting 5 validators. Their `activation_epoch` are `ref_epoch+50`. Suppose the other variables are:

```
max_exit_epoch_number = ref_epoch+10
latest_to_exit_validators_count = 1
churn_limit = 8
```

The function will return the epoch when all validators will supposedly be withdrawable:

```
(ref_epoch+10) + MIN_VALIDATOR_WITHDRAWABILITY_DELAY
```

While validators can actually be withdrawable no sooner than:

```
(ref_epoch+50 + SHARD_COMMITTEE_PERIOD + 1 + MAX_SEED_LOOKAHEAD) +
MIN_VALIDATOR_WITHDRAWABILITY_DELAY
```

Since we need to wait at least until the validators "are eligible" to initiate the exit.

It leads to creation of more requests to exit than needed, because with a smaller `withdrawal_epoch` value, fewer [`future_withdrawals`] will be calculated and more validators will need to be ejected to cover amount of withdrawal requests.

```
withdrawal_epoch = self._get_predicted_withdrawable_epoch(blockstamp, len(validators_to_eject)
+ len(validators_going_to_exit) + 1)

future_withdrawals = self._get_withdrawable_lido_validators_balance(blockstamp,
withdrawal_epoch)
expected_balance = (
    future_withdrawals +

)
if expected_balance >= to_withdraw_amount:
    return validators_to_eject
```

Recommendation

We recommend revising the calculation of the `withdrawal_epoch` in the presence of "is not eligible" validators (which have `activation_epoch + SHARD_COMMITTEE_PERIOD` > `ref_epoch + 1 + MAX_SEED_LOOKAHEAD`) in order to prevent unnecessary ejections.

Update

Lido's response

We acknowledge this finding, but we have decided not to implement the recommended solution. The scenario in question is an uncommon edge case, and the likelihood of it happening is extremely low. Meanwhile we believe that withdrawing additional validators is not a significant concern, and that validator rotation is beneficial for the protocol.

3 AUDITED INCIDENTS REPORT

OXORIO

3.1 AUDITED INCIDENTS REPORT

3.1.1 All Oracles at different points in time must build the same report for a particular epoch

Description

In the function `submitReportData` of the `AccountingOracle` contract, Oracles must submit the same report data at different points in time on all servers in order to keep track of historical share rates to values of a certain epoch. It may affect the calculation of average values in the bunker mode.

Solution

In the function `_calculate_report` of the `Accounting` class, during reports calculation, Oracles rely on the same slot - `ref_slot`, so their report data is the same. If the data of the reports still differ, then the final report is decided by consensus. As a result, the data of only one report is processed on-chain.

The `ref_slot` itself is actually calculated from the current Oracle time in the `_getCurrentFrame` function.

The `ref_slot` is determined by using the `get_blockstamp_for_report` function. In it, the Oracle checks that:

- `CONTRACT_VERSION` and `CONSENSUS_VERSION` match on-chain versions;
- report not created yet;
- have the right to submit reports;
- current `ref_slot` finalized;
- deadline not reached.

If one of the checks fails, the Oracle will not create a report for this `ref_slot`.

3.1.2 Validator did not exit by Ejector's request to exit

Description

In the function `_processExitRequestsList` of the `ValidatorsExitBusOracle` contract, a `ValidatorExitRequest` event is emitted to request an exit for the next validator, but at the consensus level, the validator may not exit during the frame.

Solution

If the Ejector made a request to exit for validator, but the node operator did not initiate exit on that validator until the Ejector's next report, then the Ejector considers the validator as "delayed" when calls the function `count_operator_delayed_validators` of the `ExitOrderIteratorStateService` class.

As a consequence, node operators with the least number of "delayed" validators will be selected first in the function `_predicates` to eject validators.

If the node operator did not initiate the exit of the validator before the `ref_slot` of the next Accounting frame, then such validators will be included in the Accounting report as "stuck" by the function `get_lido_newly_stuck_validators` of the `LidoValidatorStateService` class. Data about stuck validators is sent by the Accounting module as extra data.

In this case, the node operator will receive a penalty until it makes a "refund" for these validators by the function `updateRefundedValidatorsCount` of the `StakingRouter` contract.

4. CONCLUSION

OXORIO

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

Severity	FIXED	ACKNOWLEDGED	Total
CRITICAL	0	0	0
MAJOR	0	5	5
WARNING	1	1	2
INFO	0	4	4
TOTAL	1	10	11

THANK YOU FOR CHOOSING

0XORIO