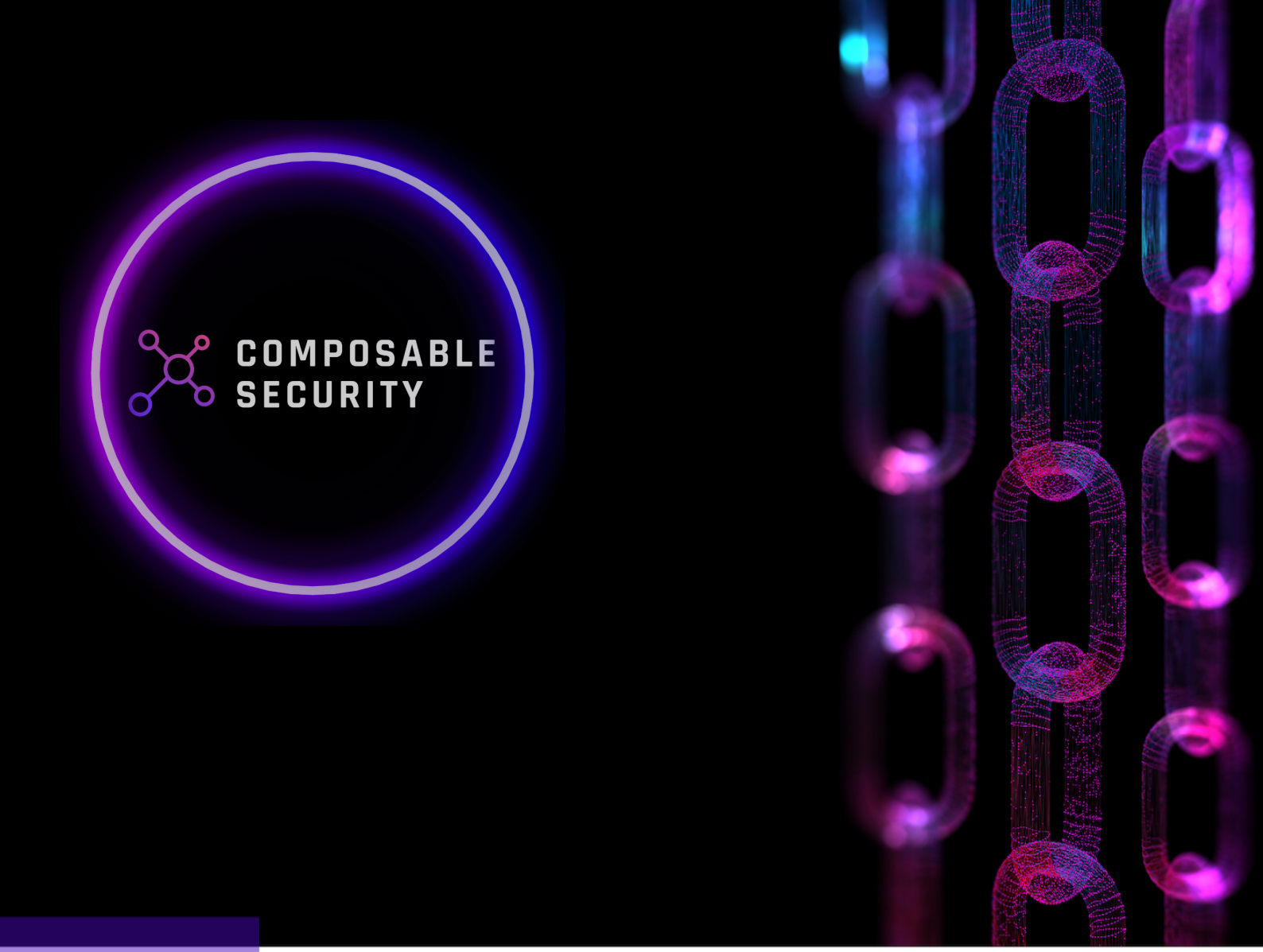


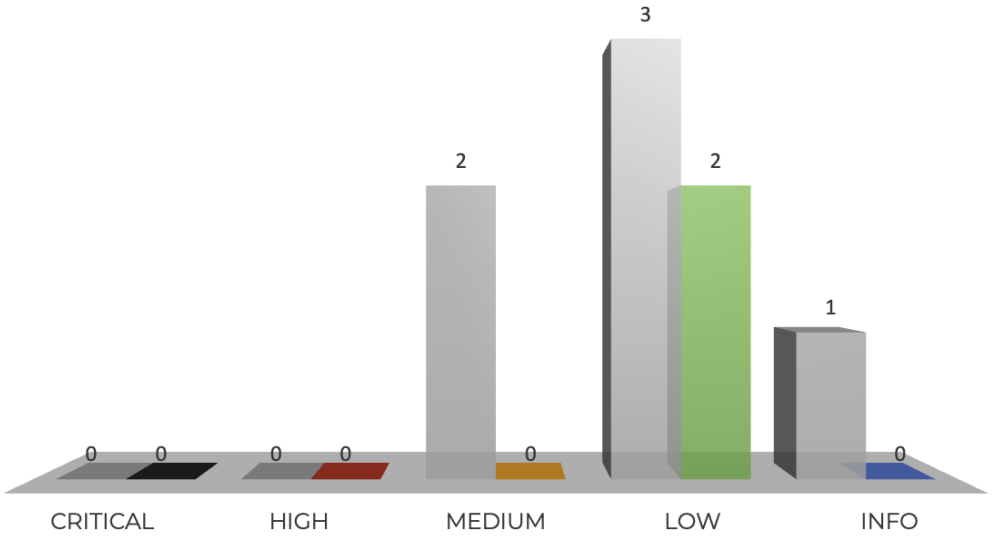
REPORT

Security review for Lido

Prepared by: Composable Security
Report ID: LDO-0dbc1d8a
Test time period: 2025-02-10 - 2025-03-10
Retest time period: 2025-03-17 - 2025-04-03
Report date: 2025-04-07
Version: 1.3
Visit: composable-security.com

Contents

1. Retest summary (2025-04-03)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2025-02-10 and 2025-03-10) were removed correctly and no longer appear in the code.

The current status of the detected issues is as follows:

- 2 vulnerabilities with a **medium** impact on risk have been fully fixed.
- 3 vulnerabilities with a **low** impact on risk were handled as follows:
  - 2 have been acknowledged,
  - 1 has been fixed.
- 1 security **recommendation** has been implemented.
- During the retest, 2 additional fixes introduced by the Lido team, were also verified:
  - A more precise check for which algorithm, pre- or post-Pectra, to compute the correlated slashing penalty. fc8ba6e6857b46d88df06787291baa8f6d400683
  - Making midterm penalty calculation consistent with specification by using slashing vector instead of effective balance. 14e28362038efb9634bd46a76f2766e4629cab1f

1.2. Scope

The retest scope included the same files, on a different commit in the same repository.

GitHub repository: https://github.com/lidofinance/lido-oracle

CommitID: 41f3f9671ea7e349e048c3ac47264a118c7983a8

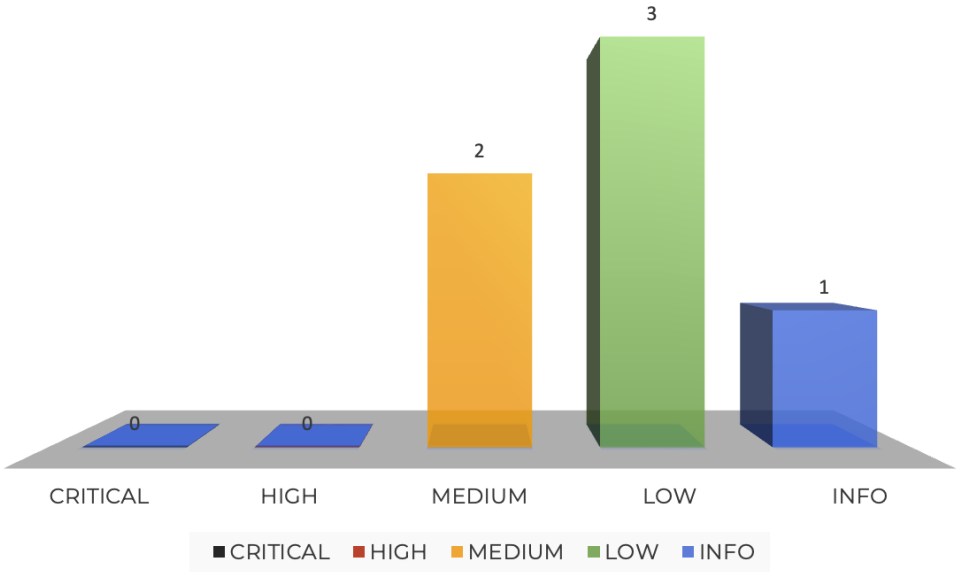
1.3. Deployments

After the security review conducted, we verified that the Dockerfile used to build an image uses the source code reviewed during retest. The published image with the manifest digest **sha256:b15bb7fcd19b6368cf8f8498b52bb527b1da1f36227a44b47b3a5709db8b4e46** corresponds to the commitID: **41f3f9671ea7e349e048c3ac47264a118c7983a8**.

2. Current findings status

ID	Severity	Vulnerability	Status
LDO-0dbc1d8a-M01	MEDIUM	Uncaught exception on creating ejection report	FIXED
LDO-0dbc1d8a-M02	MEDIUM	Node Operator can exit and slash their validator to move withdrawal finalization border epoch	FIXED
LDO-0dbc1d8a-L01	LOW	Invalid number of delayed validators stored in Prometheus	FIXED
LDO-0dbc1d8a-L02	LOW	Loss of rewards in case of omitted frame	ACKNOWLEDGED
LDO-0dbc1d8a-L03	LOW	Lack of rounding epoch to frame in rebase border epoch calculation	ACKNOWLEDGED
ID	**Severity**	**Recommendation**	**Status**
LDO-0dbc1d8a-R01	INFO	Eject validators exitable in upcoming oracle frame	IMPLEMENTED

3. Security review summary (2025-03-10)



3.1. Client project

The **Lido Oracle - Update V5** project is a core component of the Lido staking ecosystem, responsible for aggregating, verifying, and relaying critical operational data that supports the protocol's decentralized staking mechanism.

Below is an overview of its key components:

- **Accounting Oracle:** Focuses on tracking staking rewards, fees, and other financial metrics. This module is crucial for ensuring that the economic aspects of the protocol remain consistent and transparent.
- **Validators Exit Bus Oracle:** VEBO is an oracle that ejects Lido validators when the protocol requires additional funds to process user withdrawals. This component ensures that any changes in validator status are accurately reflected by generating requests to exit validators to fulfill the withdrawals demand.
- **CSM Oracle:** Integrates with Community Staking Module to report module-specific metrics, that is the rewards generated by CSM validators.

The **Oracle V5** has been engineered to ensure that the Oracle continues to operate as intended following the Pectra upgrade, with a strong focus on addressing the challenges during the immediate transition period.

The update incorporates changes that maintain the accuracy and reliability of critical operational data even in the face of significant protocol changes.

3.2. Results

The **Lido** engaged Composable Security to review the security of **Lido Oracle - Update V5**. Composable Security conducted this assessment over 4 weeks with 2 engineers.

The summary of findings is as follows:

- Before starting the review, the commitID was updated to *0dbc1d8a2acf069075118e94770c20de2c1de3d7*.
- There were **2** vulnerabilities with a **medium** impact on risk identified. Their potential consequences are:
  - Halting the report generation (LDO-0dbc1d8a-M01).
  - Bypassing the border epoch as determined by the mass slashing event (LDO-0dbc1d8a-M02).
- **Three** vulnerabilities with a **low** impact on risk were identified.
- One recommendation has been reported.
- The team was very involved in the audit and helped analyze potential attack scenarios. They took care of preparing the detailed documentation and provided answers to all the questions during the security review.
- Right after starting the security review, the Lido team reported an issue in the bunker service related to the incorrect calculation of CL rebase between blocks. Therefore, the issue has not been included in this report.
- The Lido Oracle codebase has been actively developed during the security review, and the audit team did not focus on changes made during this period.
- In addition to searching for vulnerabilities in the code, the team also focused on analyzing compatibility with Pectra. The analysis performed did not detect any unhandled inconsistencies. The detailed results of the analysis can be found in Pectra requirements that might affect Lido.

Composable Security recommends that **Lido** complete the following:

- Address all reported issues.
- Extend peer reviews with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.

3.3. Scope

The scope of the tests included selected files from the following repository.

GitHub repository: https://github.com/lidofinance/lido-oracle
CommitID: 0dbc1d8a2acf069075118e94770c20de2c1de3d7

The detailed scope of the tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that code is written according to security best practices.
- Identify security issues and potential threats both for **Lido** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subjects of the test were selected files from the **Lido** repository.

GitHub repository:
https://github.com/lidofinance/lido-oracle
CommitID: 0dbc1d8a2acf069075118e94770c20de2c1de3d7

Files in scope:

```
src
├── __init__.py
├── constants.py
├── main.py
├── metrics/*.py
├── modules/*.py
├── providers/*.py
├── services/*.py
├── types.py
├── utils/*.py
├── variables.py
└── web3py/*.py
```

Documentation:

- EIP-7600: Hardfork Meta - Pectra
- Lido Docs
- LIP-27 Ensuring Compatibility with Ethereum's Pectra Upgrade
- "Bunker mode": what it is and how it works

4.3. Threat analysis

This section summarizes the potential threats that were identified during the initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Report's submitted data, including:
  - stETH/ETH ratio components,
  - Withdrawal requests queue border,
  - Ejected validators,
- Ether managed by the protocol,
- Bonds,
- Protocols availability.

Threats and potential attackers goals:

- Manipulation of the stETH/ETH rate to steal ETH.
- Lock users' funds in the contract.
- Bonds theft.
- Running validators without required bond.
- Bypassing business logic limits (e.g. the withdrawal request delay).
- Pausing the protocol by maliciously activating bunker mode.
- Denial of Service (e.g. due to Oracle malfunction).

Potential vulnerable and attack scenarios to achieve the indicated attacker's goals:

- Excluding validators from the total ETH balance calculation.
- Front-running deposits to preset withdrawal credentials.
- Retrieving on-chain data from incorrect block via archive node.
- Invalid calculation of validator parameters after Pectra upgrade (e.g. max effective balance).
- Using post-Pectra algorithms for pre-Pectra blocks.
- Intentional slashing by an operator to break ejector module.
- Intentional slashing by an operator to manipulate the safe border calculation.
- Intentional exit of validator by its operator to break the ejector module.
- Take advantage of arithmetic errors.
- Incorrect selection of validators to be ejected (e.g. re-ejecting validators).
- Keeping ETH on validators forced to exit.
- Missing a report for one or more frames.
- Unprocessed changes introduced in the fork upgrade function (e.g. populating `pending_deposits`).
- Missing ETH balances of validators not yet added to registry but already submitted via

Deposit contract.
- Incorrect calculation of future activation and withdrawable epochs (changed in Pectra fork).
- Lack of proper update of consensus version parameter.
- Influence or bypass the business logic of the system.
- Inconsistency with documentation.
- Design issues.
- Uncaught exceptions.

4.4. Testing methodology

Security review was performed using the following methods:

- Q&A sessions with the **Lido** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests.
- **Manual review of the code.**

4.5. Disclaimer

Security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[LDO-0dbc1d8a-M01] Uncaught exception on creating ejection report

MEDIUM FIXED

Retest (2025-03-17)

The vulnerability was removed in **d57c554a6eb2da5b7fc13a9382e09a88c64b58ea** and **6f9656acedae9b4319d2137b5d63cad398fa88be**.

Before: The code checked if `no_stats.predictable_validators`, meaning it looked at a count that may not reflect the immediate availability of validators.

After: The code retrieves the group identifier (`gid`) for the node operator and then checks whether the `exitable_validators` dictionary for that `gid` is non-empty.

Additionaly, the test scenario was added to validate the improved forced exit logic in the iterator, ensuring that a node operator that appears to require a forced exit (because its stats indicate a surplus of validators relative to its forced exit target) is not processed if there are no exitable validators available.

Affected files

- exit_order_iterator.py#L298-L323
- exit_order_iterator.py#L133-L136
- exit_order_iterator.py#L162-L171

Description

The `get_remaining_forced_validators` function is designed to remove validators that are required to exit, even if the withdrawal queue is fully claimable.

```
def get_remaining_forced_validators(self) -> list[tuple[NodeOperatorGlobalIndex,
    LidoValidator]]:
        """
        Returns a list of validators from NOs that are requested for forced exit.
        This includes an additional scenario where enough validators have been
            ejected to fulfill the withdrawal requests,
        but forced ejections are still necessary.
```

```
        """
        result: list[tuple[NodeOperatorGlobalIndex, LidoValidator]] = []

        # Extra validators limited by VEBO report
        while self.index != self.max_validators_to_exit:
            for no_stats in sorted(self.node_operators_stats.values(), key=self.
                no_remaining_forced_predicate):
                if self._no_force_predicate(no_stats) == 0:
                    # The current and all subsequent NOs in the list has no forced
                        validators to exit. Cycle done
                    return result

                if no_stats.predictable_validators:
                    # When found Node Operator
                    self.index += 1
                    gid = (
                        no_stats.node_operator.staking_module.id,
                        no_stats.node_operator.id,
                    )
                    result.append((gid, self._eject_validator(gid)))
                    break

        return result
```

This function iterates through all node operators, sorting them in descending order based on the number of validators awaiting forced exit. This count is determined by subtracting the `force_exit_to` limit from `predictable_validators`.

```
@staticmethod
def _no_force_predicate(node_operator: NodeOperatorStats) -> int:
    return ValidatorExitIterator._get_expected_validators_diff(
        node_operator.predictable_validators,
        node_operator.force_exit_to,
    )
```

If there is no difference in the count, the loop terminates with a `return` statement. However, if a difference is detected, the function proceeds to remove the first validator from the `exitable_validators` list using the `_eject_validator` function.

The issue arises when a Node Operator has transient validators, which are counted in the `predictable_validators` but not included in `exitable_validators`. Consequently, if the function attempts to remove a validator when `exitable_validators` is empty, an `IndexOutOfBounds` exception occurs, halting the report generation.

Vulnerable scenario

The following steps lead to the described issue:

① A Node Operator has 1 exitable validator and 2 transient validators with no `force_exit_to` limit set.
② The `force_exit_to` limit is adjusted to 1, necessitating the exit of 2 operators while leaving one.
③ The Ejector calculates the number of predictable validators as 3.
④ The difference between the predictable validators and the limit results in 3 - 1 = 2.
⑤ The first (and only) validator from the exitable operators is ejected.
⑥ In the next iteration of the `while` loop, the difference becomes 2 - 1 = 1.
⑦ The Ejector needs to eject another validator, but when it attempts to remove the first element from the now-empty `exitable_validators` list, an uncaught exception is raised, interrupting the report processing.

Result: The report generation cannot be completed and submitted due to this error.

Recommendation

- The `while` loop should include a check to ensure that exitable validators exist for the Node Operator that has not reached the `force_exit_to` limit.
- Additionally, provisions should be made to prevent the possibility of an infinite loop when a Node Operator has not reached the `force_exit_to` limit and lacks exitable operators. Detecting this scenario is essential to avoid a Denial of Service situation.

References

1. SCSVS G4: Business logic

[LDO-0dbc1d8a-M02] Node Operator can exit and slash their validator to move withdrawal finalization border epoch

MEDIUM FIXED

Retest (2025-03-17)

The vulnerability has been removed in **67f36f0ecf92ede259ba65c6b2ef431f05672e62**.

By eliminating the filtering step, the algorithm now considers all slashed valida-

tors that are still not withdrawable rather than only those with the absolute earliest exit epoch. This means the computed "earliest predicted slashed epoch" now reflects a broader view of the validators' exit characteristics rather than being solely determined by a earliest exit epoch.

Additionally, the test scenario was added to validate ensure that there is no filtering applied to the inputs of `_find_earliest_slashed_epoch_rounded_to_frame`.

Affected files

- safe_border.py#L163-L174
- safe_border.py#L150

Description

During a mass slashing event, the withdrawal queue may delay finalization of withdrawal requests in queue.
The `get_safe_border_epoch` function, which determines the border epoch for unlocking withdrawal requests, relies on the `_get_associated_slashings_border_epoch` function to compute this epoch based on the ongoing mass slashing activity.

In a mass slashing scenario, this function returns the most recent epoch before the slashing commenced. This epoch is determined as the earliest `slashed_epoch` among all validators with incomplete slashings at the time of the `reference_epoch`, rounded to the beginning of the most recent oracle report frame.

A challenge arises because the `slashed_epoch` field is not present in the validator data; it must be inferred. If the difference between a validator's `withdrawable_epoch` and `exit_epoch` equals `MIN_VALIDATOR_WITHDRAWABILITY_DELAY`, it suggests a large exit queue, making it difficult to ascertain the precise `slashed_epoch`. Conversely, if this condition does not hold, the `slashed_epoch` can be estimated as `withdrawable_epoch - EPOCHS_PER_SLASHINGS_VECTOR` The core problem lies in the Oracle's assumption that the earliest `exit_epoch` corresponds to one of the validators involved in the mass slashing. A dishonest operator could anticipate a mass slashing event and proactively exit one of their validators.

Consequently, when the mass slashing occurs, the border epoch is calculated using the exit epochs of the validators being slashed. If the operator slashes their validator before it officially exits, the earliest `exit_epoch` will belong to that validator, resulting in a situation where the safe border is determined by its intentional slashing.

This effectively allows withdrawals submitted after the onset of the mass slashing to be unlocked.

Attack scenario

The attackers might take the following steps in sequence:

① A sophisticated operator managing multiple validators detects a potential mass slashing event.
② The operator exits one of their validators at time `t0`. This sets the exit epoch to `t10 = t0 + 10` and the withdrawable epoch to `t20 = t0 + 20` (reflecting the minimum withdrawability delay).
③ A mass slashing event begins at time `t5`. The first slashed validators have an exit epoch set at `t15 = t5 + 10` and a withdrawable epoch at `t105 = t5 + 100` (based on epochs per slashing).
④ The operator confirms the mass slashing and submits a withdrawal request at time `t8`, but it is blocked due to the active mass slashing, which establishes the safe border at `t5` (the slashed epoch of a validator with the earliest exit epoch).
⑤ The operator proceeds to slash the validator from step 2 at time `t9`. Consequently, the earliest `exit_epoch` will now be associated with the operator's validator, and the slashed epoch becomes `t9`. This action unlocks all withdrawal requests made between `t5` and `t9`, which should have been blocked.

Result of the attack: Bypassing the border epoch as determined by the mass slashing event.

Recommendation

During a mass slashing event, consider using the maximum `slashed_epoch` for all validators that are subject to slashing and have not yet reached the withdrawable state, rather than limiting the calculation to those with the earliest exit epoch.

References

1. SCSVS G4: Business logic

[LDO-0dbc1d8a-L01] Invalid number of delayed validators stored in Prometheus

LOW FIXED

Retest (2025-03-17)

The vulnerability has been removed in **07ea65e97a8c9581a1aeb3eaa3b6847b78e1c17e**. Added a new condition to verify the eligibility of validators to exit.

Affected files

- validator_state.py#L205-L217
- exit_order_iterator.py#L166-L169

Description

The variable `ACCOUNTING_DELAYED_VALIDATORS` is intended to track the number of delayed validators in Prometheus during the construction of the Ejector report.

However, the `is_validator_delayed` function, which identifies these delayed validators, currently lacks a check on whether the validators identified for exit were able to complete the exit process.

This oversight result in validators being marked as delayed even if the Node Operator could not process their exit requests, for instance, due to limits set by the `SHARD_COMMITTEE_PERIOD`.

The validator can be selected to be exited even when the period defined by `SHARD_COMMITTEE_PERIOD` variable has not passed. The `is_validator_exitable` function checks only if the validator is active.

Result: Inaccurate metrics regarding the number of delayed validators.

Recommendation

To ensure accuracy in the `is_validator_delayed` function, a new condition should be added as follows:

```
def is_validator_delayed(validator: LidoValidator) -> bool:
    return (
        validator_requested_to_exit(validator) and
        not is_on_exit(validator) and
        not validator_recently_requested_to_exit(validator) and
        validator_eligible_to_exit(validator) # <- new condition
    )
```

References

1. SCSVS G4: Business logic

[LDO-0dbc1d8a-L02] Loss of rewards in case of omitted frame

LOW ACKNOWLEDGED

Retest (2025-03-17)

The team's comment: With the upcoming consensus version change, we anticipate new behavior from the Oracles. To prevent unexpected issues—such as a report being collected under the old Oracle version but submitted under the new consensus—it has been decided that it is safer to refrain from submitting reports altogether in such cases. We address this by scheduling the voting process in a way that ensures the contract increment and consensus version update occur between the finalized report and the next reference slot. This approach mitigates risks associated with mixed versions during the reporting process.

Affected files

- consensus.py#L237-L260

Description

The `_check_compatibility` function plays a critical role in ensuring that the contract and consensus versions in the contract (on-chain) align during report generation.

Specifically, it checks that the versions match at two key moments:

1. at the time the report is generated ('latest' block),
2. at the end of the frame ('block_hash' block).

If there is a discrepancy between these versions, the report for the current frame is not processed, and later, the Oracle generates a report for the time range covering two frames. In such cases, if any Node Operator is penalized for misbehavior in the new frame, it will inadvertently affect their eligibility for rewards from the previous frame, potentially resulting in a loss of those rewards.

Vulnerable scenario

The following sequence illustrates how this issue can occur:

① The current frame's `ref_slot` is at block number 1000, allowing the Oracle to build and process a report at block 1010 (as an example) once block 1000 is finalized.
② The consensus version is updated in block 1005.
③ When the Oracle attempts to build the report in block 1010, the `_check_compatibility` function triggers a `ContractVersionMismatch` exception.
④ Later, in block 1100, a Node Operator becomes ineligible for rewards in the next frame (e.g., their validator is slashed).
⑤ The Oracle then processes a report for the next frame (e.g. `ref_slot` at 2000), during which the Node Operator's validator is excluded due to the slashing event.

⑥ As a result, the potential rewards earned from the first frame are canceled and redistributed among other Node Operators.

Result: The Node Operator fails to receive rewards from the initial frame, even though penalties should only apply to the subsequent frame.

Recommendation

Either process reports for each frame independently or ensure that the contract versions remain unchanged during the report generation and processing periods.

References

1. SCSVS G1: Architecture, design and threat modeling

[LDO-0dbc1d8a-L03] Lack of rounding epoch to frame in rebase border epoch calculation

LOW ACKNOWLEDGED

Retest (2025-03-17)

The team's comment: The described case can occur if the shift value is set arbitrarily. In the mainnet setup, the value of `FINALIZATION_MAX_NEGATIVE_REBASE_EPOCH_SHIFT` is configured according to the specification as a multiple of the oracle report frame size. The code does not round the value down to the start of the frame, as the dev team considers additional logic unnecessary. The specification will be clarified to state that the value should be aligned with the frame size.

Affected files

- safe_border.py#L99

Description

The `_get_negative_rebase_border_epoch` function determines the earliest epoch that can be allowed when bunker mode is activated following a negative rebase. The documentation specifies that the duration of withdrawal request finalization in case of negative rebase exceed `MAX_NEGATIVE_REBASE_BORDER`, which is defined in the code as `FINALIZATION_MAX_NEGATIVE_REBASE_EPOCH_SHIFT` slots. Currently, this value is set to `0x0546`, equivalent to 6 days.

As illustrated in the accompanying picture taken from the documentation, a withdrawal re-

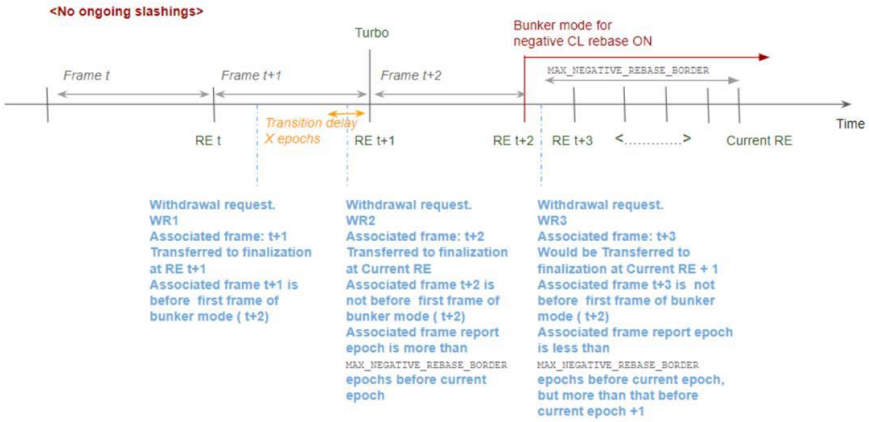
quest (WR3) submitted immediately after bunker mode activation should be accessible in the `Current RE + 1` frame. This is despite the fact that the `MAX_NEGATIVE_REBASE_BORDER` time frame has elapsed in the `Current RE` epoch. Thus, the negative rebase border epoch should be adjusted to align with the first epoch within the frame.

2. Penalties for incidents relevant to the request are socialized for the "bunker mode" due to a negative CL rebase

Formalization for this condition:

The withdrawal request's associated frame is before the first frame of the "bunker mode" due to a negative CL rebase

or associated frame report epoch is equal or more than `MAX_NEGATIVE_REBASE_BORDER = 1536 # epochs ~6.8 days` before the current report epoch.



However, the current code implementation does not round down the epoch, which leads to the WR3 request being accessible in the `Current RE` epoch instead.

```
earliest_allowable_epoch = self.get_epoch_by_slot(self.blockstamp.ref_slot) -
    max_negative_rebase
```

Result: The withdrawal request is made available one frame earlier than specified in the documentation.

Recommendation

Implement rounding down of the epoch to the first epoch within the frame.

References

1. SCSVS G1: Architecture, design and threat modeling
2. "Bunker mode": what it is and how it works
3. Lido Docs: Request finalization

6. Recommendations

[LDO-0dbc1d8a-R01] Eject validators exitable in upcoming oracle frame

INFO IMPLEMENTED

Retest (2025-03-17)

The issue was addressed in **5380490c3511f20570ce2424454a2eea21226145**

The team's comment: Function was renamed, the intent wasn't to filter exited validators from the Ethereum point of view, but filter validators which Lido considers not exitable.

Description

The `get_filter_non_exitable_validators` function is used to select only exitable validators.

However, it currently only verifies that a validator does not have the `exit_epoch` set and has not been requested to exit. This approach fails to account for scenarios where a validator may not be able to exit in the next frame, such as when the validator has just been activated and has not yet completed the requisite `SHARD_COMMITTEE_PERIOD` epochs.

Recommendation

The function should be revised to incorporate checks for all necessary exit conditions, marking validators as non-exitable if these conditions are not satisfied.

References

1. SCSVS G4: Business logic

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com