BAIL security

Kelp
RSETH

FINAL REPORT

September ‘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 - RSETH
Website	kerneldao.com
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/Kelp-DAO/KelpDAO-contracts/tree/b84996691aab617bcdffa663d75815b0f7038750
Resolution 1	https://github.com/Kelp-DAO/KelpDAO-contracts/tree/436dbf8ce7270df8e6ba899c74da86d80165ea45/contracts
Resolution 2	https://github.com/Kelp-DAO/KelpDAO-contracts/commit/015d94c768391e7c1ee921a90035c50a50b1855d LRTUnstakingVault: https://www.diffchecker.com/vrQXIUnC/ LRTWithdrawalManager: https://www.diffchecker.com/gQ5TIQU6/ (only comments) NodeDelegator: https://www.diffchecker.com/0xpt7ncy/

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High	3	1		2		
Medium	13	3		10		
Low	13	5		8		
Informational	23	13		10		
Governance	1			1		
Total	53	22		31		

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

2. Detection

FeeReceiver

The FeeReceiver does MEV/execution-layer rewards collection and distribution to the deposit pool. Funds can be sent to the deposit pool by any address, as the sendFunds function is permissionless.

Core Invariants:

INV 1: Balance must decrease by full amount when sendFunds is called
INV 2: DepositPool balance must increase by exact amount when sendFunds is called

Privileged Functions

- initialize
- setDepositPool

Issue_01	Missing event emission in setDepositPool
Severity	**Informational**
Description	If the depositPool variable is updated in the FeeReceiver contract, an event is not emitted to notify offchain monitoring tools. Since the contract is intended to hold and redirect reward tokens received from Eigenlayer to the deposit pool, it could be misused if such updates are unnoticed.
Recommendations	Consider emitting an event whenever the deposit pool is updated.
Comments / Resolution	Fixed by following recommendations.

LRTConfig

The LRTConfig does protocol configuration management, supported asset tracking, and strategy management for the LRT protocol. The contract is responsible for setting state and limits for the protocol as a whole. Contracts can query the LRTConfig contract to retrieve contract addresses, fees, and supported assets.

Core Invariants:

INV 1: Cannot remove asset if deposits exceed maxNegligibleAmount
INV 2: Cannot update strategy if NDCs have non-zero balance

Privileged Functions

- initialize
- removeSupportedAsset
- addNewSupportedAsset
- updateAssetDepositLimit
- updateAssetStrategy
- setProtocolFeeBps
- setEigenLayerRewardReceiver
- setRSETH
- setToken
- setContract
- setMaxNegligibleAmount

Issue_02	tokenMap not cleared on supported asset removal
Severity	**Low**
Description	Function removeSupportedAsset in LRTConfig does not clear the tokenMap mapping for the asset being removed. Due to this, if stETH is removed as a supported asset, stakeEthForStETH will still work since getLSTToken will return the stETH token address.
Recommendations	Consider clearing the tokenMap mapping on supported asset removal.
Comments / Resolution	Acknowledged.

Issue_03	Consider using maxNegligibleAmount for asset strategy updates and node delegator removals
Severity	**Informational**
Description	Asset strategy updates in LRTConfig and node delegator contract removals both utilize strict zero value checks. In the event there is at least 1 wei asset token that cannot be withdrawn from the node delegator contract or the asset strategy, strategy updates and node delegator removals will not be possible. *if (ndcBalance > 0) {* *revert CannotUpdateStrategyAsItHasFundsNDCFunds(ndcs[i], ndcBalance);* *}* *INodeDelegator(nodeDelegatorAddress).stakedButUnverifiedNativeETH() > 0* *||* *INodeDelegator(nodeDelegatorAddress).getEffectivePodShares() != 0*
Recommendations	Consider replacing strict zero value checks with maxNegligibleAmount to leave some buffer for dust tokens.
Comments / Resolution	Acknowledged.

LRTDepositPool

The LRTDepositPool does LST asset deposits management, ETH staking, and fund distribution across node delegators. The LRTDepositPool also includes the function getTotalAssetDeposits, which is used throughout the protocol for price calculation and withdrawal checks. The LRTDepositPool also exposes functions which are used across the protocol to calculate the price of assets compared to eth and vice versa.

Appendix: Internal Swaps

Operators can rebalance inventory inside the pool by swapping ETH for LST or LST for ETH at oracle rates. Each swap enforces a minToAssetAmout and available liquidity in the pool. If balances can't cover the return, the swap reverts.

Appendix: Deposits and Minting rsETH

Users deposit ETH or supported LSTs and receive rsETH immediately. The pool checks the minimum deposit size, current asset deposit limits, and the user's minimum rsETH expected. Minting uses oracle prices and the calculation can be observed as:

rsETH to mint = *assetAmount × assetPrice ÷ rsETHPrice*

Core Invariants:

INV 1: Total asset deposits must equal sum of assets across deposit pool, NDCs, EigenLayer, and unstaking vault
INV 2: Node delegator queue length must never exceed maxNodeDelegatorLimit
INV 3: Node delegator cannot be removed if it has non-negligible asset balance
INV 4: Deposit amount must be greater than minAmountToDeposit
INV 5: RSETH minted must be proportional to asset value deposited based on oracle price

Privileged Functions

- initialize
- swapETHForAssetWithinDepositPool
- transferAssetToNodeDelegator
- transferETHToNodeDelegator
- transferAssetToLRTUnstakingVault
- transferETHToLRTUnstakingVault
- setMaxNegligibleAmount
- setMinAmountToDeposit

- updateMaxNodeDelegatorLimit
- addNodeDelegatorContractToQueue
- removeNodeDelegatorContractFromQueue
- removeManyNodeDelegatorContractsFromQueue
- pause
- unpause
- maxApproveToLRTConverter
- stakeEthForStETH

Issue_04	transferAssetToNodeDelegator may be frontrun in order to force funds to be sent to incorrect delegator
Severity	**Governance**
Description	Due to how the index based approach in deciding which delegator is sent funds in the transferAssetToNodeDelegator function, this allows malicious admins to force the operator to send funds to a delegator the operator did not wish to send funds to. *function transferAssetToNodeDelegator(* *uint256 ndcIndex,* *address asset,* *uint256 amount* *)* *external* *nonReentrant* *onlyLRTOperator* *onlySupportedAsset(asset)* *{* *address nodeDelegator = nodeDelegatorQueue[ndcIndex];* *IERC20(asset).safeTransfer(nodeDelegator, amount);* *}* The transferAssetToNodeDelegator function can be observed above, we can see that the operator must input an index and cannot directly choose the address he wishes to send assets to. This allows for an admin to frontrun a call to this function with a removal of a node delegator such that the indexes are changed and thus the assets are sent to a different delegator than the operator intended.
Recommendations	Consider the following risk, if not acceptable, consider having delegator removals under a short timelock mechanism.
Comments / Resolution	Acknowledged.

Issue_05	getTotalAssetDeposits returns incorrect value for ETH
Severity	Medium
Description	the getTotalAssetDeposits function tries to calculate the total amount of a specific asset in the protocol. However in the case where the asset is ETH, the function will return an incorrect value due to a discrepancy in the LRTConverter. *ethLyingInConverter = ILRTConverter(lrtConverter).ethValueInWithdrawal();* We calculate the eth lying in the converter by querying ethValueInWithdrawal, this is where the problem begins. When we look at how this value is calculated, we can observe that the value is incremented by the current price of an asset at the time transferAssetFromDepositPool is called. However, the value is decremented when the asset is swapped for ETH, but the amount the value is decremented is subject to a new price of said asset compared to ETH. *returnAmount = lrtDepositPool.getSwapETHToAssetReturnAmount(asset, ethAmountSent);* where the return amount is used to decrement the value of ethValueInWithdrawal. This is a problem because at the time transferAssetFromDepositPool is called the value of the asset may be high but when it is swapped the price may then be lower, this allows for ethValueInWithdrawal to remain non zero even when there exists no assets or value in the LRTConverter. Additionally this can be triggered if there was a slash on stETH before the assets were swapped. Thus the LRTConverter will return a value that is not present in the protocol and this value is used in getTotalAssetDeposits. getTotalAssetDeposits is used in many places in the protocol which

	includes the price calculation of rsETH, and calculating whether there are enough assets in the protocol to process a withdrawal. An incorrect value will lead to an incorrect rsETH price and incorrect accounting when attempting to use funds for withdrawals. Given that each time assets are transferred and swapped may lead to ethValueInWithdrawal being inflated, over time the accounting errors will accumulate and cause a further deviance of the true rsETH price and accounting.
Recommendations	Consider utilizing the lower of the real time rate and the initial unstaked rate when calculating the eth value
Comments / Resolution	~~Failed Resolution.~~ ~~Because the ethValueInWithdrawal is increased based on the price at the time of funds entering the contract and the ethValueInWithdrawal is decreased by a different price when funds leave the contract there will still be drift between the true value and what ethValueInWithdrawal states.~~
Comments / Resolution 2	Acknowledged. The Kelp team mentioned that the LRTConverter is primarily to test as well as an access-restricted contract that has been used very rarely. Due to this explanation, we have decreased the severity from high to medium. It has to be noted that this issue remains existing and is thus a risk.

Issue_06	Incorrect residue checks allow removing node delegators with non-zero balances
Severity	Medium
Description	Functions _checkResidueEthBalance and _checkResidueLSTBalance check the token balance of the node delegator and staked balance from strategies. However, it does not check the assets that are currently undergoing unstaking. This means the node delegator contracts can be removed from the nodeDelegatorQueue even though they hold unstaking assets, which could either be withdrawn to the unstaking vault or re-added as shares as part of the completeUnstaking flow when receiveAsTokens = false.
Recommendations	It is recommended to check assets unstaking as well.
Comments / Resolution	Fixed by following recommendations.

Issue_07	stETH 1-2 wei corner case breaks composability across codebase
Severity	**Medium**
Description	As per LRTConfig, stETH is intended to be a supported asset. When stETH is deposited in the LRTDepositPool using function depositAsset, the function calculates the exchange rate using the user input parameter and then transfers in the stETH. However, due to the stETH's 1-2 wei corner case in the balanceOf and transfer functionality, the contract mints slightly extra rsETH to the user since the actual stETH received by it is lesser than the parameter value. Similar instances can be found across the codebase that impact the following functionality: - depositAssetForL1Vault in L1VaultV2 - transferAssetToNodeDelegator, transferAssetToLRTUnstakingVault, getAssetDistributionData in LRTDepositPool - redeem, transferAssetToNodeDelegator, balanceOf in LRTUnstakingVault - instantWithdrawal in LRTWithdrawalManager - depositAssetIntoStrategy, completeUnstaking, getBalances, transferBackToLRTDepositPool in NodeDelegator - deposit, bridgeTokens in RSETHPoolV3ExternalBridge
Recommendations	It is recommended to implement support across the codebase for stETH.
Comments / Resolution	Acknowledged.

Issue_08	minAmountToDeposit is used for all tokens
Severity	Low
Description	In LRTDepositPool::_beforeDeposit the minAmountToDeposit is used to compare the deposited amount for all tokens. The issue with this is that the different supported tokens have different prices, so using the same amount is incorrect since the value will be different for each token
Recommendations	Have a minAmountToDeposit per token or minAmountToDeposit should be converted to deposit token before check.
Comments / Resolution	Acknowledged.

Issue_09	Missing check for ETH as supported asset in multiple instances across codebase
Severity	**Informational**
Description	Function depositETH does not check if ETH is a supported asset in LRTConfig. While it is unlikely for ETH to be removed, it is recommended to add the modifier to the function. Similar instances across the codebase: - transferETHToNodeDelegator in LRTUnstakingVault - transferETHToLRTUnstakingVault in NodeDelegator
Recommendations	It is recommended to add the modifier on depositETH and the necessary verification on other instances in the codebase.
Comments / Resolution	~~Failed Resolution.~~ ~~transferETHToNodeDelegator & transferETHToLRTUnstakingVault functions in the LRTUnstakingVault & NodeDelegator do not have the modifier.~~
Comments / Resolution 2	Fixed by following recommendation.

Issue_10	Remove redundant stakedButUnverifiedNativeETH condition from _checkResidueEthBalance
Severity	**Informational**
Description	Function _checkResidueEthBalance checks whether the stakedButUnverifiedNativeETH value is zero before removing the node delegator contract. However, the stakedButUnverifiedNativeETH value is already included as part of the value returned by getEffectivePodShares. *function _checkResidueEthBalance(address nodeDelegatorAddress) internal view {* *if (* *INodeDelegator(nodeDelegatorAddress).stakedButUnverifiedNativeETH() > 0* *||* *INodeDelegator(nodeDelegatorAddress).getEffectivePodShares() != 0* *|| address(nodeDelegatorAddress).balance > maxNegligibleAmount* *) {* *revert NodeDelegatorHasETH();* *}* *}*
Recommendations	Consider removing the first condition.
Comments / Resolution	Fixed by following recommendations.

Issue_11	Missing functionality to revoke supported asset approvals from LRTDepositPool to LRTConverter
Severity	**Informational**
Description	Function maxApproveToLRTConverter performs an approval from the deposit pool to the converter contract. The approval can only be made for supported tokens. However, if a supported token is removed from the LRTConfig contract, the approval still remains. While this does not seem to pose a risk, it is recommended to introduce functionality to revoke approvals of unsupported assets.
Recommendations	Consider introducing functionality to revoke unsupported token approvals.
Comments / Resolution	~~Failed Resolution.~~ ~~The ability to revoke approvals is limited to only supported tokens.~~ ~~An Unsupported tokens allowance cannot be revoked.~~
Comments / Resolution 2	Acknowledged. The Kelp team mentioned that Token approvals can be revoked prior to support being removed.

Issue_12	Misleading NodeDelegatorAddedinQueue event
Severity	**Informational**
Description	The addNodeDelegatorContractToQueue function emits NodeDelegatorAddedinQueue with all input node delegators, even though it skips existing ones. This creates misleading event logs as observers cannot distinguish which delegators were actually added versus which were already registered.
Recommendations	Track and emit only newly added node delegators, or rename events for clarity.
Comments / Resolution	Fixed by following recommendations.

LRTOracle

The LRTOracle does price calculations for assets, manages RSETH exchange rates, and handles protocol fee calculations. The LRTOracle allows anyone to update the rsETH price as the function is permissionless. The contract ensures that the price remains within an acceptable percentage and any deviation requires intervention by the Manager role. The price calculation is derived by dividing the total eth in the protocol by the rsETH supply. The treasury is minted a portion of the increase in TVL.

Appendix: Price update flow

The price update begins when updateRSETHPrice or updateRSETHPriceAsManager is called. The second function is reserved for LRT managers and can bypass some price threshold restrictions.

1. **Total ETH Calculation:**
   The oracle computes the total ETH value across all supported assets by:
   - Fetching asset prices from their registered IPriceFetcher oracles
   - Multiplying each asset's total deposits in LRTDepositPool by its ETH rate
2. **New Price Derivation:**
   The new rsETH price is calculated as:

   (totalETHInProtocol - protocolFeeInETH) / totalRsethSupply

The protocol fee is charged only when the ETH value increases and the contract is not paused

Core Invariants:

INV 1: Protocol fee must not exceed TVL increase
INV 2: Daily fee minting must not exceed maxFeeMintAmountPerDay
INV 3: Price increase must not exceed pricePercentageLimit without manager approval
INV 4: Protocol must pause if price decrease exceeds pricePercentageLimit

Privileged Functions

- initialize
- updateRSETHPriceAsManager
- updatePriceOracleForValidated
- updatePriceOracleFor
- setPricePercentageLimit
- setMaxFeeMintAmountPerDay
- pause
- unpause

Issue_13	Daily fee limit blocks price updates
Severity	**High**
Description	LRTOracle's _updateRsETHPrice reverts when the daily fee minting limit is reached, preventing price updates. This forces stale prices to persist, allowing users to deposit at incorrect rates. Depositors can profit from outdated lower prices creating systemic arbitrage opportunities at the expense of other protocol users.
Recommendations	Skip fee minting when the limit is reached, but continue with price updates. Recalculate rsETHPrice and highestRsethPrice without fee component.
Comments / Resolution	Acknowledged.

Issue_14	A whale can entirely brick the profit logging for rsETH for the entire day
Severity	**Medium**
Description	Consider the following → 1.] A whale deposits enough assets in the LRTDepositPool which would mint rsETH to consume the entire currentPeriodMintedAmount in RSETH.sol → *modifier checkDailyMintLimit(uint256 amount) {* *// Check if we need to reset the period or initialize the period* *if (periodStartTime == 0 || block.timestamp >= periodStartTime + 1 days) {* *currentPeriodMintedAmount = 0;* *periodStartTime = block.timestamp;* *}* *// Check if minting would exceed the daily limit* *if (currentPeriodMintedAmount + amount > maxMintAmountPerDay) {* *revert DailyMintLimitExceeded(currentPeriodMintedAmount + amount, maxMintAmountPerDay);* *}* *currentPeriodMintedAmount += amount;* *_;* *}* 2.] Now the whale can instantly initiate a withdrawal (optional , not necessary) , and this would lead to 2 major issues → a.) Profits made from EigenLayer strategies can not be logged now for this entire day because when _updateRsETHPrice is invoked and there are profits made then rsethAmountToMintAsProtocolFee amount of rsETH is minted to the treasury , but since any new mints are blocked on RSETH.sol due to step 1.) now this tx would revert and the

	price of rsETH would not be updated. b.) No new deposits can be made for the entire day , deposits are bricked now (since mint in RSETH reverts).
Recommendations	Whitelist the treasury so that when fees are minted to the treasury the daily limit check is ignored.
Comments / Resolution	Acknowledged.

Issue_15	Missing staleness verification in getAssetPrice
Severity	**Medium**
Description	The function getAssetPrice retrieves prices from various oracle implementations (e.g. ChainlinkPriceOracle). However, none of the oracles implement staleness verification on the prices returned. *function getAssetPrice(address asset) public view onlySupportedOracle(asset) returns (uint256) {* *return IPriceFetcher(assetPriceOracle[asset]).getAssetPrice(asset);* *}* If stale prices are returned, these could be consumed by rsETH price updates and other functionality from LRTDepositPool relying on valid price data.
Recommendations	Consider implementing staleness verification thresholds to ensure prices are not stale.
Comments / Resolution	Acknowledged.

Issue_16	Storage collision in LRTOracle upgrade
Severity	**Medium**
Description	The LRTOracle upgrade replaces two legacy state variables (legacy_cooldownPeriodInTimestamp, legacy_lastUpdatedCoolDownTimestamp) with three new ones (currentPeriodMintedFeeAmount, feePeriodStartTime, maxFeeMintAmountPerDay). The first two new variables will reuse the storage slots of the legacy variables, causing their values to be reinterpreted. This will corrupt the protocol state as the legacy timestamps will be misinterpreted as fee amounts and period times.
Recommendations	Add new variables at the end of storage layout instead of reusing legacy slots.
Comments / Resolution	Acknowledged.

Issue_17	1e18 oracle price-feed check in updatePriceOracleForValidated can still fail for 1e18 price-feeds
Severity	**Low**
Description	The updatePriceOracleForValidated functions validates that a price feed is in 18 decimals using the following check *//sanity check that oracle has 1e18 precision* *uint256 price = IPriceFetcher(priceOracle).getAssetPrice(asset);* *if (price > 1e19 || price < 1e18) {* *revert InvalidPriceOracle();* *}* The issue is that asset prices can be below 1e18 even for 1e18 price-feeds. For example current stETH price is 0.998ETH which is 9.98e17 in 18 decimals
Recommendations	Consider reducing the lower bound to ~1e15.
Comments / Resolution	Fixed by following recommendations.

Issue_18	It is not possible to mark assets as not supported
Severity	**Informational**
Description	In the Oracle, an asset is considered supported if assetPriceOracle mapping is non zero, this can be confirmed by the following logic which. *modifier onlySupportedOracle(address asset) {* *if (assetPriceOracle[asset] == address(0)) {* *revert AssetOracleNotSupported();* *}* *_;* However currently in the oracle we cannot set the oracle address for an asset back to zero due to a zero value check when setting the value. *function updatePriceOracleFor(address asset, address priceOracle) public onlyLRTAdmin {* *UtilLib.checkNonZeroAddress(priceOracle);* *assetPriceOracle[asset] = priceOracle;* *emit AssetPriceOracleUpdate(asset, priceOracle);* *}* As we can observe it is not possible to remove an oracle for an asset and thus it is not possible to mark an asset as unsupported in the oracle.
Recommendations	Consider allowing the value to set to 0 or simply acknowledge the issue if this is the intended design.
Comments / Resolution	Fixed by following recommendations.

Issue_19	Protocol fee calculations round down against protocol
Severity	**Informational**
Description	The protocol fee calculations across multiple contracts consistently round down due to integer division, resulting in the protocol receiving slightly less revenue than expected. This occurs in fee calculations for rewards, deposits, and withdrawals. In the LRTOracle contract, protocol fees from rewards are calculated using `protocolFeeInETH = (rewardAmount * lrtConfig.protocolFeeInBPS()) / 10_000`, Where the division happens after multiplication, causing downward rounding. Similar patterns exist in RSETHPool for deposit fees and LRTWithdrawalManager for instant withdrawal fees.
Recommendations	Modify all fee calculations to round up to ensure protocol receives expected fees without shortfall.
Comments / Resolution	Acknowledged.

Issue_20	Price events emit before validation completes
Severity	**Informational**
Description	In LRTOracle's _updateRsETHPrice function, RsETHPriceDecrease and RsETHPriceBelowPeak events are emitted before validating if the price decrease is within limits. If the decrease exceeds limits, the protocol pauses and returns without updating the price, making these events misleading as they announce a price change that never occurred.
Recommendations	Move price-related event emissions after the isPriceDecreaseOffLimit check to ensure events reflect actual price changes.
Comments / Resolution	Fixed by following recommendations.

LRTUnstakingVault

The LRTUnstakingVault does asset unstaking management, withdrawal tracking, and handles asset transfers between node delegators. The contract includes a redeem function which is to be called by the WithdrawalManager contract when withdrawals are completed. Ultimately funds will be stored in the LRTUnstakingVault in order to cover both instant and normal withdrawals.

Core Invariants:

INV 1: Asset transfers must only go to valid node delegators in the queue
INV 2: ETH transfers must only go to valid node delegators in the queue
INV 3: sharesUnstaking can only be reduced, never increased beyond what was originally unstaking

Privileged Functions

- initialize
- redeem
- transferAssetToNodeDelegator
- transferETHToNodeDelegator
- reduceSharesUnstaking
- setMaxUncompletedWithdrawalCount
- increaseUncompletedWithdrawalCount
- decreaseUncompletedWithdrawalCount

Issue_21	Pause functionality is never utilized
Severity	**Informational**
Description	LRTUnstakingVault inherits PausableUpgradeable but never uses it in the contract on any of its functions.
Recommendations	Consider acknowledging the issue.
Comments / Resolution	Acknowledged.

LRTWithdrawalManager

The LRTWithdrawalManager does withdrawal request management, queue processing, and handles asset unlocking for users converting rsETH back to LSTs. Users may choose to instantly withdraw their funds if enabled, however this will incur a fee. The other option will see users be added to a queue and the operator will then unlock assets in the queue. After this users can complete their withdrawal.

Appendix: instantWithdrawal

The instant withdrawal lets users swap rsETH for an asset like ETH or LST right away instead of waiting in the queue. When a user calls instantWithdrawal, the contract burns their rsETH, redeems the matching asset amount from the unstaking vault, takes a small fee, and sends the rest to the user. The feature must be enabled for that asset and enough liquidity must be available in the vault. It's faster than the regular withdrawal flow but costs more due to the instant withdrawal fee.

Core Invariants:

INV 1: User cannot complete withdrawal before delay period
INV 2: Cannot unlock requests that are already unlocked

Privileged Functions

- initialize
- unlockQueue
- setMinRsEthAmountToWithdraw
- setWithdrawalDelayBlocks
- pause
- unpause
- setInstantWithdrawalEnabled
- setInstantWithdrawalFee

Issue_22	Stale price enables yield extraction attack
Severity	**High**
Description	Key protocol functions use stale rsETH prices by not updating them before calculations. When share price increases exceed instant withdrawal fees, attackers can profit through: deposit at stale low price → force price update → instant withdrawal at new high price. This cycle can be repeated whenever price deviation creates arbitrage opportunities. Affected functions: depositETH, depositAsset, initiateWithdrawal, unlockQueue, and depositETHForL1VaultETH.
Recommendations	Update rsETH price at start of affected functions. Ensure price updates cannot revert for non-invariant breaking reasons to ensure functions are accessible.
Comments / Resolution	Acknowledged.

Issue_23	sweepRemainingAssets may be DOSed indefinitely
Severity	**Medium**
Description	If an asset has any unlocked withdrawal, then calling sweepRemainingAssets will revert until the final withdrawal has been finalized. In the case where a user decides to grief and not finalize the withdrawal, the function completeWithdrawalForUser was introduced to allow the operator to force a withdrawal on the user. However in the case where the asset in question is ETH, a user can cause the transfer of ETH to his contract to fail in order to DOS sweepRemainingAssets for ETH.
Recommendations	Consider a try catch solution that will wrap ETH and send WETH in the case the eth call fails. Also document this behavior so that users who interact with contracts are aware.
Comments / Resolution	Acknowledged. The Kelp team mentioned that ETH should not accumulate in a way that requires sweeping. However documentation was added to acknowledge such a case.

Issue_24	User can hedge his losses against slashing with an instant withdraw
Severity	**Medium**
Description	A user can anticipate a slashing event on EigenLayer (precisely when the AVS calls slashOperator() on the AllocationManager), this would decrease the price of rsETH in the LRTOracle. The user can frontrun the updation of rsETH price and invoke the instantWithdrawal() function in the LRTWithdrawalManager, this would calculate the assetAmountUnlocked based on the pre-slashed rsETH price and the user would just need to pay a hardcoded fractional fee(only requirement is that the LRTUnstakingVault should have enough funds, and this can be true by previous withdrawals by the NodeDelegator which has not been unlocked yet by unlockQueue() or can be surplus assets sent by the LRTDepositPool). The asset amount received by the user after fee deduction can be much higher that what the user would have received after the slashing event. Due to this the rest of the users would have to bear the entire losses.
Recommendations	The issue seems to stem from design choice and should be acknowledged.
Comments / Resolution	Acknowledged.

Issue_25	Instant withdrawals can brick withdrawal queue
Severity	**Medium**
Description	The LRTWithdrawalManager's instant withdrawal mechanism has an availability calculation mismatch. While getAvailableAssetAmount checks total protocol assets minus committed amounts, instant withdrawals only pull from LRTUnstakingVault. Take the following example as a demonstration of how an attacker can grief the withdrawal queue: 1. LRTUnstakingVault has a value of 100 ETH allocated for queued withdrawals [60,20,10,10] 2. Attacker flashloans a value of 41 ETH 3. Deposits into the DepositPool 4. Instantly withdraws via instantWithdrawal 5. Withdrawal pulls a value of 41 ETH from UnstakingVault 6. With a balance of 59 ETH remaining for first queued withdrawal (60 ETH in value) the withdrawal cannot occur 7. All subsequent withdrawals are blocked until more assets are unstaked The attack only costs the instant withdrawal fee and can be repeated to continuously block the withdrawal queue, forcing legitimate users to wait for new EigenLayer unstaking. The attacker does not need to withdrawal all funds just enough for the first request in the queue to not be met
Recommendations	Consider restricting instant withdrawals to only use funds from LRTDepositPool, not UnstakingVault. This ensures that the committed assets remain reserved for the withdrawal queue while allowing users to still access the instant withdrawal feature.
Comments / Resolution	~~Failed Resolution.~~ ~~Attack is still possible since there is a window between the first element in the queue being unlocked and the buffer being updated to reflect the next queue amount.~~ ~~In cases where the upcoming queue amount is greater than the~~

	~~first element this buffer would be insufficient and an attacker can front-run the operator to execute this attack resulting in the same outcome as the initial finding.~~
Comments / Resolution 2	Acknowledged. The Kelp team mentioned that the buffer was added to limit the amount of funds that can be withdrawn via instant withdrawal. This buffer can be modified to address any frontrunning concerns.

Issue_26	User can gas grief operator in completeWithdrawalForUser
Severity	**Low**
Description	CompleteWithdrawalForUser allows the operator to complete withdrawals on behalf of a user. If the asset to withdraw is eth, a low-level call is made to the user address. The issue is that a malicious user can gas grief the operator by a lot more gas than required since by default “call” forwards 63/64 of remaining gas
Recommendations	Specify the amount of gas to be forwarded with the call
Comments / Resolution	Acknowledged. The Kelp team mentioned that his function is not expected to be used for eth. However, documentation was added to acknowledge such a case.

Issue_27	Users could bear slashing even when withdraw delay has elapsed
Severity	**Low**
Description	Consider the following → 1.] X amount of users initiate a withdrawal in the LRTWithdrawalManager (initiateWithdrawal()). 2.] The NodeDelegator contract would then queue a withdrawal in EigenLayer where the withdrawal delay period is MIN_DELAY_WITHDRAW_BLOCKS. 3.] These withdrawal requests would be completed when the LRTOperator invokes the unlockQueue() function. 4.] Say another Y user also initiates a withdrawal. 5.] Since in the unlockQueue function the withdrawals are being done in chunks (where the limit is decided by firstExcludedIndex) , it is possible that there was slashing on Eigen Layer but this slashing occurred after the withdawDelayBlocks has elapsed i.e. slashing window has elapsed of the X users but unlockQueue() for these withdrawals is being done after this period, the X users would experience the slashed price of rsETH for their withdrawal. 6.] For the Y users the slashed amount was justified since they are still under their withdrawDelay but it is not justified for the X users.
Recommendations	Document that until the queue is unlocked funds being unstaked will be included in any potential slashing.
Comments / Resolution	Acknowledged.

Issue_28	unlockQueue() can be performed with incorrect rsETH price in extreme slashing emergency situation
Severity	**Low**
Description	Consider the following scenario → 1.) Some X amount of users have initiated a withdrawal in LRTWithdrawalManager. 2.) The operator in EigenLayer experienced some serious slashing i.e. something like 80% of the operator shares that were delegated got slashed and due to this the price of rsETH should drop significantly, but to protect users the _updateRsethPrice pauses the LRTWithdrawalManager and just returns without updating the rsETH price → *// if price decrease is off limit, pause the protocol (unless it's already paused)* *if (isPriceDecreaseOffLimit) {* *if (!lrtDepositPool.paused()) lrtDepositPool.pause();* *if (!withdrawalManager.paused())* *withdrawalManager.pause();* *_pause();* *return;* *}* 3.) Now, the LRTOperator does unlockQueue() and that function does not have a whenNotPaused modifier (though the function is privileged this might happen unintentionally by the LRTOperator) , and it calls _unlockWithdrawalRequests and would calculate the request's payout amount using the rsETH price which hasn't been modified due to the extreme slashing i.e. this is the wrong price to use(also keep in mind unstake vault should have these amounts but its fine because there can be enough assets there to satisfy for example just one request and if for next one the assets are insufficient the loop just breaks →

	while (nextLockedNonce_ < firstExcludedIndex) { *bytes32 requestId = getRequestId(asset, nextLockedNonce_);* *WithdrawalRequest storage request = withdrawalRequests[requestId];* *// Check that the withdrawal delay has passed since the request's initiation.* *if (block.number < request.withdrawalStartBlock + withdrawalDelayBlocks) break;* *// Calculate the amount user will recieve* *uint256 payoutAmount = _calculatePayoutAmount(request, rsETHPrice, assetPrice);* *if (availableAssetAmount < payoutAmount) break; // Exit if not enough assets to cover this request* Therefore , for these request/requests (depending on how many requests with this old highly inflated price the unstake vault can satisfy) the withdrawal request(s) is confirmed with the old price which can be very high despite extreme slashing. Now whenever the withdrawal manager is unpaused the user can call completeWithdrawal() with the incorrectly large request.expectedAssetAmount.
Recommendations	Have a whenNotPaused modifier on unlockQueue.
Comments / Resolution	~~Failed Resolution.~~ ~~The unlockQueue function does not check if the oracle contract is paused. Because of this, it can still proceed with a stale price.~~
Comments / Resolution 2	Acknowledged. The Kelp team mentioned that when a significant ETH price drop occurs, the system automatically pauses the LRTWithdrawalManager, which also disables unlockQueue. If the operator explicitly unpauses the LRTWithdrawalManager, the unlock functions should operate as intended. Since unlockQueue is an operator-only function, it can always be withheld from execution if conditions are unsafe.

Issue_29	Requests which have already been unlocked are subject to updated withdrawalDelayBlocks
Severity	**Low**
Description	The issue stems from the fact that requests do not have a fixed delay from when they are initiated, this is due to the fact that withdrawalDelayBlocks may be updated and thus cause the delay of an already initiated request to be longer. However this is the case even when a request has already been unlocked. In this state, rsETH has already been burned and assets redeemed. Thus the following check will force users to wait to complete a withdrawal when they shouldn't need to. *if (block.number < request.withdrawalStartBlock + withdrawalDelayBlocks) revert WithdrawalDelayNotPassed();* The check above is present in completeWithdrawal, this check is already made when the operator calls unlockQueue so adding the same check will simply cause delays of already confirmed withdrawals if the withdrawalDelayBlocks is increased before users are able to claim.
Recommendations	Consider removing the delay checks in completeWithdrawal since these have been checked when calling unlockQueue.
Comments / Resolution	Acknowledged.

Issue_30	No fee slippage enforced on instantWithdrawal
Severity	**Informational**
Description	A user might pay an unintentionally higher fee during instantWithdrawal in a scenario where there is a setInstantWithdrawalFee call(increasing the fee) just before the instantWithdrawal which can happen if a malicious validator queues this setInstantWithdrawalFee before the instantWithdrawal tx.
Recommendations	Consider acknowledging this risk and documentation should be added for the mentioned risk.
Comments / Resolution	Acknowledged.

Issue_31	Redundant struct parameter
Severity	**Informational**
Description	In unlockQueue the firstExcludedIndex is added to the UnlockParams struct, however the value is never read from the struct and is instead always read directly from the input parameter.
Recommendations	Consider removing firstExcludedIndex from UnlockParams
Comments / Resolution	Fixed by following recommendations.

Issue_32	withdrawalDelayBlocks in LRTWithdrawalManager is not compatible with Eigen Layer's MIN_WITHDRAWAL_DELAY_BLOCKS
Severity	**Informational**
Description	The withdrawalDelayBlocks in LRTWithdrawManager is initialized by number of blocks in 8 days and max capped to number of blocks in 10 days, but in Eigen Layer the withdrawal delay is the number of blocks in 14 days. Due to this discrepancy the withdrawal process in LRTWithdrawalManager might use up assets from unstaking vault unintentionally since those assets could be meant for NodeDelegator (which would be used for depositing into Eigen strategies and delegations) and not the assets received through a completeWithdrawal transaction from EigenLayer.
Recommendations	Consider using a value for withdrawalDelayBlocks in sync with MIN_WITHDRAWAL_DELAY_BLOCKS.
Comments / Resolution	Fixed by following recommendations.

NodeDelegator

The NodeDelegator does asset staking into EigenLayer strategies, validator management, and handles ETH/LST deposits and withdrawals. This contract is fully permissioned and only the operator or manager can make state changes.

Appendix: Staking/EigenPod

The NodeDelegator is responsible for restaking native ETH into EigenLayer. Operators call stake32Eth to deposit exactly 32 ETH per validator, registering its pubkey and sending the deposit through the EigenPodManager. The contract tracks ETH that has been staked but yet verified until proof of credential is submitted.

Core Invariants:

INV 1: Cannot stake ETH without registering pubkey
INV 2: Uncompleted withdrawal count must not exceed maximum

Privileged Functions

- initialize
- initialize2
- depositAssetIntoStrategy
- delegateTo
- createEigenPod
- stake32Eth
- stake32EthValidated
- processClaim
- verifyWithdrawalCredentials
- startCheckpoint
- undelegate
- initiateUnstaking
- completeUnstaking
- transferBackToLRTDepositPool
- transferETHToLRTUnstakingVault
- maxApproveToEigenStrategyManager
- revokeApprovalToEigenStrategyManager
- pause
- unpause

Issue_33	maxUncompletedWithdrawalCount would be incorrect if the Operator in EigenLayer undelegates himself
Severity	High
Description	The LRTUnstakingVault maintains a maxUncompletedWithdrawalCount , the purpose of this max cap is that the price calculation in LRTOracle includes assets that are unstaking / queued for withdrawal on EigenLayer. The price update loops over all queued withdrawals (_getTotalEthInProtocol() → getTotalAssetDeposits() → getAssetUnstaking()) to sum balances. If that list gets too long, the loop exceeds the block gas limit and the transaction reverts. Therefore , when undelegate() is invoked in the NodeDelegator we check if the undelegation would lead to the queued withdrawals being more or equal to the maxUncompletedWithdrawalCount → *function undelegate() external whenNotPaused onlyLRTManager {* *if (elOperatorDelegatedTo() == address(0)) {* *revert CantUndelegate();* *}* *bytes32[] memory withdrawalRoots = _getDelegationManager().undelegate(address(this));* *if (* *_getUnstakingVault().uncompletedWithdrawalCount() + withdrawalRoots.length* *> _getUnstakingVault().maxUncompletedWithdrawalCount()* *) {* *revert MaxUncompletedWithdrawalsReached();* *}* But, undelegate() can also be done directly from EigenLayer where msg.sender is Operator to which NodeDelegator is delegated , see here →

	https://github.com/Layr-Labs/eigenlayer-contracts/blob/main/src/contracts/core/DelegationManager.sol#L162 This is a feature so no malicious intent lies here, an Operator can for whatever reason undelegate and when undelegate() is triggered directly from Eigen Layer we are bypassing the maxUncompletedWithdrawalCount check since in this case the increaseUncompletedWithdrawalCount() would not be triggered and defeats the entire purpose of maxUncompletedWithdrawalCount. Due to this, the updateRSETHPrice would be DoSed and impact would be high since price of rsETH would not be updated to the correct one.
Recommendations	Account for the undelegations done directly from EigenLayer too.
Comments / Resolution	Fixed by adding setUncompletedWithdrawalCount which considers undelegations done directly on EigenLayer.

Issue_34	increaseUncompletedWithdrawalCount is not invoked in initiateUnstaking
Severity	Medium
Description	The withdrawal count is not being incremented in the initiateUnstaking function in the NodeDelegator contract, the withdrawal count should be incremented by one. The purpose of the max cap(maxUncompletedWithdrawalCount) is that the price calculation in LRTOracle includes assets that are unstaking / queued for withdrawal on EigenLayer. The price update loops over all queued withdrawals (_getTotalEthInProtocol → getTotalAssetDeposits → getAssetUnstaking) to sum balances. If that list gets too long, the loop exceeds the block gas limit and the transaction reverts. This update of withdrawal count is being done correctly in undelegate where increaseUncompletedWithdrawalCount is being called for each strategy exit .
Recommendations	increaseUncompletedWithdrawalCount should be called within initiateUnstaking.
Comments / Resolution	Fixed by following recommendations.

Issue_35	Operator from eigen layer can steal all Kelp's user funds
Severity	**Low**
Description	According to the new Eigen Layer's update there can be redistributing operator sets , where the slashed assets instead of getting burnt are sent to an address assigned by the AVS, meaning the Operator to which Kelp is delegated might be registered to a redistributing operator set where the redistribution recipient can be anyone AVS wishes to be → https://github.com/Layr-Labs/eigenlayer-contracts/blob/main/src/contracts/core/AllocationManager.sol#L282 when slashed → https://github.com/Layr-Labs/eigenlayer-contracts/blob/main/src/contracts/core/DelegationManager.sol#L320 and then claim from here → https://github.com/Layr-Labs/eigenlayer-contracts/blob/main/src/contracts/core/StrategyManager.sol#L172 Essentially stealing all user funds from Kelp.
Recommendations	Acknowledge the risks of such operator sets and operators.
Comments / Resolution	Acknowledged.

Issue_36	Redelegate functionality from EigenLayer not implemented in NodeDelegator
Severity	**Informational**
Description	The current implementation of NodeDelegator does not have a redelegate functionality (leveraging the redelegate() functionality from EigenLayer's DelegationManager contract). The LRTManager would need to firstly undelegate() and then delegate to a new Operator using the delegateTo() function to perform a redelegation.
Recommendations	Implement the redelegate functionality for smooth redelegations.
Comments / Resolution	Acknowledged, redelegations will be added at a later date.

Issue_37	No functionality to consolidate validators and request partial withdrawals from eigen pod
Severity	**Informational**
Description	With the new ethereum Pectra upgrade validators can be consolidated (EIP-7251) and EigenPod supports this by introducing a requestConsolidation function where validators can be consolidated and partial withdrawals can be requested via requestWithdrawal. But currently Kelp does not support these which limits the functionality offered.
Recommendations	Consider acknowledging the current design of the node delegator.
Comments / Resolution	Acknowledged, partial withdrawals and consolidation of validators to be added at a later date.

Issue_38	Misplaced modifiers for transfer functions
Severity	**Informational**
Description	Functions in NodeDelegator: transferBackToLRTDepositPool(), transferETHToLRTUnstakingVault(), are said to be called only by LRTManager, but their modifier is LRTOperator.
Recommendations	Ensure natspec and implementation align.
Comments / Resolution	Fixed by following recommendations.

NodeDelegatorHelper

The NodeDelegatorHelper does helper functions for NodeDelegator contracts, providing utility functions for asset balance calculations and strategy management.

RSETH

The RSETH does ERC20 token management for the rsETH token, including minting, burning, and daily mint limits. It is the token that users are minted on deposits and the token that is burned on withdrawals.

Core Invariants:

INV 1: Only authorized roles can mint and burn
INV 2: Period start time must not reset before 24 hours
INV 3: Cannot mint or burn when paused

Privileged Functions

- initialize
- setMaxMintAmountPerDay
- mint
- burnFrom
- pause
- unpause

Issue_39	RsETH is not compatible with CCIP self service registration
Severity	**Medium**
Description	CCIP tokens require a token pool where the tokens are locked/burnt or released/minted upon bridging. Only the token administrator can set this pool. The issue is that in order to set the token administrator the token must either be ownable or accessControl with defaultAdmin or have the getCCIPAdmin function as seen here, however, current implementation of RsETH does not support any of these However, although not including these functions means that rsETH can not be registered via self service, the docs do say that even if these functions are not included, contacting the chainlink team can help with registration.
Recommendations	Consider including a getCCIPAdmin/owner function or acknowledging the issue if self service registration is not required.
Comments / Resolution	Acknowledged.

L1VaultV2

The L1VaultV2 does cross-chain bridging of rsETH tokens between L1 and L2, handling deposits and token wrapping/unwrapping. The bridge has 2 possible types, CCIP or LayerZero, however it cannot be both types at once. The contract does allow the bridge to switch between bridge types.

Core Invariants:

INV 1: Bridge type must be either LayerZero or CCIP
INV 2: Cannot bridge more rsETH than balance
INV 3: Native fee must cover bridge costs

Privileged Functions

- initialize
- reinitialize
- depositETHForL1VaultETH
- depositAssetForL1Vault
- unwrapWstETH
- unwrapWETH
- bridgeRsETHToL2
- bridgeRsETHToL2UsingCCIP
- setLrtDepositPool
- setRsETH
- setOFTAdapter
- setDstLzChainId
- setL2Receiver
- setDescription
- setWstETH
- setCCIPRouter
- setDestinationChainSelector
- setBridgeType

Issue_40	Surplus msg.value is not refunded to caller
Severity	**Low**
Description	Function bridgeRsETHToL2 performs the following check to ensure enough native tokens are provided to perform bridging using LayerZero. *if (msg.value < nativeFee) {* *revert InsufficientNativeFee();* *}* However, any additional msg.value - nativeFee amount of tokens are not refunded back to the caller. A similar instance exists in bridgeAssets in RSETHPoolV3ExternalBridge.
Recommendations	It is recommended to refund any extra msg.value sent.
Comments / Resolution	Fixed by ensuring msg.value == nativeFee.

Issue_41	Gas limit is not included when bridging RsETH to L2
Severity	**Informational**
Description	When bridging RsETH to L2 via LayerZero with bridgeRsETHToL2 there's no gas limit encoded in the extraOption parameter of the SendParam struct, though a default of 200,000 gas is priced in the call for some chains it might not be enough(which can cause channel blocking) or might be too much.
Recommendations	Consider setting a gas limit when bridging RsETH.
Comments / Resolution	Acknowledged.

Issue_42	Excess token approval due to LayerZero decimals
Severity	**Informational**
Description	In the L1VaultV2 contract, the bridgeRsETHToL2 function approves the full amount of rsETH to the oftAdapter before bridging. However, LayerZero's OFT (Omnichain Fungible Token) implementation converts token amounts from local decimals (LD) to standard decimals (SD), which can result in dust amounts being removed from the transfer. The sequence causing the issue: 1. Contract approves full amount to oftAdapter 2. LayerZero converts amount to standard decimals, removing dust 3. Actual transfer amount is (amount - dust) 4. Difference between approved and transferred remains as residual allowance
Recommendations	Calculate the post-conversion amount before approval and only approve the exact amount that will be transferred by the bridge.
Comments / Resolution	Acknowledged.

Issue_43	Hardcoded CCIP gas limit risks future failures
Severity	**Informational**
Description	The L1VaultV2 contract hardcodes a CCIP gas limit of 200,000 in getCCIPMessage. Chainlink's documentation explicitly advises against fixed gas limits as CCIP execution costs may change with network conditions or protocol upgrades. Without the ability to adjust this value, cross-chain messages could fail if gas requirements increase, requiring a contract upgrade to fix.
Recommendations	Consider adding a configurable gas limit that is adjustable by the admin to adapt to CCIP changes.
Comments / Resolution	Fixed by following recommendations.

RsETHTokenWrapper

The RsETHTokenWrapper does wrapping and unwrapping of alternative rsETH tokens on L2 chains, managing deposits and withdrawals.

Core Invariants:

INV 1: Token must be allowed for deposit/withdrawal
INV 2: Cannot deposit more than maxAmountToDepositBridgerAsset
INV 3: Withdrawal amount must not exceed user balance

Privileged Functions

- initialize
- reinitialize
- depositBridgerAssets
- addAllowedToken
- removeAllowedToken
- mint

Issue_44	maxAmountToDepositBridgerAsset only accounts for one altRsETH token
Severity	**Medium**
Description	RsETHTokenWrapper handles multiple altRsETH tokens(from different bridges eg ccip or layerzero) All rsETH tokens should equally be backed 1:1 for wrsETH. The issue here is, when bridged rsETH is deposited into the contract via depositBridgerAssets the maxAmountToDepositBridgerAsset function used to calculate the amount of unbacked wrsETH only accounts for the current token being deposited. Which can lead to cases where wrsETH is overbacked and tokens will be stuck in the contract.
Recommendations	maxAmountToDepositBridgerAsset should account for all supported tokens. i.e. the returned amount should be wrsETH supply - sum of balances of all altRsETH tokens in the contract.
Comments / Resolution	Acknowledged.

Issue_45	Two step deposit process for L2->L1 user may create disproportionate rsEth:wrsEth amounts causing irresolute withdraws
Severity	**Low**
Description	L2 users can deposit into Kelp by depositing on their preferred L2, which mints wrsEth. From there, the protocol bridges the assets from L2 to L1, purchases rsEth, mints it, and then bridges back to RsEthTokenWrapper on L2. The problem arises from the fact that rsEth is purchased twice, once on L2 and once on L1. If the price increases between those two purchases, the amount of rsEth minted on L1 will no longer match 1:1. These differences may be negligible at first, but as TVL grows on both L1 and L2, the discrepancy increases. One issue is that, since the ratio is not 1:1, if the full corresponding amount of rsEth is bridged to L2, the last user to withdraw or unwrap their wrsEth may not be able to redeem the full amount due to the discrepancy. This issue could occur whenever there is upward price movement. Additionally, because the protocol has a daily mint limit, a whale could deposit on L1 over multiple days, preventing minting on L2 bridged funds and further increasing the discrepancy as the price moves up. The protocol manages this with frequent cron jobs, but this does not resolve the issue.
Recommendations	Consider acknowledging this issue and be prepared to provide rsEth liquidity to L2 if needed.
Comments / Resolution	Acknowledged.

Issue_46	Unused MANAGER_ROLE in RsETHTokenWrapper
Severity	**Informational**
Description	The MANAGER_ROLE is never used in the RsETHTokenWrapper contract. Although the role was assigned on initialization, it is never used. *_setupRole(MANAGER_ROLE, manager);*
Recommendations	Consider removing the role or acknowledging the issue.
Comments / Resolution	Fixed by following recommendations.

Issue_47	Events Deposit and Withdraw emit incorrect sender address
Severity	**Informational**
Description	Events Deposit and Withdraw emit the _to address instead of the msg.sender in their second field which contrasts the event definitions. *event Deposit(address asset, address _sender, uint256 _amount);* *event Withdraw(address asset, address _sender, uint256 _amount);* *emit Withdraw(_asset, _to, _amount);* *emit Deposit(_asset, _to, _amount);*
Recommendations	Consider using msg.sender in the second field.
Comments / Resolution	Fixed by following recommendation.

RSETHPoolV3ExternalBridge

The RSETHPoolV3ExternalBridge does ETH/LST deposits for rsETH minting, manages bridging between L1 and L2, and handles fee collection.

Core Invariants:

INV 1: Daily minted amount must not exceed dailyMintLimit
INV 2: Cannot bridge more than available balance minus fees
INV 3: Token must be supported for deposits

Privileged Functions

- initialize
- reinitialize
- withdrawFees
- bridgeAssetsViaNativeBridge
- bridgeAssets
- bridgeTokens
- setFeeBps
- setRSETHOracle
- addSupportedToken
- removeSupportedToken
- setL1VaultETHForL2Chain
- setStargatePool
- setDstLzChainId
- setSupportedTokenOracle
- setTokenBridge
- setL2Bridge
- setMessenger
- pause
- unpause
- setDailyMintLimit

Issue_48	Bridger can transfer reserved protocol fees
Severity	**Medium**
Description	In the RSETHPoolV3ExternalBridge contract, the bridger role can inadvertently transfer protocol fees that should be reserved for the fee recipient. This occurs in the bridgeAssets function where the balance check does not account for the nativeFee sent with the transaction. For example: 1. Contract has 10 ETH total: 8 ETH available + 2 ETH in fees 2. Bridger calls bridgeAssets with amount = 10 ETH and nativeFee = 2 ETH 3. getETHBalanceMinusFees returns 10 ETH (12 total - 2 fees) 4. Check passes 10 is not less than 10 5. 10 ETH + 2 nativeFee is transferred (12) and contract is empty 6. Fee recipient loses access to their rightful fees
Recommendations	Modify balance check to: getETHBalanceMinusFees() - msg.value < amount to properly exclude the nativeFee from available funds calculation.
Comments / Resolution	Fixed by following recommendations.

Issue_49	Bridge fees silently lost for sonic bridge
Severity	**Low**
Description	In RSETHPoolV3ExternalBridge, the bridgeTokens function forwards msg.value to the bridge call regardless of bridge type. While Arbitrum and Lido bridges require native fees, the Sonic bridge doesn't use them. When bridging through Sonic, any msg.value sent is silently accepted but never used or returned. This inconsistency in fee handling across different bridges creates a risk where even trusted bridger roles could lose funds by following patterns established with other bridges. The issue manifests when: 1. Bridger role sends ETH with bridgeTokens call 2. Function forwards ETH to Sonic bridge via msg.value 3. Sonic bridge ignores ETH value 4. Sent ETH becomes permanently trapped
Recommendations	Document behavior or add off chain validation so that bridger does not include msg.value when bridging from sonic.
Comments / Resolution	Fixed by enforcing msg.value is 0 and adding a function that can recover ETH.

Issue_50	removeSupportedToken() could cause locked funds to protocol
Severity	**Informational**
Description	For RsEthPools on L2, it is impossible to remove a supported token. On L1, there is a deposit limit for each supported token, but on L2 there is not, only direct removal of bridges and oracles. With that said, an intentional or unintentional frontrun deposit before the removal will cause a loss of funds for the protocol (or the user). Scenario: 1. The L2 pool has 0 ethX tokens, so the manager decides to remove support for this token. 2. A malicious user, or even a user unintentionally frontrunning, deposits 100e18 ethX right before the removeSupportedToken() call. 3. Now there are 100e18 ethX in the contract that cannot be bridged, since the token is no longer supported. This results in 100e18 being stuck while the depositor still gets wrsETH. 4. The funds can only be recovered by re-supporting the token.
Recommendations	When removing support for a token consider bundling the transaction with a bridge call to ensure no tokens are left in the contract.
Comments / Resolution	Fixed by adding a check that ensures the contract does not hold any amount of said token during removal. Emptying the entire balance and removing support for a token can be done atomically in order to avoid DOS or front-running scenarios.

Issue_51	No mechanism to redeem wrsETH for underlying ETH/LSTs
Severity	**Informational**
Description	The RssETHPoolV3ExternalBridge contract provides functionality to: - Deposit native tokens or LSTs to receive wrsETH - Bridge these assets to L1 - Deposit these assets on L1 in the LRTDepositPool and receive rsETH on L1 - Bridge back rsETH to L2 to back the minted wrsETH However, there is no functionality that allows users to redeem their wrsETH and receive the underlying returns from their staked assets.
Recommendations	Consider acknowledging the issue as this seems to be a design choice.
Comments / Resolution	Acknowledged.

UnstakeStETH

The UnstakeStETH does stETH unstaking through Lido's withdrawal queue, managing withdrawal requests and claims.

Core Invariants:

INV 1: Full Allowance Must be Used When Requesting Withdrawals

Privileged Functions

- __initializeStETH

Issue_52	Missing storage gaps in unstakeStETH
Severity	**Low**
Description	LRTConverter is an upgradeable contract that inherits from unstakeStETH. This base contract defines state variables, but lacks storage gaps. When LRTConverter is upgraded, adding new variables to these base contracts could cause storage collisions with LRTConverter's own state variables.
Recommendations	Because the contracts are already deployed and adding a gap is no longer an option. Document this risk and use said documentation for future upgrades.
Comments / Resolution	Fixed.

UnstakeSwETH

The UnstakeSwETH does swETH unstaking through SwellNetwork's withdrawal system, managing withdrawal requests and claims.

Core Invariants:

INV 1: Full Allowance Must be Used When Requesting Withdrawal

Privileged Functions

- __initializeSwETH

Issue_53	Missing storage gaps in unstakeSwETH
Severity	**Low**
Description	LRTConverter is an upgradeable contract that inherits from unstakeSwETH. This base contract defines state variables, but lacks storage gaps. When LRTConverter is upgraded, adding new variables to this base contract could cause storage collisions with LRTConverter's own state variables.
Recommendations	Because the contracts are already deployed and adding a gap is no longer an option. Document this risk and use said documentation for future upgrades.
Comments / Resolution	Fixed.

WadMath

The WadMath does WAD (1e18) math operations, providing safe multiplication and division functions.

Core Invariants:

INV 1: WAD constant must be 1e18
INV 2: Division by zero must be prevented
INV 3: Multiplication must not overflow