# Kiln Security Review


**Auditors**

Optimum, Lead Security Researcher

Saw-Mon and Natalie, Lead Security Researcher

Xiaoming90, Security Researcher

0x4non, Associate Security Researcher

**Report prepared by:** Pablo Misirov



July 27, 2023

# Contents


**1 About Spearbit** **2**

**2 Introduction** **2**

**3 Risk classification** **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

**4 Executive Summary** **3**

**5 Findings** **4**
5.1 High Risk . . . 4
5.1.1 Order of calls to `removeValidators` can affect the resulting validator keys set . . . 4
5.1.2 Third-party operator could bypass `FeeRecipient` contract . . . 4
5.1.3 An operator can hijack another operator's keys and signatures to register its own fee recipient 5
5.1.4 Deposit front-running . . . 6
5.1.5 An operator can reshuffle its keys before an admin submits a limit . . . 7
5.2 Medium Risk . . . 7
5.2.1 When removing validators `OperatorIndex` is not reset . . . 7
5.2.2 `getOperatorFeeRecipient` would return wrong data for unregistered/unused public keys . . . 8
5.3 Low Risk . . . 8
5.3.1 When depositing validators for an operator it would be best to update `totalAvailableValidators` first. . . . 8
5.3.2 In some edge cases the `withdrawer`s might be forced to pay fees even for their deposited `32 ether` when withdrawing . . . 9
5.3.3 If the `limit` for an operator equals to its number of keys after removing a key in the middle of the array the `limit` would end up less than the number of keys . . . 9
5.3.4 `getWithdrawer`, `getWithdrawerFromPublicKeyRoot`, `getExitRequestedFromRoot`, and `getWithdrawnFromPublicKeyRoot` return default values for not enabled public keys . . . 10
5.3.5 Check against `address(0)` are missing . . . 10
5.3.6 `StakingContract.acceptOwnership` is missing a reset of the pending admin address . . . 11
5.3.7 `Treasury.withdraw` - Possible temporary denial of service . . . 11
5.3.8 `Treasury.setBeneficiaries` - Possible duplicates in the `beneficiaries` array . . . 11
5.4 Gas Optimization . . . 12
5.4.1 Replace `pad64` with `abi.encodePacked` . . . 12
5.4.2 Consolidate all view calls to the staking contract from a dispatcher . . . 12
5.4.3 Define a new setter for `ValidatorsFundingInfo`'s `availableKeys` . . . 13
5.4.4 The `depositAmount` is constant; therefore `toLittleEndian64` value can be precalculated an set as a constant. . . . 13
5.4.5 Replace division and mod by powers of 2 by the equivalent bitwise operations . . . 15
5.4.6 Updating `totalAvailableValidators` in `_depositOnTwoOperators` can be simplified . . . 15
5.4.7 The logic in `_depositValidatorsOfOperator`'s for loop can be slightly optimised . . . 16
5.4.8 `toLittleEndian64` function could be simplified and optimize . . . 16
5.4.9 Add more optimization runs and consider adding the flag `viaIR` . . . 17
5.4.10 `ExitRequest` event can be hoisted out of the for loop in `requestValidatorsExit` . . . 18
5.4.11 `removeValidators` can be optimised . . . 19
5.4.12 `setOperatorFee` and `setGlobalFee` can be optimised . . . 20
5.4.13 Use custom errors rather than `revert()`/`require()` strings to save gas . . . 21
5.4.14 `addOperator` can be optimised by caching the newly added operator index. . . . 22
5.5 Informational . . . 23
5.5.1 Inconsistent logic of calling `operator` in fee dispatchers . . . 23
5.5.2 `ValidatorsFundingInfo` storage packing tries to fit 4 instances in one storage slot . . . 23
5.5.3 Decrement storage slot hashes by one . . . 23
5.5.4 Mention the source of the included library. . . . 24

5.5.5 Rename `osi` to `vfi` . . . 24
5.5.6 Deposit workflow can be simplified by removing the `_withdrawer` parameter and just using `msg.sender` when needed . . . 24
5.5.7 Add more documentation for an operator's `_feeRecipientAddress` . . . 25
5.5.8 `cap`'s definition in `_updateAvailableValidatorCount` can be simplified . . . 25
5.5.9 Use named constants . . . 25
5.5.10 The `withdrawer` might be taxed for its deposited `32 ether` if they forget to request exit but the validator exits voluntarily anyway . . . 26
5.5.11 Remove unused custom errors . . . 26
5.5.12 Remove unimplemented endpoints from interfaces. . . . 26
5.5.13 No custom errors thrown when `_indexes[0] >= operator.publicKeys.length` in `removeValidators` . . . 27
5.5.14 Calculation of `pubkeyRoot` can conform the same logic as the rest of the code in `setWithdrawer` 27
5.5.15 `operators` might have/push duplicate entries. . . . 27
5.5.16 Certain parameters can only be modified by an upgrade . . . 28
5.5.17 Rename `onlyOperatorFeeRecipient` . . . 28
5.5.18 Event emission is missing . . . 29
5.5.19 Wrong revert error on ETH transfer failure in `ExecutionLayerFeeDispatcher.sol` . . . 29
5.5.20 `ExecutionLayerFeeDispatcher, ConsensusLayerFeeDispatcher` - the treasury address is missing from the `Withdrawal` event . . . 30

# 1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

# 2 Introduction

Kiln is a staking platform you can use to stake directly, or whitelabel staking into your product. It enables users to stake crypto assets, manually or programmatically, while maintaining custody of your funds in your existing solution, such Fireblocks, Copper, or Ledger.

*Disclaimer*: This security review does not guarantee against a hack. It is a snapshot in time of staking-contracts according to the specific commit. Any modifications to the code will require a new security review.

# 3 Risk classification

| Severity level | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| **Likelihood: high** | Critical | High | Medium |
| **Likelihood: medium** | High | Medium | Low |
| **Likelihood: low** | Medium | Low | Low |

## 3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

## 3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

## 3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

# 4 Executive Summary

Over the course of 20 days in total, Kiln engaged with Spearbit to review the staking-contracts protocol. In this period of time a total of **49** issues were found.

**Note: This is the initial report corresponding to the first part of the security review targeting only the staking-contracts.**

**Summary**

| | |
|---|---|
| **Project Name** | Kiln |
| **Repository** | staking-contracts |
| **Commit** | c64e11...b6a3 |
| **Type of Project** | Enterprise Staking, Liquid Staking |
| **Full Review Timeline** | April 17 - May 12 |

**Post review hash**

| | |
|---|---|
| **staking-contracts** | 004e5f...b0bc |

**Issues Found**

| Severity | Count | Fixed | Acknowledged |
|---|---|---|---|
| Critical Risk | 0 | 0 | 0 |
| High Risk | 5 | 1 | 4 |
| Medium Risk | 2 | 2 | 0 |
| Low Risk | 8 | 7 | 1 |
| Gas Optimizations | 14 | 8 | 6 |
| Informational | 20 | 15 | 5 |
| **Total** | **49** | **33** | **16** |

# 5 Findings

## 5.1 High Risk

### 5.1.1 Order of calls to `removeValidators` can affect the resulting validator keys set

**Severity:** High Risk

**Context:**

- StakingContract.sol#L545

**Description:** If two entities *A* and *B* (which can be either the `admin` or the active operator *O* with the index *I*) send a call to `removeValidators` with 2 different sets of parameters:

- $T_1 : (I, R_1)$
- $T_2 : (I, R_2)$

Then, depending on the order of transactions, the resulting set of validators for this operator might be different. Since neither party is able to know a priori if any other transaction is going to be included on the blockchain after they submit their transaction, they don't have a 100 percent guarantee that their intended set of validator keys is going to be removed.

This also opens an opportunity for either party to DoS the other party's transaction by frontrunning it with a call to remove enough validator keys to trigger the out-of-bound error:

```
operators.value[_operatorIndex].publicKeys[_indexes[i]]
```

**Recommendation:** We can send a snapshot block parameter to `removeValidators` and compare it to a stored field for the operator and make sure there have not been any changes to the validator key set since that snapshot block.

**Kiln:** Acknowledged, as

- There is off-chain monitoring to verify that the system is working properly (especially the validator list).
- In a production setup, there is a legal agreement with the operators that enforces them to run the service in the best interest of the system and the users.
- In practice, removeValidators is used rarely.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

### 5.1.2 Third-party operator could bypass `FeeRecipient` contract

**Severity:** High Risk

**Context:**

- StakingContract.sol

**Description:** Kiln operates all validators at this point in time. However, it was understood that there might be whitelisted entities besides Kiln becoming an operator and providing validators to be funded to Kiln's on-chain products in the future.

With the current approach, each third-party operator will have their own dedicated `StakingContract` contract deployed by Kiln.

These third-party operators would register the public key of their managed validators to be funded via the `addValidators` function of their dedicated `StakingContract` contract. Then, Kiln would "approve" the validators via the `setOperatorLimit` function. These approved validators would then be funded with a deposit from its users.

The third-party operator will be required to configure the fee recipient of their validators to point to the address of Kiln's deployed `FeeRecipient` contract which manages execution layer (EL) rewards (Priority fee and MEV

rewards). When the rewards are withdrawn by the stakers, the `FeeRecipient` contract will call the `ExecutionLayerFeeDispatcher` contract which sends a portion of the withdrawal assets to the operator and treasury as a fee.

However, it is possible for a validator to change its EL fee recipient at any point.

```
POST /eth/v1/validator/register_validator

[
  {
    "message": {
      "fee_recipient": "0xabcf8e0d4e9587369b2301d0790347320302cc09",
      "gas_limit": "1",
      "timestamp": "1",
      "pubkey": "0x93247f2209abcacf57b75a51dafae777f9dd38bc7053d1af526f220a7489a6d3a2753e5f3e8b1cfe39b5
      ↪  6f43611df74a"
    },
    "signature": "0x1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af888da41af171505cc411d61252fb6cb3fa
    ↪  0017b679f8bb2305b26a285fa2737f175668d0dff91cc1b66ac1fb663c9bc59509846d6ec05345bd908eda73e670af8
    ↪  88da41af171505"
  }
]
```

A malicious operator could change the EL fee recipient to point to an address controlled by them and bypass the `FeeRecipient` contract entirely. As a result, the staker and treasury will not receive any EL rewards.

**Recommendation:** Consider having some off-chain agreement or taking some on-chain collateral to disincentivize these behaviors. Off-chain monitoring can be set up to monitor the fee recipient change activities on the consensus layer.

**Kiln:** Acknowledged.

- There is off-chain monitoring in place to ensure Kiln is alerted if a fee recipient is not set properly.
- There is no way to enforce this at the Ethereum Execution Layer level.
- In a production setup, there is a legal agreement with the operators that enforces them to run the service in the best interest of the system and the users.

**Spearbit:** Acknowledged.

#### 5.1.3 An operator can hijack another operator's keys and signatures to register its own fee recipient

**Severity:** High Risk

**Context:**

- StakingContract.sol#L488-L493

**Description:** An operator can hijack another operator's public keys and signatures and call the `addValidators` endpoint. Then those keys would be registered for the hijacker and its fee recipients would receive the EL/CL operator fees since they would call into:

- `getOperatorFeeRecipient`

In the dispatcher contracts.

**Recommendation:** One can also provide an aggregated BLS signature to `addValidators` that would sign the operator recipient address and possibly the operator as well and then the protocol will use this signature along with the provided BLS public keys to verify the data.

All operators can also monitor the activities and make sure their public keys and deposit contract signatures are not used by other operators first. If they're provided data to `addValidators` reverts, they would need to generate another set of keys and signatures. A malicious operator can also DoS other operators, but they can be removed or deactivated by the admin.

**Kiln:** Acknowledged.

- There is off-chain monitoring to verify that Kiln validators are not used elsewhere.
- BLS manipulation is avoided as there is no precompile available for now.
- In a production setup, there is a legal agreement with the operators that enforces them to run the service in the best interest of the system and the users.

**Spearbit:** Acknowledged.

#### 5.1.4 Deposit front-running

**Severity:** High Risk

**Context:**

- StakingContract.sol#L232-L239

**Description:** Kiln operates all validators at this point in time. However, it was understood that there might be whitelisted entities besides Kiln becoming an operator and providing validators to be funded to Kiln's on-chain products in the future.

With their current approach, each third-party operator will have their own dedicated `StakingContract` contract deployed by Kiln.

These third-party operators would register the public key of their managed validators to be funded via the `addValidators` function of their dedicated `StakingContract` contract. The Kiln would "approve" the validators via the `setOperatorLimit` function.

As per the Ethereum consensus layer specification, the validator public key is associated with the withdrawal credentials (WC) on the first valid deposit that uses the public key. Subsequent deposits will use the WC from the first deposit even if another WC is specified.

Assume that Kiln whitelists a third-party operator called *O*, and its validators have been "approved" by Kiln. When users deposit ETH to *O*'s `StakingContract` contract, *O* could front-run them, deposit 1 ETH, and set the WC to *O*'s controlled address. When the user deposit transaction gets executed on-chain, the WC pointing to Kiln's `FeeRecipient` contract is ignored.

`1 ether` is the minimum fronrunning attack amount as per `DepositContract`

```
// Check deposit amount
require(msg.value >= 1 ether, "DepositContract: deposit value too low");
require(msg.value % 1 gwei == 0, "DepositContract: deposit value not multiple of gwei");
uint deposit_amount = msg.value / 1 gwei;
require(deposit_amount <= type(uint64).max, "DepositContract: deposit value too high");
```

*O*'s validator will manage 1 ETH of XYZ's funds and 32 ETH of users' funds, fully controlled and withdrawable by *O* since *O* controls the WC.

This vulnerability has also historically affected other liquid staking protocols (Lido, RocketRool, Tranchess).

**Recommendation:** Consider having some off-chain agreement or taking some on-chain collateral to disincentivize these behaviors. Off-chain monitoring can be set up to monitor deposits to the deposit contract on the consensus layer.

**Kiln:** Acknowledged.

- There is off-chain monitoring to verify that Kiln validators are not used elsewhere / have already been deposited.
- In a production setup, there is a legal agreement with the operators that enforces them to run the service in the best interest of the system and the users.

**Spearbit:** Acknowledged.

#### 5.1.5 An operator can reshuffle its keys before an admin submits a limit

**Severity:** High Risk

**Context:**

- StakingContract.sol#L423-L435

**Description:** An active operator can call `removeValidators` and `addValidators` beforehand to reshuffle the validator keys in storage before the admin call to the `setOperatorLimit` endpoint and potentially allow certain keys that have not been verified to be included below the `limit`.

**Recommendation:** It would be best to introduce a mechanism where the admin also passes a snapshot timestamp and also keep track of when different storage bounds have been modified for an operator (`funded`, `limit`, public key lengths, ...) and save that timestamp as the last edit timestamp for the operator in storage to compare with the snapshot timestamp provided to `setOperatorLimit`. If the provided snapshot timestamp is stale, it would revert if we are increasing the limit.

**Kiln:** Fixed in

- commit `7c1041`.
- commit `8edea6`.
- Main fix PR.

**Spearbit:** Fixed.

### 5.2 Medium Risk

#### 5.2.1 When removing validators `OperatorIndex` is not reset

**Severity:** Medium Risk

**Context:**

- StakingContract.sol#L545
- StakingContract.sol#L519-L521
- StakingContract.sol#L233

**Description:** When removing validators, the `OperatorIndex` structs are not cleaned. So a removed validator cannot be added back due to the following check:

```
if (operatorIndexPerValidator.value[pubKeyRoot].enabled) {
    revert DuplicateValidatorKey(publicKey);
}
```

that also means `getOperatorFeeRecipient` would return data that might not be relevant.

**Recommendation:** If re-adding a validator key or making sure `getOperatorFeeRecipient` not returning stale data is important, should also clean `OperatorIndex` struct for the operator in question.

**Kiln:** Fixed in by cleaning the old data

- commit `082356`
- commit `5d1466`

**Spearbit:** Fixed.

#### 5.2.2 `getOperatorFeeRecipient` would return wrong data for unregistered/unused public keys

**Severity:** Medium Risk

**Context:**

- StakingContract.sol#L232-L239

**Description:** If `pubKeyRoot` is not registered (does not belong to any added validator, `getOperatorFeeRecipient(bytes32 pubKeyRoot)` would still return the `feeRecipient` for the `0`th operator index. This can cause on and off-chain miscalculations.

Also, it can OOB revert if no operators are currently present.

**Recommendation:** We can use `operatorIndexPerValidator.value[pubKeyRoot].enabled` to signal that the validator has not been added and potentially revert or maybe return a value signifying that this `pubKeyRoot` is not enabled yet.

Also, one possibly should be able to query `operatorIndexPerValidator.value[pubKeyRoot].enabled` from an external view endpoint.

**Kiln:** Fixed by reverting if `operatorIndexPerValidator.value[pubKeyRoot].enabled` is `false`

- commit `23c726`.

**Spearbit:** Fixed.

### 5.3 Low Risk

#### 5.3.1 When depositing validators for an operator it would be best to update `totalAvailableValidators` first.

**Severity:** Low Risk

**Context:**

- StakingContract.sol#L852
- StakingContract.sol#L891-L893
- StakingContract.sol#L1035

**Description:** In `_depositOnOneOperator`, `_depositOnTwoOperators`, and `_depositOnThreeOrMoreOperators` all calculations and calls are performed before updating `totalAvailableValidators`.

```
StakingContractStorageLib.setTotalAvailableValidators(_totalAvailableValidators - _depositCount);
```

**Recommendation:** It would be best to update `totalAvailableValidators` first and then perform the rest of the logic.

**Kiln:** Fixed in:

- commit `c53823`.
- PR 84.

**Spearbit:** Fixed.

### 5.3.2 In some edge cases the `withdrawer`s might be forced to pay fees even for their deposited `32 ether` when withdrawing

**Severity:** Low Risk

**Context:**

- ConsensusLayerFeeDispatcher.sol#L73-L80

**Description:** The stacking contract admin can force a withdrawer to pay fees without the `32 ether` exemption in the case that the validator is slightly slashed before it is requested to exit but it had skimmed enough CL funds to the fee recipient contract that if the skimmed funds and the slightly below `32 ether` amount exited are summed up it goes beyond `32 ether` and the withdrawer can have their `32 ether` back without paying fees by calling withdraw once. But the admin can call withdraw twice, once before the fee recipient has received the exited amount and once after and thus forcing the withdrawer to pay fees for the whole amount.

**Recommendation:** It would be best to document this edge case and also define when the admin would call the different withdraw endpoints on behalf of a withdrawer.

**Kiln:** This edge case is partially covered in

- commit `4e178a`.

If the slashed amount is more than `1 ether`, the user will/can be taxed for all the received fees.

**Spearbit:** Fixed.

### 5.3.3 If the `limit` for an operator equals to its number of keys after removing a key in the middle of the array the `limit` would end up less than the number of keys

**Severity:** Low Risk

**Context:**

- StakingContract.sol#L545

**Description:** If all public keys and signatures were approved by setting a `limit` so that it would equal their length after we remove a validator with an index below the `limit` it will cause some of the validators with higher key index to fall above the `limit` so they won't be able to be used as the admin would need to set the `limit` again even for all these previously approved validators.

**Recommendation:** The logic in `removeValidators` can be modified to make sure the invariant `limit == keys.length` stays the same before and after calling this endpoint.

**Kiln:** Acknowledged.

- In this case, a SYS_ADMIN call to setOperatorLimit would fix the issue, which is not an operational problem to perform.
- In practice, removeValidators is used rarely.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.3.4 `getWithdrawer`, `getWithdrawerFromPublicKeyRoot`, `getExitRequestedFromRoot`, and `getWithdrawnFromPublicKeyRoot` return default values for not enabled public keys

**Severity:** Low Risk

**Context:**

- StakingContract.sol#L241-L251
- StakingContract.sol#L253-L263

**Description:** If a public key root is not enabled in the system yet, then both `getWithdrawer` and `getWithdrawerFromPublicKeyRoot` return `address(0)`. The caller should be able to make sense of this returned value.

The same issue and recommendation apply to `getExitRequestedFromRoot` and `getWithdrawnFromPublicKeyRoot`.

**Recommendation:** It would be best to expose `operatorIndexPerValidator.value[pubKeyRoot].enabled` as a viewable query so that the caller can deduce the returned `address(0)` values is due to the public key not being enabled in the system yet.

Also, the above scenario should be documented for both of these functions in this context.

Related: *`getOperatorFeeRecipient` would return wrong data for unregistered/unused public keys*

**Kiln:** Comments have been added and also `operatorIndexPerValidator.value[pubKeyRoot].enabled` has been exposed with a new endpoint:

- commit `171c6e`.

**Spearbit:** Fixed.

#### 5.3.5 Check against `address(0)` are missing

**Severity:** Low Risk

**Context:**

- StakingContract.sol#L144-L149
- StakingContract.sol#L386-L387
- StakingContract.sol#L400

**Description:** The addresses in this context are not checked to make sure they are not `address(0)`.

In the context of `setOperatorAddresses` if for example, one would have set the `_feeRecipientAddress` to `address(0)`, then one could not update the operator and the fee recipient addresses anymore unless the team would push an upgrade to the logic.

**Recommendation:** It might be best to add a check against `address(0)` or make these address changes a 2-step process if necessary.

**Kiln:** Fixed in

- commit `15743a`.
- commit `42b1d5`.

**Spearbit:** Fixed.

#### 5.3.6 `StakingContract.acceptOwnership` is missing a reset of the pending admin address

**Severity:** Low Risk

**Context:**

- StakingContract.sol#L345

**Description:** The `StakingContract` contract implements a two-step process for changing the ownership of a contract. However, it's worth noting that in the current implementation, the pending admin is not set to zero as part of the `acceptOwnership` function. As a result, the `getPendingAdmin` function may return a stale value, and the fact that `acceptOwnership` can be called more than once by a pending admin could potentially cause issues for off-chain services that rely on the `ChangedAdmin` event.

**Recommendation:** Consider setting the pending admin address to zero as part of `acceptOwnership`.

**Kiln:** Fixed in commit `fce981` by implementing the auditor's recommendation.

**Spearbit:** Fixed.

#### 5.3.7 `Treasury.withdraw` - Possible temporary denial of service

**Severity:** Low Risk

**Context:** Treasury.sol#L88-L91

**Description:** The `Treasury.withdraw` function distributes the fees that have accumulated in the `Treasury` contract to a group of pre-determined beneficiaries. It's important to keep in mind that if any of these beneficiaries encounter an error during the distribution process, the entire transaction will be reverted. If this occurs, the accrued funds cannot be released from the contract until the `Treasury` admin removes the problematic beneficiary by invoking the `setBeneficiaries` function.

**Recommendation:** Consider writing a script to monitor failed `withdraw` transactions, which would help the admin identify the problematic beneficiary and shorten the length of any temporary denial of service. In addition, consider adding the beneficiary that caused the failure to the revert error message.

**Kiln:** Fixed in commit `c64a51` by removing the `Treasury` contract.

**Spearbit:** Fixed.

#### 5.3.8 `Treasury.setBeneficiaries` - Possible duplicates in the `beneficiaries` array

**Severity:** Low Risk

**Context:** Treasury.sol#L63-L76

**Description:** The current code will not revert for duplicates inside the `beneficiaries` which may end up in a higher accumulated percentage for a given address in case the admin of the contract does not validate it off-chain. **Recommendation:** Consider adding a check to validate that the `beneficiaries` array is sorted (that each consecutive element is either greater than the other for instance), it will be an efficient way to make sure there are no duplications.

**Kiln:** Fixed in commit `c64a51` by removing the `Treasury` contract.

**Spearbit:** Fixed.

## 5.4 Gas Optimization

### 5.4.1 Replace `pad64` with `abi.encodePacked`

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L819
- StakingContract.sol#L404
- StakingContract.sol#L727

**Description:** Using the native `abi.encodePacked` just like Ethereum 2.0 deposit contract instead of `pad64` would save a considerable amount of gas.

**Recommendation:** Consider using

```
abi.encodePacked(_b, bytes16(0)) // or
abi.encodePacked(_b, bytes32(0))
```

wherever `pad64` is used.

Since these are the only 3 places where `BytesLib.pad64` has been used, we can remove this helper function from the codebase unless the team is planning to use it elsewhere.

**Kiln:** Fixed in commit `22cb2a` .

**Spearbit:** Fixed.

### 5.4.2 Consolidate all view calls to the staking contract from a dispatcher

**Severity:** Gas Optimization

**Context:**

- ConsensusLayerFeeDispatcher.sol#L60
- ExecutionLayerFeeDispatcher.sol#L59

**Description:** The `ExecutionLayerFeeDispatcher` and `ConsensusLayerFeeDispatcher` have a `dispatch` functions that makes multiple view calls into the stacking contract.

**Recommendation:** The view calls to `stakingContract` can be consolidated into one to avoid calling this external contract multiple times:

- `getExitRequestedFromRoot` (only for CL dispatcher)
- `getWithdrawnFromPublicKeyRoot` (only for CL dispatcher)
- `getGlobalFee()`
- `getOperatorFee()`
- `getWithdrawerFromPublicKeyRoot`
- `getOperatorFeeRecipient` (only needed if `operatorFee > 0`)
- `getTreasury` (only needed if `globalFee > 0`)

The endpoint can be called `getDispatchingDataFromPublicKeyRoot(_publicKeyRoot)`

**Kiln:** We want to keep the changes on EL Recipient minimal. Also, it is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

### 5.4.3 Define a new setter for `ValidatorsFundingInfo`'s `availableKeys`

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L751-L760

**Description:** Might be cheaper to define a storage setter that only sets the available count, as in these 2 cases only `availableCount` has been changed

```
if (cap <= validatorFundingInfo.funded) {
    StakingContractStorageLib.setValidatorsFundingInfo(_operatorIndex, 0, validatorFundingInfo.funded);
    ↪  // here
} else {
    newAvailableCount = uint32(cap - validatorFundingInfo.funded);
    StakingContractStorageLib.setValidatorsFundingInfo(
        _operatorIndex,
        newAvailableCount, // here
        validatorFundingInfo.funded
    );
}
```

**Recommendation:** Define a new setter for `ValidatorsFundingInfo`'s `availableKeys`.

**Kiln:** This is a minor optimization we don't consider necessary. Also, It is in our best interest to keep the changes on the On-Chain v1 codebase minimal

**Spearbit:** Acknowledged.

### 5.4.4 The `depositAmount` is constant; therefore `toLittleEndian64` value can be precalculated an set as a constant.

**Severity:** Gas Optimization

**Context:** StakingContract.sol#L828

**Description:** The `depositAmount` is constant; therefore, the value of `toLittleEndian64(depositAmount)` can be precalculated and set as a constant. This change would allow for the removal of the libs/UintLib.sol, as it is only used in these lines of the `StakingContract`.

**Recommendation:**

```
diff --git a/src/contracts/StakingContract.sol b/src/contracts/StakingContract.sol
index 6de34de..2d4087a 100644
--- a/src/contracts/StakingContract.sol
+++ b/src/contracts/StakingContract.sol
@@ -1,7 +1,6 @@
 //SPDX-License-Identifier: BUSL-1.1
 pragma solidity >=0.8.10;

-import "./libs/UintLib.sol";
 import "./libs/BytesLib.sol";
 import "./interfaces/IFeeRecipient.sol";
 import "./interfaces/IDepositContract.sol";
@@ -19,6 +18,8 @@ contract StakingContract {
     uint256 public constant SIGNATURE_LENGTH = 96;
     uint256 public constant PUBLIC_KEY_LENGTH = 48;
     uint256 public constant DEPOSIT_SIZE = 32 ether;
+    // this is the equivalent of Uint256Lib.toLittleEndian64(DEPOSIT_SIZE / 1000000000 wei);
+    uint256 constant DEPOSIT_SIZE_AMOUNT_LITTLEENDIAN64 =
↪  0x0040597307000000000000000000000000000000000000000000000000000000;
     uint256 internal constant BASIS_POINTS = 10_000;

     error Forbidden();
@@ -820,12 +821,10 @@ contract StakingContract {
             )
         );

-        uint256 depositAmount = DEPOSIT_SIZE / 1000000000 wei;
-
         bytes32 depositDataRoot = sha256(
             abi.encodePacked(
                 sha256(abi.encodePacked(pubkeyRoot, _withdrawalCredentials)),
-                sha256(abi.encodePacked(Uint256Lib.toLittleEndian64(depositAmount), signatureRoot))
+                sha256(abi.encodePacked(DEPOSIT_SIZE_AMOUNT_LITTLEENDIAN64, signatureRoot))
             )
         );
```

Gas benchmark methodology, take a snapshot;

```
> forge snapshot --snap base-snap --fuzz-seed=123
```

Compare against snapshot;

```
> forge snapshot --diff base-snap --mt testExplicitDeposit --fuzz-seed=123
....
....
Overall gas change: -295431 (-1.338%)

> forge snapshot --diff base-snap --mt testImplicitDeposit --fuzz-seed=123
....
....
Overall gas change: -206397 (-1.335%)
```

**Kiln:** Fixed by removal of `toLittleEndian64` and introducing a new named constant in

- commit `bc722b`.
- commit `a2127f`.

**Spearbit:** Fixed.

### 5.4.5 Replace division and mod by powers of 2 by the equivalent bitwise operations

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L869-L875

**Description:** In the context above either one is finding the remainder mod a power of `2` or dividing a number by power of `2`.

**Recommendation:** Although the compiler might perform the simplifications below(1, 2), it might be cheaper to enforce these in the code level:

```
// TWO_TO_POWER_N = 2 ** N

A % TWO_TO_POWER_N
A & (TWO_TO_POWER_N - 1) // optimised

A / TWO_TO_POWER_N
A >> N // optimised

A * TWO_TO_POWER_N
A << N // optimised
```

**Kiln:** Multi-operator feature has been removed, so this issue is not relevant anymore

- commit `11ccbc`

**Spearbit:** Fixed.

### 5.4.6 Updating `totalAvailableValidators` in `_depositOnTwoOperators` can be simplified

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L891-L893

**Description:** Updating `totalAvailableValidators` in `_depositOnTwoOperators` can be simplified. This is due to the way `_depositCount` is distributed between `oneDepositCount` and `twoDepositCount`. It is always guaranteed that they sum up to `_depositCount`.

**Recommendation:** The line in the context can be simplified to

```
StakingContractStorageLib.setTotalAvailableValidators(_totalAvailableValidators - _depositCount);
```

**Kiln:** Multi-operator feature has been removed, so this issue is not relevant anymore

- commit `11ccbc`.

**Spearbit:** Fixed.

### 5.4.7 The logic in `_depositValidatorsOfOperator`'s for loop can be slightly optimised

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L790-L791
- StakingContract.sol#L815

**Description:** In `_depositValidatorsOfOperator`'s for loop we call '_depositValidator' and then calculate the public key root:

```
_depositValidator(publicKey, signature, withdrawalCredentials);
bytes32 pubkeyRoot = _getPubKeyRoot(publicKey);
```

But in `_depositValidator` we calculate the public key root again.

**Recommendation:** It might be best to swap those two lines and pass the public key root along with the other parameters to avoid calculating this value twice:

```
bytes32 pubkeyRoot = _getPubKeyRoot(publicKey);
_depositValidator(publicKey, pubkeyRoot, signature, withdrawalCredentials);
```

**Kiln:** Fixed in commit `c042e1`.

**Spearbit:** Fixed.

### 5.4.8 `toLittleEndian64` function could be simplified and optimize

**Severity:** Gas Optimization

**Context:** libs/UintLib.sol#L5-L16

**Description:** The `toLittleEndian64` function could be simplified, you could use the proposed method used in the official DepositContract implementation.

**Recommendation:**

```
diff --git a/src/contracts/libs/UintLib.sol b/src/contracts/libs/UintLib.sol
index 4ff1d1f..e020c65 100644
--- a/src/contracts/libs/UintLib.sol
+++ b/src/contracts/libs/UintLib.sol
@@ -2,15 +2,26 @@
 pragma solidity >=0.8.10;

 library Uint256Lib {
-    function toLittleEndian64(uint256 _value) internal pure returns (uint256 result) {
-        result = 0;
-        uint256 temp_value = _value;
-        for (uint256 i = 0; i < 8; ++i) {
-            result = (result << 8) | (temp_value & 0xFF);
-            temp_value >>= 8;
-        }
+    error errValueTooBig();

-        assert(0 == temp_value); // fully converted
-        result <<= (24 * 8);
+    /// @dev Converts a uint256 to a little endian bytes8. Based on the implementation of the official
↪  DepositContract
+    function toLittleEndian64(uint256 _value) internal pure returns (uint256) {
+        bytes memory ret;
+        if (_value > type(uint64).max) {
+            revert errValueTooBig();
+        }
```

```
+        uint64 value = uint64(_value);
+        ret = new bytes(8);
+        bytes8 bytesValue = bytes8(value);
+        // Byteswapping during copying to bytes.
+        ret[0] = bytesValue[7];
+        ret[1] = bytesValue[6];
+        ret[2] = bytesValue[5];
+        ret[3] = bytesValue[4];
+        ret[4] = bytesValue[3];
+        ret[5] = bytesValue[2];
+        ret[6] = bytesValue[1];
+        ret[7] = bytesValue[0];
+        return uint256(bytes32(ret));
     }
 }
```

POC, first, create the base snap

```
forge snapshot --snap base-snap --fuzz-seed=123
```

Then add the proposed change to the function and re-test it. Overall gas change: -6266647 (-0.177%)

```
forge snapshot --diff base-snap --fuzz-seed=123
......
......
testImplicitDepositNotEnoughAfterFilled(uint256) (gas: -1837 (-0.201%))
testExplicitDepositAllValidators(uint256) (gas: -3571 (-0.208%))
testImplicitDepositAllValidators(uint256) (gas: -3571 (-0.208%))
testExplicitDepositNotEnoughAfterFilled(uint256) (gas: -3571 (-0.210%))
testImplicitDepositNotEnoughAfterFilled(uint256) (gas: -3571 (-0.210%))
testSetOperatorLimitDeactivated(uint256,uint8) (gas: -165853 (-1.443%))
testDistribution(uint8,uint8) (gas: -1539126 (-11.093%))
testSetOperatorLimit(uint256,uint8) (gas: -4475184 (-28.313%))
Overall gas change: -6266647 (-0.177%)
```

**Kiln:** Fixed by removing `toLittleEndian64` in

- commit `a2127f`

**Spearbit:** Fixed.

#### 5.4.9 Add more optimization runs and consider adding the flag `viaIR`

**Severity:** Gas Optimization

**Context:** hardhat.config.ts#L14-L15

**Description:** Adding more optimization runs outputs more gas-efficient code. Consider also compiling using intermediate representation.

Also, review this video: *Solidity Summit 2022 - 17 The Solidity Optimizoooor* by Hari Mulackal.

There is an issue with the tests while compiling with the flag `via-ir`, It might be worth rewriting the test to be able to run a full test with the flag `--via-ir` activated.

> The Solidity compiler uses two different optimizer modules: The “old” optimizer that operates at the opcode level and the “new” optimizer that operates on Yul IR code. The opcode-based optimizer applies a set of simplification rules to opcodes. It also combines equal code sets and removes unused code.

POC, First create the base snap:

```
forge snapshot --snap base-snap --fuzz-seed=123
```

Then, run it with more optimization runs. Overall gas gains: (-0.119%)

```
forge snapshot --diff base-snap --fuzz-seed=123 --optimizer-runs=20000
......
......
testReinitialization() (gas: -315 (-1.535%))
testTransferFunds() (gas: -275 (-1.681%))
testGetWithdrawer() (gas: -240 (-1.868%))
testGetWithdrawer() (gas: -330 (-2.552%))
testDistribution(uint8,uint8) (gas: -1067655 (-7.695%))
testSetOperatorLimitDeactivated(uint256,uint8) (gas: 2592296 (22.562%))
testSetOperatorLimit(uint256,uint8) (gas: -4906866 (-31.044%))
Overall gas change: -4201521 (-0.119%)
```

**Recommendation:** Note that `viaIR: true` is not yet added because tests don't work with it, but its worth digging into it.

The number of runs specifies roughly how often each opcode of the deployed code will be executed across the lifetime of the contract. So there is a tradeoff where a contract optimized for more (cheaper) runs is more expensive to deploy.

```
diff --git a/hardhat.config.ts b/hardhat.config.ts
index f064f6e..14a7f00 100644
--- a/hardhat.config.ts
+++ b/hardhat.config.ts
@@ -12,8 +12,9 @@ const hhuc: HardhatUserConfig = {
     settings: {
       optimizer: {
         enabled: true,
-        runs: 200,
-      }
+        runs: 20000,
+      },
     }
   },
   paths: {
```

**Kiln:** `runs` changed to `20,000` in

- commit `da3111`.

**Spearbit:** Fixed.

### 5.4.10 `ExitRequest` event can be hoisted out of the for loop in `requestValidatorsExit`

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L680-L684

**Description:** Only the `msg.sender` can be the `withdrawer` for each public key provided to `requestValidatorsExit`

```
if (msg.sender != withdrawer) {
    revert Unauthorized();
}
```

**Recommendation:** We can hoist the event out of the loop and only emit one event with all the keys:

```
// after the for loop
emit ExitRequest(msg.sender, _publicKeys);
```

The already implemented off-chain agents listening to this event would need to now slice the `_publicKeys` to get all the public keys.

**Kiln:** Acknowledged.

- We prefer having an "atomic" event feed so that it is easier to aggregate on indexers and search through explorers.
- Exit request is expected to be called only once in the validator lifecycle by the staker, so gas optimization on it has a low impact.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.4.11 `removeValidators` can be optimised

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L545

**Description:** `removeValidators` can be optimized.

**Recommendation:** Consider using the optimized version (rough) of this endpoint.

```
function removeValidators(uint256 _operatorIndex, uint256[] calldata _indexes)
    external
    onlyActiveOperatorOrAdmin(_operatorIndex)
{
    uint256 indexesLength = _indexes.length;
    if (indexesLength == 0) {
        revert InvalidArgument();
    }

    StakingContractStorageLib.ValidatorsFundingInfo memory operatorInfo = StakingContractStorageLib
        .getValidatorsFundingInfo(_operatorIndex);
    StakingContractStorageLib.OperatorsSlot storage operators =
    ↪  StakingContractStorageLib.getOperators();
    StakingContractStorageLib.OperatorInfo storage operator = operators.value[_operatorIndex];

    uint256 totalKeys = operator.publicKeys.length;
    uint256 lastIndex = _indexes[indexesLength - 1];

    if (lastIndex < operatorInfo.funded) {
        revert FundedValidatorDeletionAttempt();
    }

    for (uint256 i; i < indexesLength; ) {
        uint256 keyIndex = _indexes[i];

        if (i > 0 && !(keyIndex < _indexes[i - 1])) {
            revert UnsortedIndexes();
        }

        emit ValidatorKeyRemoved(_operatorIndex, operator.publicKeys[keyIndex]);

        unchecked {
            ++i;
        }

        uint256 lastKeyIndex = totalKeys - i;
```

```
        if (keyIndex != lastKeyIndex) {
            // assumes the invariant that `publicKeys` and `signatures` have the same length
            operator.publicKeys[keyIndex] = operator.publicKeys[lastKeyIndex];
            operator.signatures[keyIndex] = operator.signatures[lastKeyIndex];
        }

        operator.publicKeys.pop();
        operator.signatures.pop();
    }

    if (lastIndex < operator.limit) {
        operator.limit = lastIndex;
        emit ChangedOperatorLimit(_operatorIndex, lastIndex);
    }

    _updateAvailableValidatorCount(_operatorIndex);
}
```

Note that we are assuming the invariant that `publicKeys` and `signatures` have the same length

**Kiln:** Acknowledged.

- `removeValidators` is rarely used in production (never used as of now).
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.4.12 `setOperatorFee` and `setGlobalFee` can be optimised

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L465-L466
- StakingContract.sol#L475-L476

**Description:** In `setOperatorFee` and `setGlobalFee`, we both compare the provided new fee with the `BASIS_POINTS` and also the `...CommissionLimit`:

```
if (_fee > BASIS_POINTS || _fee > CommissionLimit) ...
```

But upon initialization of the operator and global commission limit, we have also checked that those values do not exceed `BASIS_POINTS`. So the first conditional statement above is not needed as it would imply the second one.

**Recommendation:** The lines in this context can be simplified to

```
if (_fee > CommissionLimit) ...
```

**Kiln:** Fixed in commit `f8578f`.

**Spearbit:** Fixed.

### 5.4.13 Use custom errors rather than `revert()`/`require()` strings to save gas

**Severity:** Gas Optimization

**Context:**

- libs/BytesLib.sol#L102-L103

**Description:** Custom errors are available from Solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas.

**Recommendation:** Add custom errors instead of `require`

```
diff --git a/src/contracts/libs/BytesLib.sol b/src/contracts/libs/BytesLib.sol
index 4cfa5c0..8e6226d 100644
--- a/src/contracts/libs/BytesLib.sol
+++ b/src/contracts/libs/BytesLib.sol
@@ -2,6 +2,9 @@
 pragma solidity >=0.8.10;

 library BytesLib {
+    error errSliceOverflow();
+    error sliceOutOfBounds();
+
     function pad64(bytes memory _b) internal pure returns (bytes memory) {
         assert(_b.length >= 32 && _b.length <= 64);
         if (64 == _b.length) {
@@ -99,8 +102,13 @@ library BytesLib {
         uint256 _start,
         uint256 _length
     ) internal pure returns (bytes memory) {
-        require(_length + 31 >= _length, "slice_overflow");
-        require(_bytes.length >= _start + _length, "slice_outOfBounds");
+        if(_length + 31 < _length) {
+            revert errSliceOverflow();
+        }
+
+        if (_bytes.length < _start + _length) {
+            revert sliceOutOfBounds();
+        }

         bytes memory tempBytes;
```

Test methodology;

1) Create a `base-snap` with;

```
forge snapshot --snap base-snap --fuzz-seed=123
```

2) Add the recommendations

3) Test against the `base-snap`

```
forge snapshot --diff base-snap --fuzz-seed=123
...
testAddOperatorLimitReached(uint128) (gas: 176 (0.000%))
testDistribution(uint8,uint8) (gas: -659798 (-4.756%))
testSetOperatorLimitDeactivated(uint256,uint8) (gas: 1785527 (15.540%))
testSetOperatorLimit(uint256,uint8) (gas: -3449562 (-21.824%))
Overall gas change: -2323657 (-0.066%)
```

**Kiln:** Acknowledged.

- We expect pad64 to be called with correct parameters (via good UI implementation + tx simulation).
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.4.14 `addOperator` can be optimised by caching the newly added operator index.

**Severity:** Gas Optimization

**Context:**

- StakingContract.sol#L364-L377

**Description:** In `addOperator` `operators.value.length` is read twice from the storage and also we perform an unnecessary subtraction which can be avoided by caching the operator index first and then pushing its info in the array.

**Recommendation:** `addOperator`'s implementation can be changed to:

```
function addOperator(address _operatorAddress, address _feeRecipientAddress) external onlyAdmin returns
↪  (uint256) {
    StakingContractStorageLib.OperatorsSlot storage operators =
    ↪  StakingContractStorageLib.getOperators();
    StakingContractStorageLib.OperatorInfo memory newOperator;

    uint256 operatorIndex = operators.value.length;

    if (operatorIndex == 251) {
        revert MaximumOperatorCountAlreadyReached();
    }
    newOperator.operator = _operatorAddress;
    newOperator.feeRecipient = _feeRecipientAddress;
    operators.value.push(newOperator);
    emit NewOperator(_operatorAddress, _feeRecipientAddress, operatorIndex);
    return operatorIndex;
}
```

**Kiln:** Acknowledged.

- In practice, this is called only once in a mono-operator setup.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

## 5.5 Informational

### 5.5.1 Inconsistent logic of calling `operator` in fee dispatchers

**Severity:** Informational

**Context:**

- ExecutionLayerFeeDispatcher.sol#L83-L88
- ConsensusLayerFeeDispatcher.sol#L103

**Description:** In `ExecutionLayerFeeDispatcher` one checks whether the operator fee is `0` or not before calling the `operator`. But the same check is missing in `ConsensusLayerFeeDispatcher`.

**Recommendation:** It would be best to keep the logic of calling `operator` the same in all dispatchers.

**Kiln:** Fixed in commit `35febe`.

**Spearbit:** Fixed.

### 5.5.2 `ValidatorsFundingInfo` storage packing tries to fit 4 instances in one storage slot

**Severity:** Informational

**Context:**

- StakingContractStorageLib.sol#L136-L179

**Description:** `ValidatorsFundingInfo` storage packing tries to fit 4 instances in one storage slot.

**Recommendation:** It would be best to document that you are trying to fit `4` `ValidatorsFundingInfo` structs in one storage slot. And that `_index` represents an operator index in the `OperatorsSlot` array.

It would also be best to use named constants in these functions.

**Kiln:** Fixed in commit `79ae4b`.

**Spearbit:** Fixed.

### 5.5.3 Decrement storage slot hashes by one

**Severity:** Informational

**Context:**

- StakingContractStorageLib.sol#L4
- ConsensusLayerFeeDispatcher.sol#L31-L34
- ExecutionLayerFeeDispatcher.sol#L30-L33

**Description:** The storage slots/hashes have known pre-images.

```
bytes32 internal constant SLOT = keccak256("path.to.slot");
```

**Recommendation:** It might be best to decrement them by one so that finding the pre-image would be harder. Or at least apply this rule to the new slots. This recommendation applies to the other contracts as well.

**Kiln:** Decrementing the storage slots of new storage vars only for retro compatibility. Fixed in commit `5650e1`.

**Spearbit:** Fixed.

#### 5.5.4 Mention the source of the included library.

**Severity:** Informational

**Context:** libs/BytesLib.sol#L3

**Description:** `libs/BytesLib.sol` is based on GNSPS/BytesLib.sol and should mention that it is based on it on the comments.

**Recommendation:** Add a comment mentioning GNSPS/BytesLib.sol

**Kiln:** Fixed in commit `48f0a3`.

**Spearbit:** Fixed.

#### 5.5.5 Rename `osi` to `vfi`

**Severity:** Informational

**Context:**

- StakingContract.sol#L860-L863
- StakingContract.sol#L914-L915
- StakingContract.sol#L932-L933

**Description:** The codebase has moved to a new storage variable definition called `ValidatorsFundingInfo`, but some variables are still using the old `osi` shorthand.

**Recommendation:** Rename `osi` to `vfi`.

**Kiln:** Fixed in

- commit `a0d5f2`
- PR 84

**Spearbit:** Fixed.

#### 5.5.6 Deposit workflow can be simplified by removing the `_withdrawer` parameter and just using `msg.sender` when needed

**Severity:** Informational

**Context:**

- StakingContract.sol#L774-L778

**Description:** Since the following old function has been removed (the initial withdrawer address is always the `msg.sender`)

```
/// @notice Explicit deposit method
/// @dev A multiple of 32 ETH should be sent
/// @param _withdrawer The withdrawer address
function deposit(address _withdrawer) external payable {
    _deposit(_withdrawer);
}
```

the `address _withdrawer` input for `_depositValidatorsOfOperator` will always be the `msg.sender`.

**Recommendation:** Therefore, we don't need to keep tab of `_withdrawer` all the way from `_deposit(...)` until we end up here. We can remove these parameters from these functions and in this function wherever `_withdrawer` is used, replace it with `msg.sender`.

The above recommendation only applies if there isn't a plan to reintroduce the removed function.

**Kiln:** Fixed in commit `4fe27f`.

**Spearbit:** Fixed.

### 5.5.7 Add more documentation for an operator's `_feeRecipientAddress`

**Severity:** Informational

**Context:**

- StakingContract.sol#L363

**Description:** `addOperator` receives a `_feeRecipientAddress` which actually has more privilege than `_operatorAddress`, since it can change both addresses by calling `setOperatorAddresses`.

**Recommendation:** It would be great to document the above in the comments for `_feeRecipientAddress`.

**Kiln:** Fixed in commit `b38fe0`.

**Spearbit:** Fixed.

### 5.5.8 `cap`'s definition in `_updateAvailableValidatorCount` can be simplified

**Severity:** Informational

**Context:**

- StakingContract.sol#L749
- StakingContract.sol#L433
- StakingContract.sol#L448
- StakingContract.sol#L535
- StakingContract.sol#L590

**Description:** Whenever `_updateAvailableValidatorCount` is called, it is guaranteed that

```
operators.value[_operatorIndex].limit <= operators.value[_operatorIndex].publicKeys.length
```

and so `cap` will always be `operators.value[_operatorIndex].limit`.

**Recommendation:** It would be best to simplify the `cap`'s definition:

```
uint256 cap = operators.value[_operatorIndex].limit;
```

It might also make sense to rename it to `limit`.

**Kiln:** Fixed in commit `29000b`.

**Spearbit:** Fixed.

### 5.5.9 Use named constants

**Severity:** Informational

**Context:**

- StakingContract.sol#L823

**Description:** There are quite a few instances where constants are used in calculations. Specifically:

- `1000000000 wei` can be defined as `ONE_GWEI` equal to `1 gwei`.

**Recommendation:** It would be best to replace these instances with named constants to add more context to these values and potentially consolidate reuses.

**Kiln:** Issue is not relevant anymore as a new named constant for `DEPOSIT_SIZE_AMOUNT_LITTLEENDIAN64` has been used:

- commit adcfe8.

Also a new named constant WITHDRAWAL_CREDENTIAL_PREFIX_01 has been introduced.

**Spearbit:** Fixed.

#### 5.5.10 The withdrawer might be taxed for its deposited 32 ether if they forget to request exit but the validator exits voluntarily anyway

**Severity:** Informational

**Context:**

- ConsensusLayerFeeDispatcher.sol#L73-L80

**Description:** If a withdrawer does not call requestValidatorsExit and the validator corresponding to _publicKeyRoot exits and then the staking contract admin or the withdrawer calls withdrawCLFee, the withdrawer will be taxed on its 32 ether stacked deposit.

**Recommendation:** withdrawer needs to monitor the deposited validators and even if they voluntarily exit, the withdrawer can call requestValidatorsExit after hand to set the right storage parameters so that when it wants to withdraw it can get the original 32 ether deposit back without getting taxed.

**Kiln:** Acknowledged.

- In practice, this should be considered as a rare edge case as Operators have SLAs in their Terms and Conditions with the user.
- In practice if this happens, manual rebates would be performed following the Terms and Conditions guarantees.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.5.11 Remove unused custom errors

**Severity:** Informational

**Context:**

- ConsensusLayerFeeDispatcher.sol#L29

**Description:** Some custom errors are not used in the codebase.

**Recommendation:** If there aren't plans to use these custom errors, it would be best to remove them from the codebase.

**Kiln:** Fixed in commit d92919.

**Spearbit:** Fixed.

#### 5.5.12 Remove unimplemented endpoints from interfaces.

**Severity:** Informational

**Context:**

- IStakingContractFeeDetails.sol#L7

**Description:** Some interface endpoints are not implemented.

**Recommendation:** If there aren't plans to implement these endpoints in the future, it would be best to remove them from the codebase.

**Kiln:** Fixed in commit b0d409.

**Spearbit:** Fixed.

#### 5.5.13 No custom errors thrown when `_indexes[0] >= operator.publicKeys.length` in `removeValidators`

**Severity:** Informational

**Context:**

- StakingContract.sol#L545

**Description:** The `_indexes` array provided to `removeValidators` is supposed to be a decreasing array. We check whether the last index is not a funded index. But we do not check whether the first index is in the allowed range (less than `operator.publicKeys.length`).

**Recommendation:** Currently `removeValidators` would revert an out-of-bound error if `_indexes[0] >= operator.publicKeys.length`. But if desired one can create a custom error for this edge case.

**Kiln:** Acknowledged.

- In practice, removeValidators is rarely used (never used as of now).
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.

**Spearbit:** Acknowledged.

#### 5.5.14 Calculation of `pubkeyRoot` can conform the same logic as the rest of the code in `setWithdrawer`

**Severity:** Informational

**Context:**

- StakingContract.sol#L404

**Description:** In `setWithdrawer`, `pubkeyRoot` is calculated as

```
bytes32 pubkeyRoot = sha256(BytesLib.pad64(_publicKey));
```

**Recommendation:** One can use the internal `_getPubKeyRoot` function like the rest of the codebase for consistency. Also, in the future, if this calculation needs modification, change this internal function:

```
bytes32 pubkeyRoot = _getPubKeyRoot(_publicKey);
```

**Kiln:** Fixed in commit `839930`.

**Spearbit:** Fixed.

#### 5.5.15 `operators` might have/push duplicate entries.

**Severity:** Informational

**Context:**

- StakingContract.sol#L364-L377

**Description:** `operators` might have/push duplicate entries. In `addOperator` when we push a new element in the array we do not check if there are already entries with the same `_operatorAddress` or `_feeRecipientAddress`.

**Recommendation:** This needs to be documented.

**Kiln:** Fixed by removing the multi-operator functionality

- commit `11ccbc`.

**Spearbit:** Fixed.

### 5.5.16 Certain parameters can only be modified by an upgrade

**Severity:** Informational

**Context:**

- StakingContract.sol#L152-L153
- StakingContract.sol#L146-L149

**Description:** With the current logic once `globalCommissionLimit` and `operatorCommissionLimit` have been initialized, the staking contract admin would not be able to change these values.

The same goes for:

- `ELDispatcher`
- `CLDispatcher`
- `DepositContract`
- `FeeRecipientImplementation`, if a change of this implementation is required one would need to keep a tab of the already deployed fee recipient and distinguish them from the to-be-deployed ones due to how the theses addresses are used in the staking contract and also registered in the deposit contract as the CL fee recipient.

**Recommendation:** If changes are required, the proxy admin would need to migrate the logic to one that would allow either reinitialization or an update process.

**Kiln:** Acknowledged. This is intended behavior as we want to cap the commission of each actor in the system.

**Spearbit:** Acknowledged.

### 5.5.17 Rename `onlyOperatorFeeRecipient`

**Severity:** Informational

**Context:**

- StakingContract.sol#L108

**Description:** The `onlyOperatorFeeRecipient` modifier would revert if the operator has been deactivated:

```
if (operatorInfo.deactivated) {
    revert Deactivated();
}
```

**Recommendation:** To better reflect the above it might be best to rename this modifier to `onlyActiveOperatorFeeRecipient`.

**Kiln:** Fixed in commit `51735d`.

**Spearbit:** Fixed.

#### 5.5.18 Event emission is missing

**Severity:** Informational

**Context:**

- StakingContract.sol#L174
- StakingContract.sol#L267
- StakingContract.sol#L339
- StakingContract.sol#L545
- StakingContract.sol#L751-L760
- StakingContract.sol#L763-L765
- Treasury.sol#L63
- Treasury.sol#L24
- Treasury.sol#L81
- UPProxy.sol#L28
- TUPProxy.sol#L33
- FeeRecipient.sol#L24-L25
- ExecutionLayerFeeDispatcher.sol#L55-L56
- ExecutionLayerFeeDispatcher.sol#L49
- ConsensusLayerFeeDispatcher.sol#L50
- ConsensusLayerFeeDispatcher.sol#L56
- AuthorizedFeeRecipient.sol#L29-L31

**Description:** The storage changes/actions in this context miss emitting an event.

**Recommendation:** Add a custom event for easier off-chain monitoring.

**Kiln:** Acknowledged.

- In practice, the current event feed is well setup in our off-chain monitoring tools and there is no product need to add more.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.
- It would be hard to manage the fact that new events are emitted for existing mainnet deployments.

**Spearbit:** Acknowledged.

#### 5.5.19 Wrong revert error on ETH transfer failure in `ExecutionLayerFeeDispatcher.sol`

**Severity:** Informational

**Context:**

- ExecutionLayerFeeDispatcher.sol#L78-L81
- ExecutionLayerFeeDispatcher.sol#L84-L87

**Description:** If the `treasury` rejects the ether, it will revert with the custom error `FeeRecipientReceiveError(data)` instead of `TreasuryReceiveError(data)`.

On the other hand, if the `operator` rejects the ether, it will revert with a custom error `TreasuryReceiveError(data)` instead of `FeeRecipientReceiveError(data)`.

**Recommendation:** Consider fixing the revert error message with the expected ones like you do it here: ConsensusLayerFeeDispatcher.sol#L97-L106;

```
diff --git a/src/contracts/ExecutionLayerFeeDispatcher.sol
↪  b/src/contracts/ExecutionLayerFeeDispatcher.sol
index 9c409c0..4ddeeeb 100644
--- a/src/contracts/ExecutionLayerFeeDispatcher.sol
+++ b/src/contracts/ExecutionLayerFeeDispatcher.sol
@@ -77,13 +77,13 @@ contract ExecutionLayerFeeDispatcher is IFeeDispatcher {
         if (globalFee > 0) {
             (status, data) = treasury.call{value: globalFee - operatorFee}("");
             if (status == false) {
-                revert FeeRecipientReceiveError(data);
+                revert TreasuryReceiveError(data);
             }
         }
         if (operatorFee > 0) {
             (status, data) = operator.call{value: operatorFee}("");
             if (status == false) {
-                revert TreasuryReceiveError(data);
+                revert FeeRecipientReceiveError(data);
             }
         }
         emit Withdrawal(
```

**Kiln:** Fixed in commit `180bba`.

**Spearbit:** Fixed.

#### 5.5.20 `ExecutionLayerFeeDispatcher`, `ConsensusLayerFeeDispatcher` - the treasury address is missing from the `Withdrawal` event

**Severity:** Informational

**Context:**

- ConsensusLayerFeeDispatcher.sol#L108-L115
- ExecutionLayerFeeDispatcher.sol#L89-L96

**Description:** Since the `treasury` address in the `StakingContract` can be updated, it's recommended to include this address as a parameter in the `Withdrawal` event. This ensures that any changes to the treasury address are reflected in future events.

**Kiln:** Acknowledged.

- In practice, the current event feed is well setup in our off-chain monitoring tools and there is no product need to add more info on these.
- It is in our best interest to keep the changes on the On-Chain v1 codebase minimal.
- It would be hard to manage the fact that new events are emitted for existing mainnet deployments.

**Spearbit:** Acknowledged.