KELPDAO LRT-ETH SECURITY AUDIT REPORT

March 4, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Kelp DAO provides an end-user solution for restaking, supporting both native Ethereum (ETH) and various staking tokens. It employs EigenLayer for its infrastructure and relies on permissioned validator services for validator node operations.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Kelp DAO
Project name	LRT-ETH
Timeline	29.01.2024 - 27.02.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
26.01.2024	8c62af6057402f4616cc2a1d3218a277a864ee2c	initial commit for the audit (https://github.com/Kelp-DAO/KelpDAO-contracts)
12.02.2024	5acc61e541581926949179d9c5d665bdfa5021ec	commit for the reaudit (https://github.com/Kelp-DAO/LRT-rsETH)
13.02.2024	e75e9ef168a7b192abf76869977cd2ac8134849c	commit with the additional fixes (https://github.com/Kelp-DAO/LRT-rsETH)

Project Scope

The audit covered the following files:

File name	Link
LRTConfig.sol	LRTConfig.sol

File name	Link
LRTOracle.sol	LRTOracle.sol
utils/UtilLib.sol	UtilLib.sol
utils/LRTConfigRoleChecker.sol	LRTConfigRoleChecker.sol
utils/LRTConstants.sol	LRTConstants.sol
LRTDepositPool.sol	LRTDepositPool.sol
RSETH.sol	RSETH.sol
NodeDelegator.sol	NodeDelegator.sol
oracles/OneETHPriceOracle.sol	OneETHPriceOracle.sol
oracles/RSETHPriceFeed.sol	RSETHPriceFeed.sol
oracles/EthXPriceOracle.sol	EthXPriceOracle.sol
oracles/ChainlinkPriceOracle.sol	ChainlinkPriceOracle.sol
oracles/SfrxETHPriceOracle.sol	SfrxETHPriceOracle.sol
cross-chain/CrossChainRateProvider.sol	CrossChainRateProvider.sol
cross-chain/RSETHRateReceiver.sol	RSETHRateReceiver.sol
cross-chain/MultiChainRateProvider.sol	MultiChainRateProvider.sol
cross-chain/CrossChainRateReceiver.sol	CrossChainRateReceiver.sol
cross-chain/RSETHRateProvider.sol	RSETHRateProvider.sol
interfaces/IEigenPodManager.sol	IEigenPodManager.sol
interfaces/IEigenStrategyManager.sol	IEigenStrategyManager.sol
interfaces/IPriceFetcher.sol	IPriceFetcher.sol

File name	Link
interfaces/IEigenPod.sol	IEigenPod.sol
interfaces/ILRTDepositPool.sol	ILRTDepositPool.sol
interfaces/ILRTOracle.sol	ILRTOracle.sol
interfaces/ILayerZeroEndpoint.sol	ILayerZeroEndpoint.sol
interfaces/ILRTConfig.sol	ILRTConfig.sol
interfaces/INodeDelegator.sol	INodeDelegator.sol
interfaces/IStrategy.sol	IStrategy.sol
interfaces/ILayerZeroReceiver.sol	ILayerZeroReceiver.sol
interfaces/IRSETH.sol	IRSETH.sol
interfaces/ILayerZeroUserApplicationConfig.sol	ILayerZeroUserApplicationConfig.sol

Deployments

Contract name	Contract deployed on mainnet	Comment
LRTOracle	0x349A73...0d70020d	proxy
LRTOracle	0xa88845...b21F73a5	implementation
LRTConfig	0x947Cb4...69D65ec7	proxy
LRTConfig	0xc5cD38...3DC29700	implementation
RSETH	0xA1290d...cB99e5A7	proxy
RSETH	0x60FF20...0095A1d2	implementation

Contract name	Contract deployed on mainnet	Comment
LRTDepositPool	0x036676...752D375D	proxy
LRTDepositPool	0xd4114d...759F97Df	implementation
SfrxETHPriceOracle	0x8546A7...568eF427	proxy
SfrxETHPriceOracle	0xD7DB96...691C7704	implementation; license*
EthXPriceOracle	0x3D08cc...aB30dFd2	proxy
EthXPriceOracle	0x0379E8...c63F129E	implementation; license*
OneETHPriceOracle	0x4cB8d6...835160c4	license*
NodeDelegator	0xFc4460...B4559f7E	implementation
NodeDelegator	0x07b96C...E4429473	proxy, index 0
NodeDelegator	0x429554...707748b3	proxy, index 1
NodeDelegator	0x92B4f5...32E6e388	proxy, index 2
NodeDelegator	0x9d2Fc9...3031C4ad	proxy, index 3
NodeDelegator	0xe80382...dF8374E4	proxy, index 4
NodeDelegator	0x049EA1...1f0FBA7B	proxy, index 5
NodeDelegator	0x545D69...1d6d3AB9	proxy, index 6
NodeDelegator	0xee5470...af2D53D3	proxy, index 7
NodeDelegator	0x4C798C...D5712F83	proxy, index 8
NodeDelegator	0x79f172...B9082c32	proxy, index 9
NodeDelegator	0x395884...1c788946	proxy, index 10
NodeDelegator	0xFc5619...2bF1cB85	proxy, index 11

- The deployments with 'license' comments have different SPDX-License-Identifier values in the actual deployments compared to those in the audited commit. This does not affect functionality.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	4
Medium	10
Low	8

ID	Name	Severity	Status
H-1	Excessive rights for the `MANAGER` role	High	Fixed
H-2	An arbitrage opportunity in the rsETH price calculation for MANAGER role	High	Fixed
H-3	Potential withdraw credentials overwritting by a malicious node operator	High	Fixed
H-4	The reward calculation may be blocked until contract upgrade	High	Acknowledged
M-1	Potential blocking of `removeNodeDelegatorContractFromQueue` calls	Medium	Acknowledged
M-2	Potential arbitrage opportunity in the rsETH price calculation for the unpriveleged users	Medium	Acknowledged
M-3	Potential yield stealing due to centralized oracle updates	Medium	Acknowledged
M-4	Unsafe contract deployment and update process	Medium	Acknowledged
M-5	Lack of verification for the native `ETH` balance and staking balance in the `eigenPod`	Medium	Fixed

M-6	Inconsistent support for a native `ETH` token	Medium	Fixed
M-7	Protocol functionality lockout upon adding a new token	Medium	Acknowledged
M-8	Centralization risks	Medium	Acknowledged
M-9	The possible inability to initialize the `eigenPod` field	Medium	Acknowledged
M-10	The Ether can't be withdrawn from the NodeDelegator contract under certain conditions	Medium	Acknowledged
L-1	Using `ERC20.transfer` instead of `SafeERC20.safeTransfer` calls	Low	Acknowledged
L-2	A risk of blocking the functionalities with whitelisting an invalid `NodeDelegator` or `Strategy`	Low	Acknowledged
L-3	A potential race condition possibliity caused by the `removeNodeDelegatorContractFromQueue` function	Low	Acknowledged
L-4	A risk of admin rights revocation due to an incorrect `LRTConfig` address update	Low	Fixed
L-5	Removed `NodeDelegator` addresses from the `LRTDepositPool` queue cannot be added again	Low	Fixed
L-6	Excessive flexibility of LRTConfig	Low	Acknowledged
L-7	Event emission before executing the depositIntoStrategy operation	Low	Fixed
L-8	Inability to remove tokens from the supported list	Low	Acknowledged

1.6 Conclusion

The project under review is currently on an ongoing development stage of implementing the withdrawal mechanism, yet it has already been deployed to the Ethereum mainnet and is operational with a Total Value Locked (TVL) of approximately 140K ETH at the time of this audit. The notable difference between the deployed version and the version subjected to audit include:

- **Integration of Native ETH Deposits**: The introduction of native ETH deposit functionality alongside its utilization for restaking in the EigenPod contract, a component of the Eigen Layer project.
- **Asset Swapping within LRTDepositPool**: The audited framework permits governance to execute asset swaps within the LRTDepositPool contract. This capability is aimed at optimizing asset allocation across the Eigen Layer strategies, particularly targeting those that are underperforming or have been temporarily paused.

The audit process revealed 4 high severity vulnerabilities; notably, these were absent in the version currently deployed. Additionally, several vulnerabilities were categorized as low severity, attributing to the upgradeable nature of the contracts within the project.

The project functions as a pool of multiple ERC-20 LST ETH tokens. Depositors contribute these tokens to the pool in exchange for RSETH shares, which are subsequently engaged in Eigen Layer strategies for reward generation, thereby contributing to the accumulation of the RSETH token's value.

The audit highlighted several key areas of potential attack vectors that could compromise its integrity and the security of depositor stakes. Below is a detailed examination of these vectors and recommended strategies for mitigation:

- **1. Oracle logic**

  - The project's reliance on Chainlink oracles introduces a risk of arbitrage opportunities that could be exploited at the expense of depositor stakes. This vulnerability stems from the potential discrepancy between the actual market price and the oracle-reported price.
  - To mitigate this risk, it is crucial to implement a robust withdrawal function that can safeguard against such arbitrage scenarios. This function should be designed to minimize the impact of price discrepancies on withdrawals.
  - The selection of whitelisted tokens and corresponding oracles requires careful consideration to prevent economically viable attacks. Such attacks could involve inflating the value of illiquid tokens to mint themselves an excessive amount of RSETH, subsequently withdrawing valuable liquid assets like native ETH or stETH from the pool.

- **2. Deployment and update intervention**

  - The deployment and update procedures, as currently designed, are susceptible to intervention by third parties. Although the project has been successfully deployed without incident, there is still a need to revise these processes to ensure they are conducted in a more secure, atomic manner.

- The addition of new asset tokens poses a risk of exposing the contract to intermediate states, as this action is executed through multiple transactions. These states could potentially hinder the functionality related to updating the RSETH price and managing the node delegator list within the LRTDepositPool contract.

- **3. Token balance manipulation**

  - The project's methodology for accounting rewards based on token balances opens the door to manipulation through direct transfers. This includes the potential for artificially inflating the RSETH rate via donations or other means, which is considered safe as much, and would be registered as accumulated rewards.
  - The process for removing existing NodeDelegators and Strategies validates their balances being null. This design allows third parties to block their removal by conducting direct transfers, thereby maintaining a non-zero balance.
  - Inflation attack is deemed infeasible due to the substantial existing total supply of the RSETH token and the minimum deposit amount requirement set by governance. This measure restrains such exploits by making them economically unviable.

- **4. Centralization risks**

  - The governance structure utilizes a 2/4 multisig account for the Manager role and a 3/5 multisig account for the Admin role, with overlapping ownership and an additional address for the latter. This setup, while robust, still presents centralization risks.
  - The Admin role possesses significant authority, including the ability to alter contract implementations, designate RSETH minters and burners, and adjust crucial addresses within the LRTConfig contract. To mitigate the risk of inadvertent disruptive actions by the Admin, the introduction of timelocks on critical governance operations is recommended. Timelocks would provide a buffer period allowing for the identification and correction of potential errors before they take effect.
  - The Manager role was previously able to perform token transfers within the project and manage the asset list, including the withdrawal of tokens, presenting a potential exploit risk. This issue has been addressed by revoking certain privileges associated with this role.
  - Transitioning governance to a DAO structure with embedded timelocks and community-driven decision-making for crucial changes can significantly reduce centralization risks and enhance the security.

- **5. ETH2 stealing with malicious validators**

  - The risk of an ETH2 deposit theft via malicious validators exists, stemming from externally generated validator deposit data. Although the project intends to employ KYC-verified validators, reducing the likelihood of such attacks, the threat remains.
  - To fully exclude the risk of this concern, establishing a robust infrastructure that includes safeguards against malicious validator activities is essential. This could involve enhanced validation processes and security measures to protect user deposits.

- **6. General Project Architecture Review**

    - The project incorporates pausable states from the Eigen Layer contracts, with a partial resolution involving swaps within the DepositPool contract.
    - Updating the RSETH price function requires manual periodic activation to refresh reserve values. This design choice limits scalability, as adding new node delegators and assets increases operational gas costs.
    - The withdrawal mechanism is currently under development, with the detailed specifics of token redemption processes yet to be clarified.
    - The multi-token pool mechanism equitably distributes rewards among users, irrespective of the individual profitability of the strategies they participate in.

Addressing these vulnerabilities and concerns is crucial for the project's long-term viability and success. Implementing the recommended changes will enhance the overall security of the project.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Excessive rights for the `MANAGER` role
Severity	High
Status	Fixed in 9ca0bb57

Description

The current system configuration grants the `MANAGER` role extensive rights giving its owner the possibility to withdraw all the tokens from the contract. Specifically, the `MANAGER` is authorized to:

- Swap the supported tokens within the LRTDepositPool.sol#L355.
- Introduce new supported tokens to the LRTConfig.sol#L60 contract.
- Set the `IPriceFetcher` oracles to the LRTOracle.sol#L93 for the supported tokens.
- Transfer assets from the `NodeDelegator` contracts to the NodeDelegator.sol#L103.

Advantaging these permissions, a manager is able to add their own ERC-20 token and PriceFetcher for it to the system, transfer all the available tokens from the `NodeDelegator` contracts to `DepositorPool`, and then withdraw them using `swapAssetWithinDepositPool`.

While the MANAGER role acts as a sort of restricted administrative account, the fact that the MANAGER and DEFAULT_ADMIN roles are separated is the evidence that, by design, this role is not considered to be absolutely trustworthy.

Recommendation

We recommend revoking the access to call `LRTConfig.addNewSupportedAsset`, `LRTOracle.updatePriceOracleFor` and `LRTDepositPool.swapAssetWithinDepositPool` from the `MANAGER` role, confining such capabilities exclusively to the admin.

H-2	An arbitrage opportunity in the rsETH price calculation for MANAGER role
Severity	High
Status	Fixed in 9ca0bb57

Description

There is an arbitrage opportunity for the `MANAGER` role using the `swapAssetWithinDepositPool` function. Even if some `MANAGER` permissions are revoked, the permission to call `swapAssetWithinDepositPool` is enough to perform profitable arbitrage.

Recommendation

For the multi-asset pools, we recommend using a collaterial-debt approach or AMM liquidity pool approach. Using price oracles for determining the exchange rate within multi-asset pools is generally not recommended.

H-3	Potential withdraw credentials overwritting by a malicious node operator
Severity	High
Status	Fixed in 630dc384

Description

While performing a stake into the EigenLayer NodeDelegator.sol#L185, the Ether could be stolen.

The specification of ETH2.0 staking allows for two types of deposits: the initial deposit and the top-up deposit, which increases the balance of a previously made initial deposit. Unfortunately, the current implementation of the mainnet deposit contract does not sufficiently distinguish between these types of deposits. This oversight allows an attacker to front-run a KelpDAO's deposit with their own initial deposit causing KelpDAO's deposit to be treated as a top-up of the attacker's deposit. Consequently, KelpDAO's withdrawal credentials will be ignored, and the assets will be accounted for on behalf of the attacker.

This issue has been classified as high severity due to the potential for permanent loss of project liquidity. The classification also takes into consideration the developers' statement that all validator nodes are permissioned and have passed KYC, mitigating the overall risk assessment.

Recommendation

Even though this is an architectural issue of ETH2.0 itself, we recommend applying a workaround fix for this issue.

```
        bytes32 actualRoot = depositContract.get_deposit_root();
        if (expectedDepositRoot != actualRoot) {
            revert InvalidDepositRoot(actualRoot);
        }
```

The expected deposit root can be calculated using appropriate offchain mechanics, ensuring that no initial deposit was made to the given pubkey at the time of the expected deposit root calculation.

H-4	The reward calculation may be blocked until contract upgrade
Severity	High
Status	Acknowledged

Description

Currently, the `NodeDelegator.initiateWithdrawRewards()` will NodeDelegator.sol#L232 if the balance of the EigenPod exceeds 16 ETH. This is intended to distinguish between the staking rewards and the stake withdrawal.

It is expected that rewards will be less than 16 ETH; otherwise, something unexpected has occurred (i.e., the validator initiated the withdrawal) and should be resolved manually. It is an ad-hoc temporary solution that will require a contract upgrade by design.

This finding is rated HIGH as the reward calculation may be blocked until a manual contract upgrade.

Recommendation

We recommend developing and upgrading to a long-term solution that does not lead to the freezing of the rewards.

Client's commentary

> Blocking of reward doesn't cause any loss of funds. Later in our next upgrades we are building long term solution, we will withdraw all rewards eth and build a better solution to accomplish this.

2.3 Medium

M-1	Potential blocking of `removeNodeDelegatorContractFromQueue` calls
Severity	Medium
Status	Acknowledged

Description

The issue is found in the LRTDepositPool.sol#L263 function.

This function incorporates an internal verification to ensure the `NodeDelegator` expected to be removed has a zero balance. However, this process opens up the possibility of the manipulation through front-running, where an external party can send 1 wei of token to that `NodeDelegator`, thereby reverting the execution of the removal process.

This vulnerability is classified as `medium` as it poses a risk to potentially blocking the removal process until admin updates the proxy implementation.

Recommendation

We recommend pulling the tokens from the `NodeDelegator` at the time of removal.

Client's commentary

> The fix encompasses only check whether the NDC has ETH staked in EL. We opted to use a private RPC url that has a private pool when we see that there are frontrunning happening. The transaction sent to a private pool rather than the public mempool would avoid this.

M-2	Potential arbitrage opportunity in the rsETH price calculation for the unpriveleged users
Severity	Medium
Status	Acknowledged

Description

Due to the nature of the Chainlink price oracles, the upcoming price updates are predictable. Additionally, the predictable price update can be forced by an attacker by calling the updateRSETHPrice function, which is decentralized/unrestricted. It allows the attacker to predict a profitable price deviation and perform a deposit, external DEX swap, or withdraw to get a guaranteed profit from their actions.

Currently, the withdraw functionality is not implemented, so we can't assess whether it would be safe or not.

Recommendation

For the multi-asset pools, we recommend using a collaterial-debt approach or AMM liquidity pool approach. Using price oracles for determining the exchange rate within multi-asset pools is generally not recommended.

Client's commentary

> 1. We are not using chainlink.
> 2. Manager is using assets/oracles set by admin.
> 3. swapAssetWithinDepositPool is now swapAssetToETH removing the option to profit by a malicious manager
>
>   Profiting updateRSETHPrice is not practical. Every single LST/LRT runs into this problem. The reason this isn't a feasible attack is because of low staking rates. For example, at 3.5% ETH staking rewards a year, exchange rate changes by < 0.01% every day.
> 4. To capture these rewards, users have to deploy a large amount of capital. This is the first barrier. Kelp DAO is at $360M TVL today making this an extremely expensive attack.
> 5. Withdrawals (yet to be implemented) will impose unbonding period greater than 1 day. This enforces a normalizing behavior on user deposits and withdrawals thereby removing any incentive here.

M-3	Potential yield stealing due to centralized oracle updates
Severity	Medium
Status	Acknowledged

Description

The centralized mechanism of the `LRTOracle`, which requires direct invocation of the `updateRSETHPrice` function to register collected rewards, introduces a vulnerability that could enable yield theft. The scenario unfolds as follows:

- Suppose the total supply of `RSETH` and the ETH equivalent locked in the project both stand at 1000. At this point, the `RSETH` to ETH exchange rate equals 1:1.
- One of the events occurs where 1 ETH is donated to the pool or an equivalent amount of rewards is distributed, or the price value of the locked tokens increases by 0.1%.
- An attacker can exploit this minting 1000 `RSETH` for themself by depositing an equivalent of 1000 ETH before the `updateRSETHPrice` transaction, thus capitalizing on the outdated exchange rate before the rewards are recognized.
- Following the `updateRSETHPrice` call, the exchange rate adjusts to 1 `RSETH` per 1.005 ETH, resulting in a 0.5 ETH profit for the attacker.

This issue is classified as `medium` due to its potential to permit unauthorized individuals to steal yield from the pool.

Recommendation

We recommend using timelock of at least one week on the withdrawal process. Implementing such a safeguard makes the described attack economically impractical due to impossibility to instantly redeem the accrued yield.

Client's commentary

> This attack is not practically feasible. It takes a great deal of capital, intent to grief to attempt this attack. Despite that, a user inadvertantly locks their capital up for multiple days making this part of a pooled staking deposit/withdraw behavior.

M-4	Unsafe contract deployment and update process
Severity	Medium
Status	Acknowledged

Description

This issue is identified in the DeployLRT.s.sol scripts and in the history of deploy transactions [1], [2].

The deployment and update process for the `TransparentUpgradeableProxy` contracts is currently made using two sequential transactions:

- Update the proxy implementation
- Invoke the `initialize` function or an update routine.

This approach gives third parties an opportunity to intervene in the deployment or update phase, potentially taking governance control over the deployed contracts.

Recommendation

We recommend utilizing the `upgradeAndCall` function introduced by `EIP-1967` and passing the calldata to the constructor of the proxy to udpate and initialize contracts using a single atomic transaction.

Client's commentary

> The deployment scripts have no bearing to the contracts that are already deployed. To fix future issues, we will make fixes in a future update.

M-5	Lack of verification for the native `ETH` balance and staking balance in the `eigenPod`
Severity	Medium
Status	Fixed in 026d228c

Description

There is a lack of verification for the native `ETH` balance and staking balance in the `eigenPod` within the `removeNodeDelegatorContractFromQueue` function, potentially leading to the discrepancies of the `RSETH` price.

Recommendation

We recommend checking zero balance during the `removeNodeDelegatorContractFromQueue` function.

Client's commentary

> Verification is paused on eigenlayer contracts since they launched, i.e. no one has verified yet. We have confirmed this with EigenLayer.

M-6	Inconsistent support for a native `ETH` token
Severity	Medium
Status	Fixed in 9ca0bb57

Description

The current strategy for integrating the `ETH` support within the contract relies on usage of the native `ETH`, as opposed to the use of the ERC20 interface for balance checks and transactions. Consequently, native `ETH` is treated as an exceptional case, necessitating duplicate operations and the inclusion of a mocked `ETH` address within the list of supported assets. This approach introduces several issues:

- The balance of the mocked token is verified in the LRTDepositPool.sol#L263 function.
- The `isSupportedAsset` check modifier is overlooked in several functions, including LRTDepositPool.sol#L146, LRTDepositPool.sol#L336, NodeDelegator.sol#L175.
- The LRTDepositPool.sol#L348 function lacks native ETH support.

Recommendation

We recommend utilizing the `WETH` contract instead of native `ETH` to facilitate more polimorphic architecture. This causes converting native `ETH` into `WETH` in the `LRTDepositPool.depositETH` function and utilizing the `WETH` as a general `ERC20` token, thereby eliminating the need for duplicate functions and the inclusion of a mocked asset among the supported assets. Furthermore, for accurate `NodeDelegate.stakeETH` processing, a `WETH` token can be unwrapped to the native `ETH` prior to its transfer to the `EigenPod` and wrapped again upon withdrawal.

M-7	Protocol functionality lockout upon adding a new token
Severity	Medium
Status	Acknowledged

Description

To add a supported token to the project, the administrator and manager have to call 3 different functions:

- addNewSupportedAsset() in LRTConfig
- updateAssetStrategy() in LRTConfig
- updatePriceOracleFor() in LRTOracle

If the sequence of calls to these functions is contained in different transactions, the protocol is in an intermediate state for some time, during which the updateRSETHPrice() function is blocked.

Recommendation

We recommend consolidating the mentioned functions into one and making the token addition procedure atomic.

Client's commentary

> Addition of newer assets are an infrequent occurence. While this causes minor incovenience to users, asset additions are conducted professionaly by signers of admin and manager multisigs. We anticipate little to none as impact because of asset addition.

M-8	Centralization risks
Severity	Medium
Status	Acknowledged

Description

The project administrator and manager have unrestricted rights, including:

- Contract code updates,
- Project configuration changes,
- Addition of an arbitrary number of RSETH minters (though minting capability is only used in LRTDepositPool),
- Oracle appointments.

These permissions pose centralization risks.

Recommendation

We recommend creating a DAO system with time-delayed execution of updates.

Client's commentary

> Every single DeFi protocol out there runs on a select set of people being able to update/rewrite almost all code. With Kelp, our ADMIN multisig is public and signers are public too. Our signers have started other successful DeFi companies in the past. Since Kelp does not have a governance token yet, it is too early for us to discuss the end state of governance here.

M-9	The possible inability to initialize the `eigenPod` field
Severity	Medium
Status	Acknowledged

Description

The problem has been identified in the NodeDelegator.sol#L185 function.

Invoking the `stakeETH` method prior to executing `createEigenPod` leads to an implicit deployment of the `eigenPod`. However, this action fails to initialize its address within the `NodeDelegator` contract. Consequently, it blocks the `verifyWithdrawalCredentials` and `getETHEigenPodBalance` preventing accurate accounting of withdrawals and rewards derived from the ETH restaking in the pod. This condition persists until admin intervenes to update the implementation of proxy with a function that recovers the address.

This issue is classified as `high` due to the inability to initialize `eigenPod`, which may result in discrepancies in the accounting of withdrawal and reward assets, affecting the `RSETHPrice` value. Recovery from this state is feasible only through a proxy update.

Recommendation

We recommend initializaing the `eigenPod` field in the `stakeETH` function, if it is wasn't deployed earlier explicitly using the `createEigenPod` call.

Client's commentary

> Before validator keys could be generated, we need the EigenPod address. So there is no case we could attribute a random validator key to a NDC without successfully passing offchain checks. Additionaly, the suggested modification costs more gas.

M-10	The Ether can't be withdrawn from the NodeDelegator contract under certain conditions
Severity	Medium
Status	Acknowledged

Description

If `ETH_TOKEN` is not included in the list of supported assets, Ether can still be LRTDepositPool.sol#L352 to the `NDC` using the `transferETHToNodeDelegator` function. However, it NodeDelegator.sol#L103 using the `transferBackToLRTDepositPool` function.

Recommendation

We recommend checking the presence of `ETH_TOKEN` in the supportedAssets list during the execution of the sendETHFromDepositPoolToNDC function.

Client's commentary

> We would add ETH_TOKEN as a part of upgrade process.
> Even if someone sends ETH to NDC using transferETHToNodeDelegator or using normal eth transfer, we are okay with ETH lying there in NDC.

2.4 Low

L-1	Using `ERC20.transfer` instead of `SafeERC20.safeTransfer` calls
Severity	Low
Status	Acknowledged

Description

Utilizing `ERC20.transfer` instead of `SafeERC20.safeTransfer` within the protocol is identified as a security risk. This is due to the possibility for certain tokens to adopt non-standard variations of the `IERC20` interface, which do not return boolean values upon executing transfers. Additionally, the protocol's `swapAssetWithinDepositPool` function currently omits the verification of return values from transfers. Although the tokens currently whitelisted implement the standard `ERC20` implementation, the protocol does not accommodate a broader spectrum of `IERC20` token variations, rendering it potentially incompatible with certain tokens.

Recommendation

We recommend adopting the use of `SafeERC20.safeTransfer` throughout the implementation.

Client's commentary

> We are using only LST which was created a year or so ago. They follow the correct ERC20 standard. We feel that the safeERC20 lib would increase the gas use without much benefit. The benefit would be if we used any ERC20 contract address or known tokens with this issue such as USDT.

L-2	A risk of blocking the functionalities with whitelisting an invalid `NodeDelegator` or `Strategy`
Severity	Low
Status	Acknowledged

Description

The incorporation of `NodeDelegator` into the `LRTDepositPool` that is doesn't implement the `INodeDelegator` interface or the `Strategy` to `LRTConfig` that is not whitelisted by the Eigen Layer protocol's strategy list presents a risk of blocking the `LRTOracle.RSETHPriceUpdate` functions and inability to remove these contracts from the added list due to require checks represented as view function calls.

Recommendation

We recommend using the `EIP-165` interface for the identification of contracts that implement the `NodeDelegator` interface, and checking that the `lrtConfig` of added `NodeDelegator` is the same as the `lrtConfig` of `LRTDepositPool`. Additionally, for the `Strategy` contracts, a validation process should be implmented to ensure that a strategy proposed for addition has been authorized for deposits within the `StrategyManager`.

Client's commentary

> We appreciate the callout here but we believe Kelp DAO will perform validations and checks to ensure contracts matching the right interface are deployed as Node Delegators. As an added security layer, having fixed the Node removal logic, we are fairly certain bad NDCs can be removed.

L-3	A potential race condition possibliity caused by the `removeNodeDelegatorContractFromQueue` function
Severity	Low
Status	Acknowledged

Description

This issue is identified in the LRTDepositPool.sol#L317 and LRTDepositPool.sol#L336 functions of the `LRTDepositPool` contract.

A race condition arises when the `transferAssetToNodeDelegator`(or `transferETHToNodeDelegator`) and `removeNodeDelegator` functions are executed concurrently. Specifically, if the `removeNodeDelegator` function is processed first, leading to the deletion of the intended `NodeDelegator` for transfer, subsequent asset transfers via `transferAssetsToNodeDelegator` could inadvertently be directed to another address. This issue arises from the usage of the `ndcIndex` argument within the `transferAssetToNodeDelegator` and `transferEthToNodeDelegator` functions, which can point to another `NodeDelegator` after the deletion.

Recommendation

We recommend using an address as an argument of `NodeDelegator` expected to receive the transfer instead of its index within the queue.

Client's commentary

> RemoveNodeDelegatorContract and transfer funds are both functions of manager (a multisig with multiple required signers). It is unlikely for this issue as describe to happen. We do not expect to change code to address this issue.

L-4	A risk of admin rights revocation due to an incorrect `LRTConfig` address update
Severity	Low
Status	Fixed in 9ca0bb57

Description

The issue is identified in theLRTConfigRoleChecker.sol#L61 function.

There exists a potential risk when the admin rights could inadvertently be revoked through the updating of the `LRTConfig` address with an incorrect address. Such a scenario may occur if the new address provided does not correspond to an actual `LRTConfig` contract, thereby disrupting the functionality of all operations dependent on the `LRTConfig` interface. Alternatively, if the new address is a valid `LRTConfig` contract but configured with a different `DEFAULT_ADMIN` address, the admin role would be transferred to the holder of the new address.

Recommendation

We recommend eliminating the `updateLRTConfig` function, as the `LRTConfig` contract is designed as an `UpgradeableProxy`, direct changes of its address pose risks to the governance and protocol security.

L-5	Removed `NodeDelegator` addresses from the `LRTDepositPool` queue cannot be added again
Severity	Low
Status	Fixed in 9ca0bb57

Description

The issue is identified in the LRTDepositPool.sol#L291 function.

The `addNodeDelegatorContractToQueue` function includes a verification step to ensure that a node delegator already exists in the `isNodeDelegator` mapping. If the mapping indicates presence, the function skips adding the address back into the NodeDelegators list. This mechanism, while intended to prevent duplicates, inadvertently creates a block against re-adding a node delegator once it has been removed. The `removeNodeDelegatorContractFromQueue` function doesn't mark the address as deleted in the mapping, thereby rendering its reintegration into the `lRTDepositPool` impossible under the current logic.

Recommendation

We recommend adjusting the `removeNodeDelegatorContractFromQueue` function to reset the corresponding `isNodeDelegator` mapping entry for the removed address to `false`.

L-6	Excessive flexibility of LRTConfig
Severity	Low
Status	Acknowledged

Description

In addition to the ability to update contracts and change configuration addresses "on the fly" by calling setters in the LRTConfig contract, there are mappings, such as tokenMap and contractMap, containing token and contract addresses obtained from hashes declared in the LRTConstants contract. This architecture is redundant and inefficient in terms of gas usage.

Using immutable variables (alongside upgradable contracts) instead of constants and mappings provides greater gas efficiency, reduces code volume, and enhances readability.

Recommendation

We recommend using immutable variables instead of constants and mappings.

Client's commentary

> Most functions on LRT Configs are callable by Admin which indicates the infrequent usage of these functions. We appreciate the suggestion to follow another pattern for enhanced readability. We do not believe there is a need to change any code in this file because of this suggestion.

L-7	Event emission before executing the depositIntoStrategy operation
Severity	Low
Status	Fixed in 9ca0bb57

Description

In the depositAssetIntoStrategy() function, the AssetDepositIntoStrategy event is emitted before the actual execution of the depositIntoStrategy operation. According to common rules, events should be emitted after the execution of any operation.

Recommendation

We recommend swapping the emission of the AssetDepositIntoStrategy event and the execution of the depositIntoStrategy function.

L-8	Inability to remove tokens from the supported list
Severity	Low
Status	Acknowledged

Description

When adding a large number of tokens, the project may encounter gas limits when updating the oracle price in updateRSETHPrice(), as this function iterates through all tokens.

Recommendation

We recommend adding a function to remove tokens from the system.

Client's commentary

> We don't anticipate more than 10 assets in Kelp. This finding does not necessitate changes in Kelp's code.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes