BAIL security

Kelp DAO

Core – Differential

FINAL REPORT

December ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:

Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Kelp Dao – Core – Differential - Audit Report
Website	kerneldao.com
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/Kelp-DAO/KelpDAO-contracts/tree/5d930dddffffd3f7a3fcd763f88d5b2fb13b112a
Resolution 1	https://github.com/Kelp-DAO/KelpDAO-contracts/tree/2db8d320b798c9d11566c95d5e25a7cc45cf9642

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no changes made)	Failed resolution	Open
High						
Medium						
Low	7	5		2		
Informational	8	5		3		
Governance	1			1		
Total	16	10		6		

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

LRTConfig

LRTConfig is a configuration management contract that serves as the central registry for protocol parameters, supported assets, contract addresses, and role-based access control. This contract is used throughout the protocol by all major components including LRTDepositPool, LRTOracle, NodeDelegator, and LRTWithdrawalManager to retrieve configuration data and validate asset support. The diff introduces an emergency pause mechanism allowing protocol operators to halt all pausable contracts simultaneously in critical situations.

The new pauseAll function enables PAUSER_ROLE holders to pause the deposit pool, withdrawal manager, oracle, rsETH token, and all node delegators in a single transaction. This centralized pause capability provides a coordinated emergency response mechanism across the entire protocol. The function iterates through all node delegators in the queue and pauses each one that isn't already paused, ensuring system-wide protection during critical events. Access control is enforced through DEFAULT_ADMIN_ROLE for administrative functions, MANAGER role for operational parameters, and TIME_LOCK_ROLE for adding new supported assets.

Privileged Functions

- pauseAll

Core Invariants:

INV 1: Only PAUSER_ROLE can trigger the pauseAll emergency function.

LRTConverter

LRTConverter is an adapter contract managing LST unstaking operations and tracking ETH value during conversions. Used by operators to unstake stETH to ETH via Lido's withdrawal queue while maintaining protocol accounting. The diff transitions asset transfer functions from OPERATOR_ROLE to the new ASSET_TRANSFER_ROLE for improved permission granularity.

The access control change for transferAssetFromDepositPool and transferAssetToDepositPool now requires ASSET_TRANSFER_ROLE instead of OPERATOR_ROLE, enabling separation of fund movement privileges from general operational tasks. These functions update the ethValueInWithdrawal accounting variable to track value locked in conversion processes. The OPERATOR_ROLE retains control over actual unstaking operations via unstakeStEthnd claiming via claimStEth, while MANAGER role manages the whitelist for withdrawal intent declarations. The withinUnstakeLimits modifier consumes whitelisted allowance before checking against active user withdrawals.

Privileged Functions

- transferAssetFromDepositPool
- transferAssetToDepositPool

Core Invariants:

INV 1: Only ASSET_TRANSFER_ROLE can transfer assets between deposit pool and converter.

LRTDepositPool

LRTDepositPool is the primary entry point for users depositing LSTs and ETH into the protocol in exchange for rsETH shares. Used by end users to enter the protocol and by operators to distribute assets to NodeDelegators and the unstaking vault. The diff removes legacy unstaking vault asset tracking and transitions asset transfer functions from OPERATOR_ROLE to the new ASSET_TRANSFER_ROLE for improved permission separation.

The removal of getAssetsUnstaking call in getAssetDistributionData eliminates legacy tracking of assets in delayed EigenLayer withdrawals at the vault level, as this is now handled elsewhere. The removal of assetUnstakingFromEigenLayer initialization from getAssetDistributionData for LSTs reflects the same legacy cleanup. Access control for transferAssetToNodeDelegator, transferETHToNodeDelegator, transferAssetToLRTUnstakingVault, and transferETHToLRTUnstakingVault now requires ASSET_TRANSFER_ROLE instead of OPERATOR_ROLE, creating a dedicated permission for fund movements. The pause function now requires PAUSER_ROLE instead of LRT_MANAGER for consistency.

Privileged Functions

- transferAssetToNodeDelegator
- transferETHToNodeDelegator
- transferAssetToLRTUnstakingVault
- transferETHToLRTUnstakingVault

Core Invariants:

INV 1: Only ASSET_TRANSFER_ROLE can transfer assets to node delegators and unstaking vault.

LRTOracle

LRTOracle is a price calculation contract that computes rsETH exchange rates relative to ETH based on total protocol assets and supply. Used by LRTDepositPool to determine mint amounts during deposits and by LRTWithdrawalManager for withdrawal calculations. The diff introduces period alignment for fee minting limits through a reinitialize function, adds view functions for transparency into daily limits and reset timing, and refines the fee minting limit check logic while removing an unreachable TVL invariant check.

The reinitialize function sets feePeriodStartTime within the last 24 hours to establish aligned daily periods for fee minting limits, ensuring consistent reset times. New view functions getCurrentPeriodStartTime, remainingDailyFeeMintLimit, and getNextDailyLimitResetTimestamp provide transparency into the fee minting system's state. The _checkAndUpdateDailyFeeMintLimit now initializes the period if unset and uses getCurrentPeriodStartTime for precise period alignment when resetting, preventing period drift. The previous protocol fee TVL increase invariant check was removed as unreachable due to mathematical constraints. Pause access changed from MANAGER to PAUSER_ROLE for consistency with protocol-wide emergency controls.

Privileged Functions

- pause

Core Invariants:

INV 1: Fee minting cannot exceed maxFeeMintAmountPerDay in any 24-hour period.

Issue_01	Redundant `feePeriodStartTime == 0` check
Severity	**Info**
Description	`feePeriodStartTime` is set in reinitializer, which means it's value cannot be 0 as there is no function to set/reset it otherwise. This means the feePeriodStartTime checks have no effect since it is never true.
Recommendations	Consider removing these checks.
Comments / Resolution	Fixed by following recommendation.

LRTWithdrawalManager

LRTWithdrawalManager is a request-based withdrawal system enabling users to convert rsETH back to LSTs or ETH through queued withdrawals and instant withdrawals. Used by rsETH holders to exit positions, with operators unlocking queued requests as assets become available. The diff introduces Aave v3 integration for idle ETH yield generation, removes initialization requirements from completion functions, adds customizable instant withdrawal fee recipients, and enhances access control for queue unlocking.
The Aave integration deposits unlocked ETH awaiting withdrawal into Aave v3 to earn yield for the protocol treasury, with principal tracking to prevent withdrawing accrued interest for user redemptions. New functions configureAaveIntegration, setAaveIntegrationEnabled, depositIdleETHToAave, collectInterestToTreasury, and emergencyWithdrawFromAave manage this integration. The unlockQueue function now accepts both ASSET_TRANSFER_ROLE and OPERATOR_ROLE via onlyAssetTransferOrOperatorRole modifier, enabling more flexible operational control. The instantWithdrawal function now supports directing fees to a configurable recipient address via setInstantWithdrawalFeeRecipient, defaulting to protocol treasury if unset. The _processWithdrawalCompletion automatically withdraws from Aave when the contract balance is insufficient to fulfill redemptions.

Privileged Functions

- configureAaveIntegration
- setAaveIntegrationEnabled
- depositToAaveExternal
- depositIdleETHToAave
- collectInterestToTreasury
- emergencyWithdrawFromAave

Core Invariants:

INV 1: Only ASSET_TRANSFER_ROLE or OPERATOR_ROLE can unlock withdrawal queues.
INV 2: Only MANAGER role can configure Aave integration, enable/disable it, set instant withdrawal parameters, and sweep remaining assets.
INV 3: Only OPERATOR_ROLE can manually deposit idle ETH to Aave and collect accrued interest.
INV 4: Total unlocked ETH in Aave cannot be withdrawn for user redemptions, only principal.
INV 5: Aave integration cannot be enabled without valid gateway, aWETH, pool, and data provider addresses.
INV 6: Interest collection only succeeds if Aave balance exceeds deposited principal.
INV 7: When Aave is disabled, all ETH must be withdrawn from Aave before disabling.

Issue_02	Emergency withdrawal locks unclaimed interest
Severity	**Low**
Description	The emergencyWithdrawFromAave function leaves accrued interest locked in the contract when executed without first collecting interest to treasury. When managers perform emergency withdrawals to pull principal from Aave, any interest that has accumulated remains as idle ETH in the LRTWithdrawalManager contract. This interest cannot be easily recovered because the sweepRemainingAssets function requires all unlocked withdrawals to be completed before sweeping. If the withdrawn funds are subsequently redeposited to Aave, the totalETHDepositedToAave increases to include the previously earned interest, effectively converting protocol-owned interest into principal. This results in permanent loss of the interest attribution, preventing the protocol from claiming those earnings as treasury revenue.
Recommendations	Collect interest to the treasury before calling the withdraw function in emergencyWithdrawFromAave.
Comments / Resolution	Fixed by following recommendation.

Issue_03	Disabling Aave reverts with accrued interest
Severity	**Low**
Description	The setAaveIntegrationEnabled function cannot successfully disable Aave integration when interest has accrued or donations exist, blocking exits. When disabling, the function attempts to withdraw the full aaveBalance which includes both principal and accumulated interest. However, the _withdrawFromAave function enforces principal-only withdrawals, calculating withdrawablePrincipal as the minimum of aaveBalance and totalETHDepositedToAave. When interest accrues, aaveBalance exceeds totalETHDepositedToAave, making withdrawablePrincipal equal to totalETHDepositedToAave. The withdrawal request for the larger aaveBalance then reverts with InsufficientPrincipalOnAave. This prevents managers from disabling Aave integration in emergency scenarios, such as Aave pool exploits or liquidity crises, until interest is separately collected.
Recommendations	Collect interest before withdrawing from Aave when disabling integration, and consider directly calling the gateways withdrawETH instead of using _withdrawFromAave.
Comments / Resolution	Fixed by collecting interest first and modifying the _withdrawFromAave.

Issue_04	Reconfiguration locks funds in old Aave pool
Severity	Low
Description	The configureAaveIntegration function updates Aave contract addresses without withdrawing deposited funds from the existing configuration, rendering those assets inaccessible. When managers reconfigure Aave integration, the function revokes approval for the old aWETH token and updates all storage variables to point to new contracts. However, any ETH previously deposited to the old Aave pool remains in the old aaveAWETH contract while the totalETHDepositedToAave counter continues tracking those deposits. The contract loses the ability to withdraw from the old pool because all internal functions now reference the new aaveAWETH address. Unlike setAaveIntegrationEnabled which withdraws all funds before disabling, configureAaveIntegration provides no mechanism to recover funds from the previous configuration, creating a permanent asset lock scenario.
Recommendations	Withdraw all interest and funds from Aave at the beginning of configureAaveIntegration as well as update related state variables before updating to new contract addresses.
Comments / Resolution	Fixed by following recommendation.

Issue_05	Missing Aave capacity check leads to capital inefficiency
Severity	Low
Description	The unlockQueue function attempts to deposit the full assetAmountUnlocked to Aave without checking available pool capacity, leading to capital inefficiency. When the Aave pool is at or near its supply cap, the entire deposit fails in the try-catch block and funds remain idle in the contract. The contract implements getAaveAvailableCapacity to query remaining capacity, but unlockQueue does not utilize this before attempting deposits. When Aave has partial capacity available, the protocol misses yield generation opportunities. For example, if 10 ETH of capacity exists but 100 ETH is unlocked, the deposit fails completely when 10 ETH could have been deposited successfully. The comment acknowledges pool capacity failures, but the implementation does not optimize for partial deposits that would maximize capital.
Recommendations	Compare the deposit amount to the available capacity and deposit up to the cap if the amount exceeds capacity.
Comments / Resolution	Acknowledged.

Issue_06	Dangling approval for permanently disabled aave integration
Severity	**Low**
Description	In LRTWithdrawalManger, whenever the aave configuration is changed via `configureAaveIntegration` the approval is revoked for previous configuration. However in the case the protocol decides to permanently disable aave integration rather than switch to a new config the approval for this disable config cannot be revoked since disabling is done via `setAaveIntegrationEnabled.`
Recommendations	Consider also revoking and applying approvals on enable and disable operations.
Comments / Resolution	Fixed by following recommendation.

Issue_07	Dust amount of profits can be left unclaimed due to rounding
Severity	**Informational**
Description	There can exist a 1-2 wei rounding when querying the aaveAWETH balanceOf() and this case is handled in _checkHealthAave() . But the interest amount calculation does not account for this rounding in the collectInterestToTreasury() function , this means that the interestAmount calculation can be 1-2 wei off too and some aWETH can be left unclaimed.
Recommendations	Consider acknowledging the issue.
Comments / Resolution	Acknowledged.

Issue_08	instantWithdrawalFeeRecipient cannot be set to zero
Severity	**Informational**

Description	When the instantWithdrawalFeeRecipient address is set, all instant withdrawal fees are redirected to it instead of the treasury. When it is not set, the fees are sent to the treasury.
	However function setInstantWithdrawalFeeRecipient does not allow setting the fee recipient back to zero address due checkNonZeroAddress. Setting it to zero address should be allowed since the instant withdrawal fees can then be directed to the treasury.
	address feeRecipient = instantWithdrawalFeeRecipient;
	    if (feeRecipient == address(0)) {
	      // Backwards-compatible default: send fees to the protocol treasury
	      feeRecipient = lrtConfig.getContract(LRTConstants.PROTOCOL_TREASURY);
	    }
	    if (fee > 0) {
	      _transferAsset(asset, feeRecipient, fee);
	      emit InstantWithdrawalFeeCollected(msg.sender, asset, fee);
	    }
	function setInstantWithdrawalFeeRecipient(address feeRecipient) external onlyLRTManager {
	    UtilLib.checkNonZeroAddress(feeRecipient);
	    instantWithdrawalFeeRecipient = feeRecipient;
	    emit InstantWithdrawalFeeRecipientUpdated(feeRecipient);
	  }

Recommendations	Consider removing the checkNonZeroAddress check from setInstantWithdrawalFeeRecipient. Alternatively if the fee recipient will be set to the treasury address, consider documenting this in comments for clarity.

Comments / Resolution	Fixed. Added documentation

Issue_09	Disabling AAVE integration can be griefed
Severity	**Informational**

Description	AAVE integration can be enabled/disabled using the setAaveIntegrationEnabled() function. When changing the current AAVE pool due to current pool being paused the AAVE integration would be disabled , this operation can be griefed by an attacker by donating 1 wei of aaveAWETH which would trigger a withdrawal →
	
	if (!enabled) {
	// Withdraw all ETH from Aave back to contract
	uint256 aaveBalance = aaveAWETH.balanceOf(address(this));
	if (aaveBalance > 0) {
	_withdrawFromAave(aaveBalance);
	}
	}
	And since AAVE pool is paused the call would revert blocking the disable mechanism. This can also happen naturally without the griefing vector where disabling on kelp will fail while funds are on Aave and Aave is paused.

Recommendations	Consider acknowledging the issue
Comments / Resolution	Acknowledged.

NodeDelegator

NodeDelegator is a contract that manages individual delegation of restaked assets to EigenLayer operators, with each instance representing a separate operator delegation. Used by LRTDepositPool to distribute assets across multiple EigenLayer operators for diversification. The diff removes legacy nonce-based withdrawal tracking logic and transitions asset transfer functions from OPERATOR_ROLE to the new ASSET_TRANSFER_ROLE for improved permission granularity.

The removal of lastNonce checks from completeUnstaking and getAssetUnstaking eliminates legacy pre-slashing withdrawal handling code that is no longer needed. The access control change for transferBackToLRTDepositPool and transferETHToLRTUnstakingVault now requires ASSET_TRANSFER_ROLE instead of OPERATOR_ROLE, separating fund movement privileges from general operational tasks. The pause function now requires PAUSER_ROLE instead of MANAGER role, aligning with protocol-wide emergency response controls. The completeUnstaking function now directly calls decreaseUncompletedWithdrawalCount without conditional nonce-based logic.

Privileged Functions

- transferBackToLRTDepositPool
- transferETHToLRTUnstakingVault
- pause

Core Invariants:

INV 1: Only ASSET_TRANSFER_ROLE can transfer assets back to the deposit pool or unstaking vault.

RSETH

RSETH is an upgradeable ERC20 token representing shares in the Kelp DAO liquid restaking protocol, minted when users deposit LSTs and burned during withdrawals. Used by LRTDepositPool for minting on deposits and by LRTWithdrawalManager for burning during withdrawal processing. The diff introduces comprehensive emergency response mechanisms including transfer blocking, fund recovery, permanent exemptions, and enhanced daily mint limit tracking with period alignment.

The new transfer blocking system allows MANAGER role to freeze rsETH transfers from specific addresses for 24-hour periods, with the ability to recover frozen funds to a custody address while blocks are active. Permanent exemptions can be added for protocol contracts that should never be blocked. The reinitialize function sets the custody address and aligns the period start time within the last 24 hours for accurate daily limit tracking. New view functions provide transparency into remaining daily mint limits and next reset timestamps. The pause function access changed from MANAGER to PAUSER_ROLE for consistency with protocol-wide pause mechanics, and getCurrentPeriodStartTime ensures period alignment accounting for skipped days.

Privileged Functions

- addPermanentExemptions
- blockUserTransfers
- setCustodyAddress
- recoverFrozenFunds

Core Invariants:

INV 1: Custody address must be non-zero when set or during fund recovery.
INV 2: Transfers from blocked addresses revert until block expires or funds are recovered.
INV 3: Permanently exempt addresses can never have their transfers blocked.
INV 4: Only MANAGER role can add permanent exemptions which cannot be reversed.
INV 5: Only ADMIN can recover frozen funds while the transfer block is active.
INV 6: Only ADMIN can set the custody address for recovered funds.

Issue_10	LRTManager can pause minting by setting maxMintAmountPerDay to 0
Severity	**Governance**
Description	The LRTManager role can pause rsETH minting by setting maxMintAmountPerDay to 0. This is an issue since the LRTManager role should not have pausing capabilities in the case it's ever compromised.
Recommendations	Consider setting a minimum maxMintAmountPerDay as a safeguard.
Comments / Resolution	Acknowledged.

Issue_11	Minting to blocked addresses bypasses blocks
Severity	**Low**
Description	The _transfer function does not enforce transfer blocks when minting rsETH tokens, allowing blocked addresses to receive newly minted tokens. The transfer block enforcement only checks the from address to prevent outbound transfers from blocked users, but does not validate the to address during mint operations where from equals address zero. If unauthorized minting occurs due to a compromised minter role or system malfunction, the protocol blocking mechanism fails to prevent token issuance to blocked addresses. While blocked recipients cannot transfer these tokens elsewhere, the unbacked minting creates downstream accounting issues. The rsETH supply increases without corresponding asset backing, distorting the rsETHPrice calculation and potentially enabling exploitation during the period between detection and remediation.
Recommendations	Add a transfer block check for the to address in _transfer when from is address zero to prevent minting to blocked addresses. Alternatively block all transfers to blocked addresses.
Comments / Resolution	Fixed by following recommendation.

Issue_12	checkDailyMintLimit() logic allows larger mints in short periods
Severity	Low
Description	checkDailyMintLimit in LRTOracle and RSETH is supposed to check mint limits within a period where a period is a day long but the logic allows for the max to be exceed from the perspective of an arbitrary 24 hour period., consider the following case → 1.) Assume current blocktimestamp is X and on initialize periodStart = X 2.) After 3.9 days the modifier is triggered and a large mint occurs. 3.) As soon as 0.1 day has elapsed then the period can be shifted again since blockTimestamp > periodStart + 1 day (because periodStart points to X + 3 and now we are at X + 4) 4.) This means just after 0.1 day the max mint amount can be minted again. Resulting in more than the max mint amount being minted in a short period of time.
Recommendations	Consider acknowledging and keeping such cases in mind when configuring the max daily limit. Where a large mint can happen at the end of an epoch and at the beginning of the next.
Comments / Resolution	Acknowledged.

Issue_13	Blocked users can become exempt
Severity	**Informational**
Description	The addPermanentExemptions function allows managers to mark addresses as permanently exempt without checking if those addresses currently have active transfer blocks. When an address is added to the permanent exemption list, the _transfer function skips all block enforcement checks due to the isPermanentlyExempt condition. This means a user with an active transfer block can immediately bypass their restriction if granted permanent exemption status, without any delay or explicit acknowledgment that an active block is being overridden. While this requires manager action and represents an unlikely operational scenario, it creates an inconsistency where blocked addresses can gain immediate transfer capability through exemption rather than having the block explicitly cleared or expired first.
Recommendations	Consider adding a check in addPermanentExemptions to verify whether addresses currently have active transfer blocks before granting permanent exemption status.
Comments / Resolution	Fixed by following recommendation.

Issue_14	Blocking reverts on single exempted address
Severity	**Informational**
Description	The blockUserTransfers function reverts the entire transaction if any address in the batch is permanently exempt or zero address, preventing all other addresses from being blocked. In time-sensitive scenarios where managers need to quickly block multiple addresses to prevent fund movement, this all-or-nothing behavior creates operational risk. If an operator mistakenly includes one permanently exempt address or zero address in a batch of accounts that need immediate blocking, none of the addresses get blocked, requiring the operator to identify the problematic address, rebuild the list, and resubmit the transaction.
Recommendations	Skip addresses that are permanently exempt or zero address instead of reverting, and emit events for skipped addresses to maintain traceability.
Comments / Resolution	Fixed by skipping exempt and zero addresses instead of reverting.

Issue_15	Redundant `periodStartTime == 0` check
Severity	**Informational**
Description	`checkDailyLimit` and `getNnextDailyLimitResetTimestamp` both include a `periodStartTime == 0` check. However, we can see that the periodStartTime is already set in the reinitializer and there's no way to set/reset after that, which means the check doesn't actually do anything.
Recommendations	Consider removing the check.
Comments / Resolution	Fixed by following recommendation.

Issue_16	`blockUserTransfer` can be front-run to rescue funds
Severity	**Informational**
Description	The `blockUserTransfer` function is used to temporarily freeze rsETH tokens in an address by preventing transfers from the target address for 24 hrs. The issue is a malicious actor can simply front-run the blockUserTransfer call to move their tokens to a different address each time. Making it difficult for their tokens to be frozen/locked.
Recommendations	Consider acknowledging the issue.
Comments / Resolution	Acknowledged.

LRTConfigRoleChecker

LRTConfigRoleChecker is an abstract base contract providing role-based access control modifiers inherited by all major protocol contracts. This contract bridges role checks with the centralized LRTConfig access control system, enabling consistent permission enforcement across LRTDepositPool, NodeDelegator, LRTWithdrawalManager, LRTOracle, and RSETH. The diff adds two new modifiers to support the newly introduced ASSET_TRANSFER_ROLE for granular control over fund movements.

The new onlyAssetTransferRole modifier restricts function access exclusively to addresses holding ASSET_TRANSFER_ROLE, while onlyAssetTransferOrOperatorRole allows either ASSET_TRANSFER_ROLE or OPERATOR_ROLE to execute. These modifiers enable separation of asset transfer privileges from general operational duties, allowing the protocol to delegate fund movement capabilities to specialized addresses without granting full operator permissions. All modifiers check msg.sender against roles stored in the LRTConfig contract, maintaining centralized role management.

Privileged Functions

none

Core Invariants:

INV 1: Only addresses with ASSET_TRANSFER_ROLE can execute functions protected by onlyAssetTransferRole modifier.
INV 2: Either ASSET_TRANSFER_ROLE or OPERATOR_ROLE holders can execute functions protected by onlyAssetTransferOrOperatorRole modifier.

LRTConstants

LRTConstants is a library contract providing constant values and helper functions for accessing protocol configuration. Used throughout the protocol as a single source of truth for role identifiers, contract keys, and common addresses. The diff introduces a new access control role for asset transfer operations, separating this privilege from the broader OPERATOR_ROLE.

The new ASSET_TRANSFER_ROLE constant defines a distinct permission level specifically for moving assets between protocol contracts. This role separation allows for more granular access control where asset transfer operations can be delegated independently from other operational duties. The library continues to provide helper functions that wrap LRTConfig contract lookups, enabling cleaner code throughout the protocol when accessing configured contract addresses.

Privileged Functions

none

Core Invariants:

INV 1: The ASSET_TRANSFER_ROLE is a distinct role separate from OPERATOR_ROLE for moving assets between contracts.