sigma prime

KELP DAO

LRT-ETH Withdrawals

Security Assessment Report

Version: 2.1

June, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the KELP DAO smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the KELP DAO smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the KELP DAO smart contracts.

Overview

The Kelp DAO LRT (Liquid Restaking Token) project is a liquid restaking solution on Ethereum, designed to enhance the staking experience. It's a non-custodial protocol that allows users to stake their assets into and earn rewards without locking their funds, thereby maintaining liquidity.

The core components of the codebase are the following:

1. **`LRTConfig.sol`** Serves as the configuration center for the protocol. It manages the list of supported assets, their deposit limits and corresponding staking strategies.
2. **`LRTDepositPool.sol`** Handles the deposits of liquid staking tokens (LSTs) and facilitates the allocation of assets to various node delegators. It also manages the minting of rsETH tokens.
3. **`NodeDelegator.sol`** Manages the delegation of assets to different strategies and transferring assets back to `LRTDepositPool.sol` .
4. **`LRTWithdrawal.sol`** Handles all withdrawal requests for either LRT and Ether.
5. **`LRTUnstakingVault.sol`** Poses as a vault for unstaked assets ready for withdrawal.

6. `LRTConverter.sol` Enables asset exchange/conversion between withdrawn assets on EigenLayer with rsETH.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to files at commit ed6fa16.

The list of assessed contracts is as follows:

- LRTConverter.sol
- LRTDepositPool.sol
- LRTUnstakingVault.sol
- LRTWithdrawalManager.sol
- NodeDelegator.sol
- LRTConstants.sol
- DoubleEndedQueue.sol

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Coverage Limitations

Due to a time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 23 issues during this assessment. Categorised by their severity:

- Critical: 2 issues.
- Medium: 3 issues.
- Low: 6 issues.
- Informational: 12 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the KELP DAO smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
KELP-01	Variable `stakedButUnverifiedNativeETH` Is Never Decremented	Critical	Resolved
KELP-02	The Value of `ethValueInWithdrawal` Is Not Normalised To 18 Decimals	Critical	Resolved
KELP-03	NDC Index Shuffling Issue In `LRTDepositPool`	Medium	Closed
KELP-04	Dust Donation Disruption In NDC Removal Process	Medium	Resolved
KELP-05	Misrepresentation Of ETH Balance In NDCs	Medium	Resolved
KELP-06	Inconsistency In `minAmountToDeposit` Validation	Low	Resolved
KELP-07	`ETH_TOKEN` Not Supported By Default	Low	Closed
KELP-08	Potential Off-by-One Error In Withdrawal Handling	Low	Resolved
KELP-09	`rsETH` Oracle Updates Can Be Sandwiched	Low	Closed
KELP-10	Deposits Via `LRTConverter` May Be Vulnerable To Inflation Attacks	Low	Resolved
KELP-11	`minAmountToDeposit` Is Independent Of The Deposited Asset	Low	Closed
KELP-12	Potential `OutOfGas` Revert On `receive()` Function	Informational	Resolved
KELP-13	Missing Event Emission In `transferBackToLRTDepositPool()`	Informational	Resolved
KELP-14	Use OpenZeppelin's SafeERC20 Over Standard ERC20	Informational	Resolved
KELP-15	Enhancement For Asset Strategy Verification In `getAssetsUnstaking`	Informational	Resolved
KELP-16	Operational Redundancy In Node Delegator Limit Management	Informational	Closed
KELP-17	Gas Inefficiency In Multiple Modifier Checks	Informational	Resolved
KELP-18	Gas Efficiency Concern With NDC Iteration	Informational	Closed
KELP-19	Duplicate Code in `rsETH` Amount Calculation	Informational	Resolved
KELP-20	Conversion Limit Mapping Access Restricted	Informational	Resolved
KELP-21	Missing Input Checks In `completeUnstaking()`	Informational	Resolved
KELP-22	Superfluous Handling In NodeDelegator `receive` Function	Informational	Resolved
KELP-23	Miscellaneous General Comments	Informational	Resolved

KELP-01	Variable `stakedButUnverifiedNativeETH` Is Never Decremented		
Asset	`NodeDelegator.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The incorrect accounting of variable `stakedButUnverifiedNativeETH` may cause inaccurate `rsETH` price.

Variable `stakedButUnverifiedNativeETH` in `NodeDelegator` is meant to track the amount of native `ETH` staked through EigenLayer. As a result it is incremented when the function `stake32Eth()` is called. However, when a `NodeDelegator` withdraws `ETH` using functions `initiateNativeEthWithdrawBeforeRestaking()` and `claimNativeEthWithdraw()` it is not decremented.

When the unstaked `Ether` is then sent to a user who wishes to withdraw, the Ether will still be counted towards the total assets of the protocol even though it is no longer a part of the protocol. This results in the price of `rsETH` being higher than it should be. When users withdraw assets at this inflated `rsETH` price, the protocol will suffer significant losses.

Recommendations

Decrement the value of `stakedButUnverifiedNativeETH` when the `Ether` is unstaked to ensure the accounting of assets is accurate at all times.

Resolution

The development team has mitigated this issue by fixing the accounting of `stakedButUnverifiedNativeETH`, as per commit 7db0e43.

KELP-02	The Value of ethValueInWithdrawal Is Not Normalised To 18 Decimals		
Asset	LRTConverter.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

In function `convertEigenlayerAssetToRsEth()`, the value of `ethValueInWithdrawal` is not divided by the price of Ether, which will cause the `rsETH` price calculated in `LRTOracle` to be much higher than it should.

The variable `ethValueInWithdrawal` on line [117] is calculated as the product of `assetAmount` and `LRTOracle.getAssetPrice(asset)`, which are both in the scale of 10^{18}. As a result, `ethValueInWithdrawal` is in the scale of 10^{36}. This value would highly impact the price of `rsETH`.

`ethValueInWithdrawal` is used in `LRTDepositPool.getETHDistributionData()` as a part of `ethStakedInEigenLayer` in the following formula:

LRTDepositPool.sol

```
ethStakedInEigenLayer += ILRTConverter(lrtConverter).ethValueInWithdrawal();
```

Then, the function `LRTOracle.updateRSETHPrice()` calculates the `rsETH` price based on sum of the function `LRTDepositPool.getTotalAssetDeposits()` of the different assets as follows:

LRTDepositPool.sol

```
uint256 totalAssetAmt = ILRTDepositPool(lrtDepositPoolAddr).getTotalAssetDeposits(asset);
totalETHInPool += totalAssetAmt * assetER;
```

The variable `ethValueInWithdrawal` is a part of of the return value of `getTotalAssetDeposits()`. Therefore, the price of `rsETH` would be very high and up to 10^{36}.

Recommendations

Divide the `ethValueInWithdrawal` in `convertEigenlayerAssetToRsEth()` by the price of `ETH` which is 10^{18}.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-03	NDC Index Shuffling Issue In `LRTDepositPool`		
Asset	`LRTDepositPool.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `removeNodeDelegatorContractFromQueue()` function in `LRTDepositPool` contract employs a mechanism to remove a Node Delegator Contract (NDC) by swapping the target NDC with the last in the list on line [**336**], potentially altering NDC indices. This approach can lead to two primary issues:

1. **Race Condition**

   Operations like `transferAssetToNodeDelegator()` or `transferETHToNodeDelegator()` may revert if they target the last NDC index which gets removed or swapped during execution.

2. **Incorrect NDC Operation**

   If an NDC other than the last one is removed, subsequent operations might act on an incorrect NDC due to the shift in indices.

Recommendations

Consider implementing a more stable indexing mechanism that does not rely on the position within an array or explore the possibility of using a mapping structure to track NDCs, which inherently avoids the problem of shifting indices.

Additionally, introducing checks or mechanisms to handle ongoing operations gracefully during the removal process could prevent potential race conditions and ensure operational accuracy.

Resolution

The development team has closed the issue with the following comment.

> *"We are fine with this design. Removing a NDC is a very rare event. We will update our NDC indices in our docs and take care of operations."*

KELP-04	Dust Donation Disruption In NDC Removal Process		
Asset	`LRTDepositPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

A malicious entity could interfere with the removal of a Node Delegator Contract (NDC) by sending a trivial amount of `ETH` or other supported tokens to the NDC.

The vulnerability exists in the `LRTDepositPool` contract, specifically within the `removeNodeDelegatorContractFromQueue()` function on line [**281**], due to the function's requirement for the NDC to have a zero balance of both `ETH` and any supported tokens before proceeding with the removal.

Such a scenario could lead to operational inefficiencies and hinder the protocol's ability to manage NDCs effectively.

Recommendations

Consider implementing a threshold amount below which the balance is considered negligible and does not prevent NDC removal. This approach can minimize potential disruption by ensuring that dust amounts of assets do not hinder operational processes.

Resolution

The development team has fixed the issue by introducing the `maxNegligibleAmount` variable in commit 4054dad.

KELP-05	Misrepresentation Of ETH Balance In NDCs		
Asset	`LRTDepositPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The usage of `ethToStake()` instead of the actual balance (`address.balance()`) on line [**138**] of `LRTDepositPool` may not accurately reflect the total Ether held by Node Delegator Contracts (NDCs) `ethLyingInNDCs` .

This approach overlooks any `ETH` rewards that NDCs might have accumulated, potentially under reporting the total `ETH` available. Not accounting for the rewards would slightly impact the `rsETH` price since `ethLyingInNDCs` intervenes in the calculation of the price.

Recommendations

Consider replacing `ethToStake()` with a method or property that accounts for the total Ether balance of NDCs, including both staked Ether and any rewards. This change ensures a more accurate representation of Ether assets within the system.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-06	Inconsistency In `minAmountToDeposit` Validation		
Asset	`LRTDepositPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

In `LRTDepositPool` contract, the `setMinAmountToDeposit()` function lacks validation to ensure the new minimum deposit amount is greater than zero.

This omission could render the check on line [**228**] redundant, as setting `minAmountToDeposit` to zero would allow deposits of any size, contradicting the intended restriction mechanism.

Recommendations

Ensure `minAmountToDeposit_ > 0` in the setter function to enforce the protocol's minimum deposit threshold logic effectively. Alternatively, adjust the conditional check to `if (depositAmount < minAmountToDeposit)` if the intention is to allow `minAmountToDeposit` to be zero.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-07	`ETH_TOKEN` Not Supported By Default		
Asset	`NodeDelegator.sol, LRTDepositPool.sol, LRTWithdrawalManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `ETH_TOKEN` constant, representing Ethereum in the system, is not automatically included as a supported asset during the initialisation of `LRTConfig`, where only `stETH` and `ethX` are initially supported.

This oversight could impair critical system functionalities reliant on `ETH_TOKEN` being recognised as a supported asset.

The following functions across various contracts presume `ETH_TOKEN` as a supported asset for their operations:

1. `NodeDelegator.transferBackToLRTDepositPool()`
2. `LRTDepositPool.getAssetDistributionData()`
3. `LRTDepositPool.removeNodeDelegatorContractFromQueue()`
4. `LRTWithdrawalManager.completeWithdrawal()`

Recommendations

Ensure `ETH_TOKEN` is recognised as a supported asset within the system's configuration, particularly during the initialisation phase of `LRTConfig` or through an early protocol setup operation.

Resolution

The development team has opted to close this issue with the following comment:

> "Protocol supports `ETH_TOKEN` as seen in supportedAsset list: https://etherscan.io/address/0x947Cb49334e6571ccBFEF1f1f1178d8469D65ec7#readProxyContract#F8"

KELP-08	Potential Off-by-One Error In Withdrawal Handling		
Asset	`LRTWithdrawalManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In `LRTWithdrawalManager`, the condition on line [182] checks if a user's first withdrawal request nonce is greater than or equal to `nextLockedNonce[asset]` to determine if a withdrawal is still pending. This logic, combined with the usage of `upperLimit` in `_unlockWithdrawalRequests()`, suggests an off-by-one error, potentially causing confusion about which withdrawals are eligible for completion.

The comparison `if (usersFirstWithdrawalRequestNonce >= nextLockedNonce[asset] revert WithdrawalNotPending();` may incorrectly block withdrawals that should be unlockable, based on the natural expectation that `upperLimit` denotes the last request index to unlock.

Recommendations

To resolve potential confusion and align with expected behavior, consider revising the condition to check if `usersFirstWithdrawalRequestNonce` is strictly greater than `nextLockedNonce[asset]`, implying that the request is indeed locked, not pending. The error message should also be updated to `WithdrawalLocked()` to more accurately reflect the failed condition.

Additionally, review the logic surrounding `upperLimit` and its documentation to ensure it clearly indicates whether this index is included or excluded from the unlocking process, addressing any off-by-one discrepancies.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-09	`rsETH` Oracle Updates Can Be Sandwiched		
Asset	`LRTOracle.sol, LRTDepositPool.sol, LRTWithdrawalManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

An attacker could frontrun a `rsETH` price update by depositing before the price update and withdrawing right after it. In this manner, any incoming rewards can be sandwiched. Depending on the size of the rewards the profitability will differ, allowing an attacker to receive part of the yield without actually contributing to the protocol.

However, seeing as the withdrawal delay is rather large (+-8 days) the profitability would strongly depend on the size of the `rsETH` price update. As such, this is rated as a low severity issue.

Recommendations

Some of the possible ways to mitigate this include:

- Charge a small fee for depositing. The fee should be equal to the expected value of the price update.
- Enforce a minimum time required to be in the pool before withdrawing is allowed.
- Ensure price updates are small enough so that this attack is not profitable. This means that `updateRSETHPrice()` should be called regularly and that rewards should come in small, frequent batches.

Resolution

The development team has opted to close the issue with the following statement:

> *"We have a large withdrawal delay of 7-10 days and we update `rsETH` price every 1-2 days. Additionally, the malicious user would have to incur gas fees on the deposit and withdrawal, these should be large enough to discourage such activities."*

KELP-10	Deposits Via LRTConverter May Be Vulnerable To Inflation Attacks		
Asset	LRTConverter.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The function `convertEigenlayerAssetToRsEth()` allows users to convert their EigenLayer assets and receive `rsETH` in return, enabling an alternative way to deposit assets into the protocol. However, in contrast to depositing via `LRTDepositPool` there are 2 protections missing, exposing the user the price manipulation attacks such as inflation attacks:

1. There is no `minimumExpectedReturn` parameter. This means a user has no control over the amount of `rsETH` they wish to receive. As a result, an attacker could frontrun a deposit and skew the price of `rsETH` resulting in the user receiving less `rsETH` than they may expect.
2. There is no minimum deposit required. This allows an attacker to deposit a very small amount of assets, opening up the possibility for inflation attacks.

The testing team understands that the function `convertEigenlayerAssetToRsEth()` currently can only be called by the `LRTOperator`. Depending on how this is handled offchain, this could significantly decrease the likelihood of exploitation. Regardless, since the protocol team expressed interest in potentially allowing users to execute this function in the future, it is advisable to highlight these issues.

Recommendations

Add the two mentioned protections, similar to how they are implemented in `LRTDepositPool`.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-11	`minAmountToDeposit` Is Independent Of The Deposited Asset		
Asset	`LRTDepositPool.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `minAmountToDeposit` is the same for each asset that can be deposited. This can cause issues when the assets have a large difference in value per unit (= per wei), which commonly occurs with assets that have different number of decimals.

The variable `minAmountToDeposit` ensures that only deposits with an amount of assets larger than `minAmountToDeposit` are allowed. This provides protection against inflation attacks.

In order to launch a successful inflation attack an attacker must first deposit a small amount of assets to mint a small amount of shares. The attacker will then 'inflate' the value of these shares by donating assets. Because `minAmountToDeposit` protects against small deposits, it makes inflation attacks much less feasible, unless assets with differing decimals are involved:

```
function _beforeDeposit(
        address asset,
        uint256 depositAmount,
        uint256 minRSETHAmountExpected
   )
     private
     view
     returns (uint256 rsethAmountToMint)
   {
        if (depositAmount == 0 || depositAmount < minAmountToDeposit) {
            revert InvalidAmountToDeposit();
        }
```

For example: A realistic value for `minAmountToDeposit` would be 10^{16} or `0.01 ether`. This works for assets such as Ether with a relatively small value per unit. A problem arises when the protocol wishes to support assets with a larger value per unit such as `USDC`. Since it only has 6 decimals, its value per unit is substantially larger. In this case 10^{16} is no longer a realistic value for `minAmountToDeposit` since it requires a user to deposit 10^{10} USD. This would force the protocol to decrease `minAmountToDeposit` to enable users to make deposits with `USDC`, decreasing the protocol's resistance against inflation attacks as a result.

Recommendations

Change `minAmountToDeposit` into a mapping such that each supported asset can have its own value for `minAmountToDeposit`.

Resolution

The development team has opted to close the issue with the following statement.

> *"For the near future, we don't plan on non-eth based assets, so we are good for now. We will be sure to take care once we add assets like `USDC`, or others with high value per unit"*

KELP-12	Potential `OutOfGas` Revert On `receive()` Function
Asset	`NodeDelegator.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `receive()` functions fail with `OutOfGas` error message if the transaction of sending Ether is done through `transfer()` or `send()`. It is only successful through low-level `call()` due to the excessive gas usage on `receive()`.

In `NodeDelegator` contract, the issue arises because the Ether transfer using `transfer()` and `send()` methods have insufficient gas allowance to execute the entire commands on the `receive()` function.

The testing team understands that all of Ether transfers within the target contracts utilise low-level `call()`. However, there is no guarantee that external contracts that interact with the assessed system have a similar approach.

Recommendations

Ensure this behaviour is understood and well documented.

Resolution

The development has resolved the issue by adding documentation in commit 0dce7d4.

KELP-13	Missing Event Emission In `transferBackToLRTDepositPool()`
Asset	`NodeDelegator.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `transferBackToLRTDepositPool()` function in `NodeDelegator` contract lacks an event emission upon successful execution.

Events are crucial for tracking contract activities on the blockchain, providing transparency and enabling off-chain applications to react to changes. The absence of an event for asset transfers back to the `LRTDepositPool` may hinder the ability to monitor these transactions effectively.

Recommendations

Introduce an event emission within the `transferBackToLRTDepositPool()` function to signal the successful transfer of assets.

Resolution

The development has fixed the above issue as per commit 7db0e43.

KELP-14	Use OpenZeppelin's SafeERC20 Over Standard ERC20
Asset	`*.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The contracts use the `ERC20` standard and not the recommended `SafeERC20` standard from OpenZeppelin.

Recommendations

Consider using OpenZeppelin's `SafeERC20` over the standard `ERC20`.

Resolution

The development team has fixed the above issue by using the `SafeERC20` library, as per commit 7db0e43.

KELP-15	Enhancement For Asset Strategy Verification In `getAssetsUnstaking`
Asset	`LRTUnstakingVault.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `getAssetsUnstaking()` in `LRTUnstakingVault` contract currently lacks checks to ensure the passed asset is supported by a matching strategy, unlike similar checks present in functions within `LRTDepositPool`, such as `getAssetDistributionData()`.

This inconsistency could lead to queries for assets without strategies, potentially causing confusion or incorrect assumptions about the contract's state.

Recommendations

Consider implementing the `onlySupportedAsset` modifier for `getAssetsUnstaking()` to align with the verification patterns used in other parts of the codebase.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-16	Operational Redundancy In Node Delegator Limit Management
Asset	`LRTDepositPool.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `updateMaxNodeDelegatorLimit()` function in `LRTDepositPool` contract allows the `LRTAdmin` to modify the maximum number of Node Delegator Contracts (NDCs) that can be queued. Concurrently, adding new NDCs via `addNodeDelegatorContractToQueue()` is also restricted to the `LRTAdmin`.

This setup introduces unnecessary operational complexity without enhancing security, as it effectively means the `LRTAdmin` is imposing a limit on themselves, which could complicate managing NDCs.

Recommendations

Consider reassessing the necessity of dynamically managing the `maxNodeDelegatorLimit` given the sole authority of `LRTAdmin` over adding NDCs.

Simplifying this mechanism could involve setting a sensible permanent limit that accommodates expected growth or removing the limit altogether, assuming `LRTAdmin` manages NDC additions judiciously.

Reducing administrative friction in this area could streamline protocol operations and lower the risk of self-imposed operational bottlenecks.

Resolution

The development team has opted to close the above issue.

KELP-17	Gas Inefficiency In Multiple Modifier Checks
Asset	`LRTDepositPool.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

In `LRTDepositPool` contract, the function `removeManyNodeDelegatorContractsFromQueue()` on line [349] iteratively calls `removeNodeDelegatorContractFromQueue()`, invoking the `onlyLRTAdmin` modifier with each iteration.

This repeated check for administrative privileges introduces unnecessary gas costs, as the modifier's condition is unlikely to change between iterations within a single transaction.

Recommendations

Consider optimising gas usage by restructuring the code to perform administrative privilege checks only once per transaction.

One approach could involve consolidating the modifier check outside of the loop or redesigning the removal logic to reduce redundant checks.

Resolution

The development team has fixed the above issue in commit f98011b.

KELP-18	Gas Efficiency Concern With NDC Iteration
Asset	`LRTDepositPool.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The code segment from line [**111-118**] in `LRTDepositPool` contract involves iterating over Node Delegator Contracts (NDCs) to compute asset distributions.

This iterative approach can become gas-intensive and potentially exceed block gas limits as the number of NDCs increases, leading to failed transactions or prohibitive costs for users.

LRTDepositPool.sol

```
110 uint256 ndcsCount = nodeDelegatorQueue.length;
    for (uint256 i; i < ndcsCount;) {
112     assetLyingInNDCs += IERC20(asset).balanceOf(nodeDelegatorQueue[i]);
        assetStakedInEigenLayer += INodeDelegator(nodeDelegatorQueue[i]).getAssetBalance(asset);
114     unchecked {
            ++i;
116     }
    }
```

Recommendations

Consider optimizing the iteration mechanism to reduce gas consumption. One approach could be adopting a more gas-efficient data structure.

For example, using an advanced method such as OpenZeppelin's `EnumerableSet.AddressSet`. However, changing the data type would require refactoring on many parts of the contract.

Resolution

The development team has opted to close the above issue.

KELP-19	Duplicate Code in `rsETH` Amount Calculation
Asset	`LRTConverter.sol, LRTDepositPool.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `getRsETHAmountToMint()` in both `LRTConverter` and `LRTDepositPool` contracts contains identical code for calculating the amount of `rsETH` to mint based on a given asset amount.

This redundancy could lead to maintenance issues, such as the need to update the logic in two places if changes are required in the future, increasing the risk of inconsistencies.

Recommendations

Consider refactoring to eliminate duplicate code.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-20	Conversion Limit Mapping Access Restricted
Asset	`LRTConverter.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `conversionLimit` mapping in `LRTConverter` contract lacks a public getter, limiting the ability to access current asset conversion limits. This can hinder effective management and transparency.

Recommendations

Consider making `conversionLimit` public to automatically generate a getter or introduce a manual getter function for controlled access.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-21	Missing Input Checks In `completeUnstaking()`
Asset	`LRTUnstakingVault.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `completeUnstaking()` in `LRTUnstakingVault` contract does not check that the parameter `assets` has the same length as `queuedEigenLayerWithdrawal.shares`, even though this is implicitly assumed in the for-loop on line [**111**].

Additionally, no checks are performed to ensure that the length of `assets` is larger than zero.

LRTUnstakingVault.sol

```
96  function completeUnstaking(
        IStrategy.QueuedWithdrawal calldata queuedEigenLayerWithdrawal,
98      IERC20[] calldata assets,
        uint256 middlewareTimesIndex
100 )
        external
102     onlyLRTManager
        nonReentrant
104 {
        address eigenlayerStrategyManagerAddress = lrtConfig.getContract(LRTConstants.EIGEN_STRATEGY_MANAGER);
106
        // Finalize withdrawal with Eigenlayer Strategy Manager
108     IEigenStrategyManager(eigenlayerStrategyManagerAddress).completeQueuedWithdrawal(
            queuedEigenLayerWithdrawal, assets, middlewareTimesIndex, true
110     );
        for (uint256 i = 0; i < assets.length;) {
112         sharesUnstaking[address(assets[i])] -= queuedEigenLayerWithdrawal.shares[i];
            unchecked {
114             i++;
            }
116     }
        emit EigenLayerWithdrawalCompleted(
118         queuedEigenLayerWithdrawal.depositor, queuedEigenLayerWithdrawal.withdrawerAndNonce.nonce, msg.sender
        );
120 }
```

Recommendations

Consider implementing input validation checks to avoid unexpected behaviour and improve the readability of error messages.

Resolution

The development team has fixed the above issue in commit 931965c.

KELP-22	Superfluous Handling In NodeDelegator `receive` Function
Asset	`NodeDelegator.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `receive()` function in `NodeDelegator` primarily emits events to log the allocation of incoming `ETH` among different categories (exit validators, extra stake, rewards). However, the function doesn't perform any actual fund transfer or allocation beyond adjusting state variables and emitting events.

Recommendations

Consider enhancing documentation to explain the function's role and its interaction with the broader system.

Resolution

The development team has fixed the above issue as per commit 7db0e43.

KELP-23	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Incorrect NatSpec Comments For `transferToLRTUnstakingVault`**
   Related Asset(s): NodeDelegator.sol
   The NatSpec comments for the `transferToLRTUnstakingVault()` function in `NodeDelegator` contract incorrectly replicate those of the `transferBackToLRTDepositPool()` function.

   Consider updating the NatSpec comments for the `transferToLRTUnstakingVault()` function to accurately reflect its unique purpose.

2. **Incomplete Function Definition In `INodeDelegator`**
   Related Asset(s): INodeDelegator.sol
   The `INodeDelegator` contract interface does not contain the complete function definition of `NodeDelegator` contract.
   Consider adding the missing function definitions in the `INodeDelegator`.

3. **Redundant Information In `EigenLayerWithdrawalCompleted` Event**
   Related Asset(s): LRTUnstakingVault.sol
   In the `LRTUnstakingVault` contract, the `EigenLayerWithdrawalCompleted` event includes `msg.sender` as a parameter to indicate the caller of the transaction. However, given that the functions triggering this event can only be called by the `LRTManager` due to access control restrictions, the inclusion of `msg.sender` provides no additional useful information.
   Consider removing the `msg.sender` parameter from the `EigenLayerWithdrawalCompleted` event.

4. **Naming Confusion With `balanceOf` Function In `LRTUnstakingVault`**
   Related Asset(s): LRTUnstakingVault.sol
   The `balanceOf()` function in `LRTUnstakingVault` contract, used to return the contract's balance of a specified asset, shares its name with the widely recognized `ERC20` token standard's `balanceOf()` method, which returns the balance of tokens for a specific address.
   However, the mechanics and purpose of `LRTUnstakingVault`'s `balanceOf()` differ, as it pertains to the contract's asset holdings rather than an individual's token balance. This naming overlap could lead to confusion regarding the function's behavior and expectations, particularly for those familiar with `ERC20` interactions.
   Consider renaming the `balanceOf()` function in `LRTUnstakingVault` to more accurately reflect its functionality and distinguish it from the `ERC20` `balanceOf()`.

5. **`ethPricePerUint` Declaration Could Be Constant**
   Related Asset(s): LRTDepositPool.sol
   In `LRTDepositPool` contract, the variable `ethPricePerUint` is set to `1e18` on line [**430**] and is not modified elsewhere, indicating it has a constant value throughout the contract's lifecycle.
   Consider declaring `ethPricePerUint` as a constant at the contract level.

6. **Inconsistent Naming Convention for Internal Functions**

   Related Asset(s): LRTDepositPool.sol

   The `LRTDepositPool` contract does not consistently use the `_` prefix for internal function names, a convention observed in other contracts such as `LRTWithdrawalManager` (e.g., `_addUserWithdrawalRequest()`). This inconsistency in naming conventions may lead to confusion and diminish code readability, particularly in distinguishing between external/public and internal/private function calls at a glance.

   Align with the established naming convention by prefixing all internal/private functions with an underscore `_`. This change will enhance consistency across the codebase, improving readability and maintainability.

7. **Typos In Function Comments And Naming**

   Related Asset(s): LRTDepositPool.sol, LRTConverter.sol, ILRTConverter.sol, NodeDelegator.sol

   There's a typographical errors on the following lines:

   - On line [**106**] in `LRTDepositPool`'s comments: "updagraes" should be "upgrades".
   - On line [**74**] in `LRTDepositPool`'s function: `checkIfDepositAmountExceedesCurrentLimit()` should be `checkIfDepositAmountExceedsCurrentLimit()`
   - On line [**99**] in `LRTConverter`'s function: `WithdrawalRootAlreadyProcess()` should be `WithdrawalRootAlreadyProcessed()`
   - On line [**13**] in `ILRTConverter`'s parameter: "reciever" should be "receiver"
   - On line [**208**] in `LRTWithdrawalManager`'s comment: "Lasts withdrawal requests index to consider unlocking." should be "The last withdrawal request index to unlock".
   - On line [**315**] in `NodeDelegator`'s comment: "completle" should be "completely"

   Consider correcting the typos to enhance clarity and documentation quality.

8. **Missing NatSpec Documentation On `finalizeConversion`**

   Related Asset(s): LRTConverter.sol

   The `finalizeConversion()` function in `LRTConverter` is missing NatSpec documentation.

   Consider adding comprehensive NatSpec comments to the `finalizeConversion()` function.

9. **Function Visibility Suggestion for `initiateUnstaking`**

   Related Asset(s): NodeDelegator.sol

   The `initiateUnstaking()` function in `NodeDelegator` is currently defined with public visibility. Given its specific use case and the absence of a need for it to be called internally by other functions within the NodeDelegator contract, switching its visibility to external could optimize gas costs.

   Consider changing the visibility of the `initiateUnstaking()` function from public to external.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have fixed issues above as deemed necessary, as per commit 7db0e43.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `Forge` framework was used to perform these tests and the output is given below.

```
[PASS] testFail_convertEigenlayerAssetToRsEth_finalizeConversion() (gas: 4159733)
[PASS] test_addConvertableAsset_removeConvertableAsset_happyPath(address) (runs: 1000, μ: 47992, ~: 47978)
[PASS] test_getRsETHAmountToMint(uint8,uint256,uint256) (runs: 1000, μ: 69967, ~: 70119)
[PASS] test_removeConvertableAsset_happyPath(address) (runs: 1000, μ: 39016, ~: 39016)
[PASS] test_setConversionLimit(address,uint256) (runs: 1000, μ: 54791, ~: 55488)
[PASS] test_swapEthToAsset_happyPath() (gas: 3781689)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 688.86ms

Running 19 tests for test/LRTDepositPool.t.sol:LRTDepositPoolTest
[PASS] testFail_depositAsset_inflation_noProtection() (gas: 395490)
[PASS] test_addNodeDelegatorContractToQueue(uint256) (runs: 1000, μ: 8021823, ~: 7790137)
[PASS] test_addNodeDelegatorContractToQueue_duplicate() (gas: 113793)
[PASS] test_addNodeDelegatorContractToQueue_maximumLimitReached(address[]) (runs: 1000, μ: 61368, ~: 60917)
[PASS] test_depositAsset(uint256) (runs: 1000, μ: 427061, ~: 427061)
[PASS] test_depositAsset_inflation_protected() (gas: 380833)
[PASS] test_depositETH(uint256) (runs: 1000, μ: 213644, ~: 213644)
[PASS] test_deposit_with_node_delegator(uint256) (runs: 1000, μ: 1332840, ~: 1332840)
[PASS] test_getRsETHAmountToMint_depositAsset() (gas: 505709)
[PASS] test_getRsETHAmountToMint_initial() (gas: 82647)
[PASS] test_pause_unpause() (gas: 50859)
[PASS] test_removeManyNodeDelegatorContractsFromQueue() (gas: 398233)
[PASS] test_removeNodeDelegatorContractFromQueue() (gas: 352577)
[PASS] test_removeNodeDelegatorContractFromQueue_nonExistence() (gas: 309672)
[PASS] test_setMinAmountToDeposit(uint256) (runs: 1000, μ: 56365, ~: 57102)
[PASS] test_swapETHForAssetWithinDepositPool(uint256) (runs: 1000, μ: 309356, ~: 309356)
[PASS] test_transferAssetToNodeDelegator() (gas: 1378374)
[PASS] test_transferETHToNodeDelegator() (gas: 731579)
[PASS] test_updateMaxNodeDelegatorLimit(uint256) (runs: 1000, μ: 41732, ~: 41985)
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 5.13s

Running 17 tests for test/LRTWithdrawalManager.t.sol:LRTWithdrawalManagerTest
[PASS] test_initiateWithdrawal_exceedAmount(uint8,uint256,uint256) (runs: 1000, μ: 485945, ~: 488768)
[PASS] test_initiateWithdrawal_exceedAmountToWithdraw() (gas: 1516690)
[PASS] test_initiateWithdrawal_exceedsDepositAmount(uint8,uint256,uint256) (runs: 1000, μ: 321955, ~: 322192)
[PASS] test_initiateWithdrawal_happyPath(uint8,uint256,uint256) (runs: 1000, μ: 489096, ~: 492334)
[PASS] test_initiateWithdrawal_invalidAmountToWithdraw(uint8,uint256,uint256) (runs: 1000, μ: 316165, ~: 316284)
[PASS] test_initiateWithdrawal_noApproveRsETH(uint8,uint256,uint256) (runs: 1000, μ: 299191, ~: 299396)
[PASS] test_initiateWithdrawal_unlockQueue_completeWithdrawal_eth_happyPath() (gas: 54748299)
[PASS] test_initiateWithdrawal_unlockQueue_completeWithdrawal_happyPath(uint8) (runs: 1000, μ: 3761573, ~: 3761600)
[PASS] test_pause_paused() (gas: 61748)
[PASS] test_pause_unpause_happyPath() (gas: 49248)
[PASS] test_receive(uint256) (runs: 1000, μ: 22580, ~: 22848)
[PASS] test_setMinAmountToWithdraw(uint256) (runs: 1000, μ: 38133, ~: 38480)
[PASS] test_setWithdrawalDelayBlocks_happyPath(uint256) (runs: 1000, μ: 42578, ~: 42342)
[PASS] test_setWithdrawalDelayBlocks_tooSmall(uint256) (runs: 1000, μ: 35707, ~: 35934)
[PASS] test_unlockQueue_emptyUnstakingVault(uint8,uint256,uint256,uint256) (runs: 1000, μ: 107056, ~: 107136)
[PASS] test_unlockQueue_noPendingWithdrawals(uint8,uint256,uint256,uint256) (runs: 1000, μ: 332542, ~: 332526)
[PASS] test_unpause_notPaused() (gas: 33566)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 11.96s

Running 6 tests for test/DoubleEndedQueue.t.sol:DoubleEndedQueueTest
[PASS] test_clear(uint256) (runs: 1000, μ: 1292259, ~: 1292532)
[PASS] test_popBack(uint256) (runs: 1000, μ: 5349661, ~: 4102090)
[PASS] test_popFront(uint256) (runs: 1000, μ: 5696676, ~: 4428895)
[PASS] test_pushBack(uint256,uint256) (runs: 1000, μ: 1393959, ~: 1397969)
[PASS] test_pushBack(uint256[]) (runs: 1000, μ: 3111329, ~: 3104249)
[PASS] test_pushFront(uint256,uint256) (runs: 1000, μ: 1398941, ~: 1377320)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 118.05s

Running 6 tests for test/LRTUnstakingVault.t.sol:LRTUnstakingVaultTest
[PASS] test_addSharesUnstaking_happyPath(uint8,address,uint256) (runs: 1000, μ: 351627, ~: 352605)
```

```
[PASS] test_addSharesUnstaking_notLRTNodeDelegator(uint8,address,uint256) (runs: 1000, μ: 56671, ~: 56903)
[PASS] test_completeUnstaking((address[],uint256[],address,(address,uint96),uint32,address),uint256) (runs: 1000, μ: 446057, ~:
      ↪  446180)
[PASS] test_receive(uint256) (runs: 1000, μ: 24428, ~: 24723)
[PASS] test_redeem_eth(uint256) (runs: 1000, μ: 53010, ~: 52720)
[PASS] test_redeem_token(uint256) (runs: 1000, μ: 566709, ~: 566500)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 118.05s

Running 19 tests for test/NodeDelegator.t.sol:NodeDelegatorTest
[PASS] testFail_receive_send() (gas: 17085)
[PASS] testFail_receive_transfer() (gas: 16980)
[PASS] test_createEigenPod() (gas: 364223)
[PASS] test_depositAssetIntoStrategy() (gas: 230314)
[PASS] test_initiateNativeEthWithdrawBeforeRestaking_claimNativeEthWithdraw_transferToLRTUnstakingVault() (gas: 54471209)
[PASS] test_initiateUnstaking() (gas: 543893)
[PASS] test_maxApproveToEigenStrategyManager() (gas: 598929)
[PASS] test_receive_call() (gas: 30574)
[PASS] test_receive_case1() (gas: 78101)
[PASS] test_receive_case2() (gas: 80847)
[PASS] test_receive_case3() (gas: 97919)
[PASS] test_sendETHFromDepositPoolToNDC() (gas: 66024)
[PASS] test_stake32Eth() (gas: 427217)
[PASS] test_stake32EthValidated() (gas: 493563)
[PASS] test_transferBackToLRTDepositPool_ETH() (gas: 62666)
[PASS] test_transferBackToLRTDepositPool_asset() (gas: 112628)
[PASS] test_transferToLRTUnstakingVault() (gas: 63224)
[PASS] test_upgradeAndCall_ndc() (gas: 5752767)
[PASS] test_upgrade_ndc() (gas: 2876846)
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 118.05s

Ran 6 test suites: 73 tests passed, 0 failed, 0 skipped (73 total tests)
sonicskye@GCI9:~/sigp/stader/kelp-lrt-eth-withdrawals/Kelp-LRT-ETH-Withdrawals-review/test-forge$ forge test
[:] Compiling...
No files changed, compilation skipped

Running 19 tests for test/NodeDelegator.t.sol:NodeDelegatorTest
[PASS] testFail_receive_send() (gas: 17085)
[PASS] testFail_receive_transfer() (gas: 16980)
[PASS] test_createEigenPod() (gas: 364223)
[PASS] test_depositAssetIntoStrategy() (gas: 230314)
[PASS] test_initiateNativeEthWithdrawBeforeRestaking_claimNativeEthWithdraw_transferToLRTUnstakingVault() (gas: 54471209)
[PASS] test_initiateUnstaking() (gas: 543893)
[PASS] test_maxApproveToEigenStrategyManager() (gas: 598929)
[PASS] test_receive_call() (gas: 30574)
[PASS] test_receive_case1() (gas: 78101)
[PASS] test_receive_case2() (gas: 80847)
[PASS] test_receive_case3() (gas: 97919)
[PASS] test_sendETHFromDepositPoolToNDC() (gas: 66024)
[PASS] test_stake32Eth() (gas: 427217)
[PASS] test_stake32EthValidated() (gas: 493563)
[PASS] test_transferBackToLRTDepositPool_ETH() (gas: 62666)
[PASS] test_transferBackToLRTDepositPool_asset() (gas: 112628)
[PASS] test_transferToLRTUnstakingVault() (gas: 63224)
[PASS] test_upgradeAndCall_ndc() (gas: 5752767)
[PASS] test_upgrade_ndc() (gas: 2876846)
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 306.17ms

Running 19 tests for test/LRTDepositPool.t.sol:LRTDepositPoolTest
[PASS] testFail_depositAsset_inflation_noProtection() (gas: 395490)
[PASS] test_addNodeDelegatorContractToQueue(uint256) (runs: 1000, μ: 8218755, ~: 7790137)
[PASS] test_addNodeDelegatorContractToQueue_duplicate() (gas: 113793)
[PASS] test_addNodeDelegatorContractToQueue_maximumLimitReached(address[]) (runs: 1000, μ: 61385, ~: 61537)
[PASS] test_depositAsset(uint256) (runs: 1000, μ: 427051, ~: 427061)
[PASS] test_depositAsset_inflation_protected() (gas: 380833)
[PASS] test_depositETH(uint256) (runs: 1000, μ: 213644, ~: 213644)
[PASS] test_deposit_with_node_delegator(uint256) (runs: 1000, μ: 1332840, ~: 1332840)
[PASS] test_getRsETHAmountToMint_depositAsset() (gas: 505709)
[PASS] test_getRsETHAmountToMint_initial() (gas: 82647)
[PASS] test_pause_unpause() (gas: 50859)
[PASS] test_removeManyNodeDelegatorContractsFromQueue() (gas: 398233)
```

```
[PASS] test_removeNodeDelegatorContractFromQueue() (gas: 352577)
[PASS] test_removeNodeDelegatorContractFromQueue_nonExistence() (gas: 309672)
[PASS] test_setMinAmountToDeposit(uint256) (runs: 1000, μ: 56425, ~: 57102)
[PASS] test_swapETHForAssetWithinDepositPool(uint256) (runs: 1000, μ: 309356, ~: 309356)
[PASS] test_transferAssetToNodeDelegator() (gas: 1378374)
[PASS] test_transferETHToNodeDelegator() (gas: 731579)
[PASS] test_updateMaxNodeDelegatorLimit(uint256) (runs: 1000, μ: 41817, ~: 41985)
Test result: ok. 19 passed; 0 failed; 0 skipped; finished in 4.48s

Running 6 tests for test/LRTConverter.t.sol:LRTConverterTest
[PASS] testFail_convertEigenlayerAssetToRsEth_finalizeConversion() (gas: 4159733)
[PASS] test_addConvertableAsset_removeConvertableAsset_happyPath(address) (runs: 1000, μ: 47993, ~: 47978)
[PASS] test_getRsETHAmountToMint(uint8,uint256,uint256) (runs: 1000, μ: 69940, ~: 70119)
[PASS] test_removeConvertableAsset_happyPath(address) (runs: 1000, μ: 39016, ~: 39016)
[PASS] test_setConversionLimit(address,uint256) (runs: 1000, μ: 54791, ~: 55488)
[PASS] test_swapEthToAsset_happyPath() (gas: 3781689)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 13.24s

Running 18 tests for test/LRTWithdrawalManager.t.sol:LRTWithdrawalManagerTest
[PASS] test_initiateWithdrawal_exceedAmount(uint8,uint256,uint256) (runs: 1000, μ: 485828, ~: 488768)
[PASS] test_initiateWithdrawal_exceedAmountToWithdraw() (gas: 1516690)
[PASS] test_initiateWithdrawal_exceedsDepositAmount(uint8,uint256,uint256) (runs: 1000, μ: 321989, ~: 322192)
[PASS] test_initiateWithdrawal_happyPath(uint8,uint256,uint256) (runs: 1000, μ: 489801, ~: 492334)
[PASS] test_initiateWithdrawal_invalidAmountToWithdraw(uint8,uint256,uint256) (runs: 1000, μ: 316123, ~: 316284)
[PASS] test_initiateWithdrawal_noApproveRsETH(uint8,uint256,uint256) (runs: 1000, μ: 299196, ~: 299396)
[PASS] test_initiateWithdrawal_unlockQueue_completeWithdrawal_eth_happyPath() (gas: 54748299)
[PASS] test_initiateWithdrawal_unlockQueue_completeWithdrawal_happyPath(uint8) (runs: 1000, μ: 3761580, ~: 3761600)
[PASS] test_initiateWithdrawal_unlockQueue_completeWithdrawal_multiWithdrawals(uint8) (runs: 1000, μ: 3860119, ~: 3860155)
[PASS] test_pause_paused() (gas: 61748)
[PASS] test_pause_unpause_happyPath() (gas: 49248)
[PASS] test_receive(uint256) (runs: 1000, μ: 22593, ~: 22848)
[PASS] test_setMinAmountToWithdraw(uint256) (runs: 1000, μ: 38178, ~: 38480)
[PASS] test_setWithdrawalDelayBlocks_happyPath(uint256) (runs: 1000, μ: 42575, ~: 42342)
[PASS] test_setWithdrawalDelayBlocks_tooSmall(uint256) (runs: 1000, μ: 35699, ~: 35934)
[PASS] test_unlockQueue_emptyUnstakingVault(uint8,uint256,uint256,uint256) (runs: 1000, μ: 107072, ~: 107136)
[PASS] test_unlockQueue_noPendingWithdrawals(uint8,uint256,uint256,uint256) (runs: 1000, μ: 332538, ~: 332574)
[PASS] test_unpause_notPaused() (gas: 33566)
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 13.24s

Running 6 tests for test/DoubleEndedQueue.t.sol:DoubleEndedQueueTest
[PASS] test_clear(uint256) (runs: 1000, μ: 1294307, ~: 1292532)
[PASS] test_popBack(uint256) (runs: 1000, μ: 5576345, ~: 4102111)
[PASS] test_popFront(uint256) (runs: 1000, μ: 5595027, ~: 4009491)
[PASS] test_pushBack(uint256,uint256) (runs: 1000, μ: 1417868, ~: 1397969)
[PASS] test_pushBack(uint256[]) (runs: 1000, μ: 3224739, ~: 3346214)
[PASS] test_pushFront(uint256,uint256) (runs: 1000, μ: 1401407, ~: 1377320)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 28.34s

Running 8 tests for test/LRTUnstakingVault.t.sol:LRTUnstakingVaultTest
[PASS] testFail_receive_send_unstakingVault(uint256) (runs: 1000, μ: 33412, ~: 34021)
[PASS] testFail_receive_transfer_unstakingVault(uint256) (runs: 1000, μ: 17498, ~: 17787)
[PASS] test_addSharesUnstaking_happyPath(uint8,address,uint256) (runs: 1000, μ: 351519, ~: 352694)
[PASS] test_addSharesUnstaking_notLRTNodeDelegator(uint8,address,uint256) (runs: 1000, μ: 56682, ~: 56903)
[PASS] test_completeUnstaking((address[],uint256[],address,(address,uint96),uint32,address),uint256) (runs: 1000, μ: 447127, ~:
      ↪  444668)
[PASS] test_receive_call_unstakingVault(uint256) (runs: 1000, μ: 24459, ~: 24788)
[PASS] test_redeem_eth(uint256) (runs: 1000, μ: 53002, ~: 52720)
[PASS] test_redeem_token(uint256) (runs: 1000, μ: 566715, ~: 566500)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 125.38s

Ran 6 test suites: 76 tests passed, 0 failed, 0 skipped (76 total tests)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact				
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].

σ'