Rio Network

Rio Liquid Restaking Tokens

Security Assessment Report

Version: 2.0

April, 2024

Contents

Introduction

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

Disclaimer

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

Document Structure

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

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

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

Overview

Rio Network is a network for issuing Liquid Restaking Tokens (LRTs), with a mission to provide the best access to risk-managed `ETH` total return without sacrificing liquidity.

LRTs aggregate rewards from both staking and restaking operations into a fungible token (eg. `reETH`) while abstracting the complexity of running highly-available infrastructure, selecting restaking applications, and selling rewards from the end user.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to files at commit 1e0fd7a and changes in PR#117.

Retesting was performed on commit 99cde8c.

The list of assessed files is as follows:

- ChainlinkPriceFeed.sol
- RioLRT.sol
- RioLRTAVSRegistry.sol
- RioLRTAssetRegistry.sol
- RioLRTCoordinator.sol
- RioLRTDepositPool.sol
- RioLRTIssuer.sol
- RioLRTOperatorDelegator.sol
- RioLRTOperatorRegistry.sol
- RioLRTRewardDistributor.sol
- RioLRTWithdrawalQueue.sol
- RioLRTCore.sol
- RioLRTOperatorRegistryStorageV1.sol
- Array.sol
- Asset.sol
- Constants.sol
- LRTAddressCalculator.sol
- Memory.sol
- OperatorOperations.sol
- OperatorRegistryV1Admin.sol
- OperatorUtilizationHeap.sol
- ValidatorDetails.sol

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

Approach

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

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

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

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

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

Output for these automated tools is available upon request.

Coverage Limitations

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

Findings Summary

The testing team identified a total of 16 issues during this assessment. Categorised by their severity:

- Critical: 5 issues.
- High: 1 issue.
- Medium: 2 issues.
- Low: 4 issues.
- Informational: 4 issues.

Detailed Findings

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

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

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

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

Summary of Findings

ID	Description	Severity	Status
RIO-01	Malicious Validators Can Steal ETH Deposits By Configuring Custom Withdrawal Addresses	Critical	Resolved
RIO-02	Validators Not Reported For Early Exits	Critical	Resolved
RIO-03	Erroneous Public Key After Validator Swap Operation	Critical	Resolved
RIO-04	Function `receive()` Always Fails	Critical	Closed
RIO-05	Incorrect Use Of Heap Causes Denial Of Service	Critical	Resolved
RIO-06	Reentrancy During Deposit Allows Unbacked Minting Of LST Tokens	High	Closed
RIO-07	Lack of Onchain Validator Confirmation and Verification	Medium	Closed
RIO-08	Depositors Lose LRT Due To Rounding Differences	Medium	Resolved
RIO-09	No Duplicate Contract Protection In The AVS Registry	Low	Resolved
RIO-10	Asset Removal Can Be Blocked By Factors Outside Of The Protocol's Control	Low	Resolved
RIO-11	Insufficient Checks When Setting An Asset Price Feed	Low	Resolved
RIO-12	Potential Locked ETH When Issuing A New LRT	Low	Resolved
RIO-13	`getPrice()` Will Call Any Address	Informational	Resolved
RIO-14	Potentially More Expensive `get()` Methods	Informational	Closed
RIO-15	Cumbersome Process Of Changing Reward Share Values	Informational	Resolved
RIO-16	Miscellaneous General Comments	Informational	Resolved

RIO-01	Malicious Validators Can Steal ETH Deposits By Configuring Custom Withdrawal Addresses		
Asset	`RioLRTCoordinator.sol, *DepositPool.sol, *OperatorDelegator.sol, OperatorOperations.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

It is possible for operators to configure their validators with withdrawal addresses other than the Rio Eigenpod. As a result, all ETH sent to these validators would then only be withdrawable to that malicious address.

This can be done by simply making a direct 1 ETH deposit into the Ethereum deposit contract from a malicious withdrawal address.

In the current system design, as communicated by the development team, this can be done at any time as the withdrawal address check is contained within `RioLRTOperatorRegistry.verifyWithdrawalCredentials()`, which is only called after the deposit has been made.

Furthermore, the checks implemented in this function (Eigenlayer's `verifyWithdrawalCredentials()`) cannot prevent the attack even if it was made earlier in the process. This is because an operator can set up the validators without withdrawal addresses and then frontrun the deposit transaction to add them later. The deposit transaction would still go through.

Note, the system has a tendency to make a large number of deposits to validators of the same operator at the same time. It is possible that over 3,200 ETH could be stolen in a single transaction using this attack.

Recommendations

There are multiple potential approaches to preventing this attack:

- One possibility is to have the Rio protocol deposit 1 ETH to the validators first, check the withdrawal address, and then deposit the remaining 31 ETH. This approach still risks 1 ETH per validator, though, for a potential maximum theft of 110 ETH per transaction, as currently configured.
- Another potential mitigation is to have an automated off chain system to check on the consensus layer that each validator does not have a withdrawal address and record the deposit data root. The automated system would then initiate the deposit transaction (via `rebalance()`) with the deposit data root as an argument. The deposit functions would then check `get_deposit_root()` on the Ethereum deposit contract and proceed only if it was unchanged.

Resolution

The addition of an automated off-chain system was added to check whether each validator lacks a withdrawal address and signs the deposit data root, exposing the root and signature via an API. As well as the addition of the removal capability of confirmed validator keys along with pending keys.

This issue has been addressed in commit 2d0638f.

RIO-02	Validators Not Reported For Early Exits		
Asset	`utils/ValidatorDetails.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

If a validator exits early, and is next in the sequence to be exited, reporting the event through a call to `reportOutOfOrderValidatorExits.reportOutOfOrderValidatorExits()` will always revert. As a result, the validator will not be reported for an early exit.

Also, if `reportOutOfOrderValidatorExits()` is called to report `n` validators, and this range overlaps with the next `n` validators in the sequence to exit, that function call will also revert.

The issue occurs because `ValidatorDetails.swapValidatorDetails()` has a check on line [**129-131**] which reverts if called to swap two overlapping ranges. This function is always called in `reportOutOfOrderValidatorExits()` to swap the reported validators with the next validators in sequence, but there are no checks to ensure that these ranges do not overlap. In the case where the next validator in line to exit is reported, it will always overlap with itself, so it cannot be reported for an early exit.

Recommendations

Change the logic in `reportOutOfOrderValidatorExits()` to only swap the necessary validator range, leaving overlapping members of both ranges in place.

Resolution

The substitution of a check to allow swapping between adjacent indexes was made.

This issue has been addressed in commit 25bcb51.

RIO-03	Erroneous Public Key After Validator Swap Operation		
Asset	`utils/ValidatorDetails.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The function `ValidatorDetails.swapValidatorDetails()` copies the pubkey bytes into memory in the wrong order and overwrites the middle of the swapped keys with zeroes. This will result in the loss of saved validator keys during the protocol's operation.

On line [**149**], the assembly variable `_part2`, when loaded from storage, contains the last 16 bytes of the pubkey in its first 16 bytes. When this is stored into the `key1` memory variable, it is shifted right by 16 bytes and stored from the position `0x30` onwards, so that the assigned data does not extend into the memory space of another variable. However, this requires `part2` to be assigned before `part1`, as in the function `loadValidatorDetails()`. If `part2` is assigned first, it overwrites the second half of `part1` with its leading zeroes.

The exact same issue is present for `key2` on line [**158-159**].

This can be plainly observed from this test output where the original key data is a string of 1s or 4s:

```
ValidatorDetailsSwapped(
operatorId: 43,
pubkey1: 0x11111111111111111111111111111110000000000000000000000000000001111111111111111111111111111111,
pubkey2: 0x44444444444444444444444444444440000000000000000000000000000004444444444444444444444444444444
)
```

Recommendations

Change implementation of how the second part of the key is stored in storage. Do not shift left on save and right on load. Instead, store bytes 16..47 in the second key storage slot. This saves gas and also allows the key parts to be loaded in any order and saved in any order.

Resolution

The change in writing to storage was implemented.

This issue has been addressed in commit b9ff082.

RIO-04	Function `receive()` Always Fails		
Asset	`restaking/RioLRTOperatorDelegator.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The `receive()` function always fails to receive native tokens due to improper gas management.

The `receive()` function calls `Asset.transferETH()` to transfer native tokens to `rewardDistributor()` address. If we assume that this call will execute `RioLRTRewardDistributor.receive()`, then the following conditions occur:

- `Asset.transferETH()` has a gas limit of 10,000 gas.
- In most cases, the function `RioLRTRewardDistributor.receive()` will call `Asset.transferETH()` three times, in order to transfer native tokens to `treasury`, `operatorRewardPool`, and `depositPool()` addresses.

Since `RioLRTRewardDistributor.receive()` is called using 10,000 gas, then it is unreasonable to expect the gas to last three times the limit. Therefore, the operation always fails.

Recommendations

Replace `transferETH()` on `RioLRTOperatorDelegator` with another operator that has a more flexible gas limit.

Resolution

The issue has been addressed in PR#117 . The `Asset.transferETH()` no longer limits the gas usage to 10,000 gas.

However, the lack of limitation would open up the possibility that a receiver could insert any code into their `receive()` function and this would then be executed when they are sent ETH. Therefore, consider adding a function called something like `trustedTransferETH()` which has no gas limit. Use this for the `RioLRTOperatorDelegator.receive()` function and restore `transferETH()` to its previous gas limited, protected form.

The development team have further acknowledged the issue.

RIO-05	Incorrect Use Of Heap Causes Denial Of Service		
Asset	`utils/OperatorRegistryV1Admin.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

There exists an edge case where deactivating a certain operator disrupts the mechanics of the heap (implemented in `utils/OperatorUtilizationHeap.sol`) such that no new operator can be added because the operation will always revert with `DivWadFailed()` message.

The edge case occurs if the following conditions hold:

- Add operators, where the total number of operators is >= 3
- Deactivate operator, where the deactivated operator is the last operator (with the highest id)

The issue is caused by `getOperatorUtilizationHeapForETH()` where this function assigns `heap.operators[]` directly instead of utilising the heap's feature to insert new data (i.e., `insert()`).

It is worth noting that the same issue was discovered in function `getOperatorUtilizationHeapForStrategy()` where it directly assigns `heap.operators[]` instead of `heap.insert()`, which prevents a successful call to `RioLRTOperatorRegistry.setOperatorStrategyShareCaps()`.

Recommendations

Replace the code on line [**379-382**] to use `heap.insert()`.

Resolution

New logic wherein operator IDs outside of the current `heap.count()` are removed from storage was added.

This issue has been addressed in commit 473cb0b.

RIO-06	Reentrancy During Deposit Allows Unbacked Minting Of LST Tokens		
Asset	`restaking/RioLRTCoordinator.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

If an asset with a transfer hook capability was deposited, it would be possible to execute a reentrancy attack during the call to `transferFrom()` on line [82] and mint more LRT tokens than anticipated.

The attack is possible because, after the `transferFrom()` call on line [82], the state of the protocol would be that the balance of the deposit pool has increased, but the total supply of the `RioLRT` token has not. This will distort the output of the following functions:

- `RioLRTCoordinator.convertFromUnitOfAccountToRestakingTokens()`
- `RioLRTCoordinator.convertToUnitOfAccountFromRestakingTokens()`
- `RioLRTCoordinator.convertFromAssetToRestakingTokens()`
- `RioLRTCoordinator.convertToAssetFromRestakingTokens()`
- `RioLRTCoordinator.convertToSharesFromRestakingTokens()`

Two of these functions are used in `requestWithdrawal()` to assess how many assets should be paid out for the deposited number of LRT tokens. It could be possible to call `deposit()` and, within the reentrancy attack, queue withdrawals of the deposited asset at the distorted rate (the attacker would need to have acquired some LRT tokens in advance).

The attack is even more effective if there are assets already in `RioLRTDepositPool` from other users, as they can also be withdrawn at the advantageous rate.

The main mitigation against this attack is that the reentrancy needs to be possible: ie. it needs to be possible for the attacker to take over the control flow during the `transferFrom()` on line [82]. This would be possible if any supported asset ever implemented any kind of transfer hook. Given that external complex assets like restaking tokens are being used, and that these external protocols could easily be in active development, it is considered a realistic risk that at some point one of the assets might make a call on transfer under some circumstance that might possibly allow an external call.

Recommendations

Implement a reentrancy guard on `deposit()` and `requestWithdrawal()`. Ideally, this reentrancy guard should also cover all of the above mentioned conversion functions. This would provide other protocols with protection from read-only reentrancy attacks via the Rio protocol.

Resolution

The development team acknowledged the issue and it will be managed by the protocol on an ongoing basis.

RIO-07	Lack of Onchain Validator Confirmation and Verification		
Asset	`restaking/RioLRTOperatorRegistry.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The Rio system relies on an automated offchain system to check the validity of the validator signature and confirm that a provided `pubKey` is not a duplicate of either another queued validator in the Rio system or of an existing validator on the consensus layer.

If this automated system fails to run, the confirmation period stored in `validatorDetails.nextConfirmationTimestamp` will automatically expire and the validator will be considered confirmed without any checks being run.

The development team stated that it is unlikely, but not impossible, for the automated checking system to malfunction and fail to run. It is even less likely that the automated checking system would fail to run at the precise time when validators are being confirmed. It is, however, still possible, and the consequences are potentially significant - if a validator with an invalid signature deposits to the Ethereum deposit contract, the 32 ETH deposit would be lost.

Recommendations

Instead of a passive expiry period implemented on line [273], implement an automated confirmation system to submit a transaction to increase the value of `s.operatorDetails[operatorId].validatorDetails.confirmed` after its checks have been made.

Resolution

The development team acknowledged the issue and provided the following comment:

> *"We plan to use a sufficiently long key review period along with monitoring to ensure that no keys are missed during review. In addition, as a part of the solution for RIO-01, confirmed keys will undergo review and may be removed if necessary."*

RIO-08	Depositors Lose LRT Due To Rounding Differences		
Asset	`restaking/RioLRTCoordinator.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

When `requestWithdrawal()` is called to withdraw ETH, the amount of ETH received is rounded down to the nearest gwei. However, the number of LRT tokens transferred in by the user is not modified and remains as the amount specified in the parameter `amountIn`. That number of LRT tokens is then burned, resulting in a small loss to the withdrawing user.

Recommendations

Consider recalculating the amount of LRT tokens to burn based off the rounded down `sharesOwed` value.

Alternatively, it may be that the development team do not wish to preserve these amounts, reasoning that automatically creating many small residual balances of LRT tokens is undesirable. In this case, consider informing users of the potential for that small precision loss.

Resolution

This issue has been addressed by removing the rounding logic from `requestWithdrawal()` in a PR outside the scope of this review.

RIO-09	No Duplicate Contract Protection In The AVS Registry		
Asset	`restaking/RioLRTAVSRegistry.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Because there is no duplicate checking in the functions of this contract, it is possible to register the same registry and slashing contracts for different AVS ids. Activating and deactivating one AVS id would then have the consequence of adding or removing the contract of another AVS id from the mappings `_isActiveSlashingContract` and `_isActiveRegistryContract`. This would affect the outputs of the associated getter functions.

This is not an issue if it is certain that no AVS will ever have the same registry or slashing contract as another AVS.

Recommendations

As all the functions that modify the AVS registry are `onlyOwner`, it is possible that this issue can be managed through administration.

It might be beneficial to add checks to `addAVS()` to prevent any AVS that shares a registry, or a slashing contract, with another AVS from being added.

Resolution

The addition of a check was made to prevent AVSs from sharing slashing or registry contracts.

This issue has been addressed in commit a902c28.

RIO-10	Asset Removal Can Be Blocked By Factors Outside Of The Protocol's Control		
Asset	`restaking/RioLRTAssetRegistry.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There is a check in the function `removeAsset()` on line [**252**] that blocks removal of an asset if, amongst other factors, the deposit pool contract has any balance. This could be triggered by a donation of the asset to the protocol or by some issue in the token contract of the asset.

In the former case, a tiny donation would require a withdrawal before the asset could be removed. To prevent a repeat of this attack, the withdrawal and asset removal would need to be processed in the same transaction.

The latter case would occur if there is some unforeseen problem with the token contract. A worst case scenario would be if the deposit pool were blacklisted by the contract and was unable to interact with it at all.

In this scenario, only a contract upgrade with new code would enable the asset to be removed.

Recommendations

Consider a forced removal process that allows admin to remove an asset even with a balance. It might be desirable to submit the balance amount to be ignored as one of the parameters to this function to ensure that circumstances have not changed.

Resolution

The addition of a forced asset removal function, `forceRemoveAsset()`, was made that allows the DAO to remove an underlying asset with a balance.

This issue has been addressed in commit 3cae8d4.

RIO-11	Insufficient Checks When Setting An Asset Price Feed		
Asset	`restaking/RioLRTAssetRegistry.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When adding an asset, there are various checks performed on the asset's price feed, however, some of these checks are not implemented in `setAssetPriceFeed()`. As such, it is possible to bypass these checks by simply setting an asset price feed later.

For ERC20 assets, there is a check on line [337] to ensure that the price feed has the correct number of decimals.

```
if (IPriceFeed(newPriceFeed).decimals() != priceFeedDecimals) revert INVALID_PRICE_FEED_DECIMALS();
```

For ETH, there is a check on line [331] to ensure that no price feed is set:

```
if (config.asset == ETH_ADDRESS) {
    if (config.priceFeed != address(0)) revert INVALID_PRICE_FEED();
```

Without these checks, it would be possible (albeit unlikely) for `owner` to add a price feed to the ETH asset or add a price feed with the wrong number of decimals for an ERC20 asset.

Recommendations

Implement the missing checks in `setAssetPriceFeed()`.

Resolution

The addition of price feed checks were made.

This issue has been addressed in commit d16d79d.

RIO-12	Potential Locked ETH When Issuing A New LRT		
Asset	`restaking/RioLRTIssuer.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

It is possible to configure a new LRT with an ERC20 asset and send ETH in that transaction, which would then result in it being locked in the `RioLRTIssuer` contract. As that contract is upgradeable, it would be possible to rescue the trapped ETH, but only by a contract upgrade.

The function `issueLRT()` is `payable`. If the asset in the `config` argument is not ETH, but ETH is still sent in the call to `issueLRT()`, then the test on line [**165**] will not pass and any `msg.value` sent with the call would be ignored.

Note, the issue may only occur during issuance of a new LRT, which can only be done by `owner`, which is assumed to be a trusted account.

Recommendations

Implement the following check on line [**169**]:

```
} else if (msg.value > 0) {
    revert INVALID_ETH_PROVIDED();
}
```

Resolution

Recommended check was added to ensure `ETH` is not provided when the sacrificial deposit asset is not `ETH`.

This issue has been addressed in commit c4c6386.

RIO-13	`getPrice()` Will Call Any Address
Asset	`restaking/RioLRTAssetRegistry.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

With the exception of the zero address, `getPrice(address)` will call `getPrice()` on any supplied address and return the result.

Whilst this does not lead to any observed direct vulnerability within the current code, it could cause a risk to future code or external code. It is possible that other code could assume that any price returned by the protocol's asset registry relates to a supported asset or is at least a legitimate price feed.

Recommendations

Consider checking the supplied address parameter to see if it is a price feed supported by the protocol.

Alternatively, if the overhead from such a check is deemed unjustified, add a clear warning to all documentation for this function that explains that the function performs no checks on the price feed and that output should not be trusted unless the price feed parameter is known and trusted.

Resolution

The addition of a warning was made to the `getPrice()` function to alert others to the untrusted nature of the function.

This issue has been addressed in commit 8d19e33.

RIO-14	Potentially More Expensive `get()` Methods
Asset	`restaking/base/RioLRTCore.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The address getter functions in `RioLRTCore` contract such as `coordinator()` and `assetRegistry()` calculate the requested data. This method can be more expensive in the long term than a standard `store-retrieve` method.

Our experiment indicates 937 gas when calling `LRTAddressCalculator.getCoordinator()`, whereas it costs 22,459 gas to store an address and 286 gas to retrieve the address from storage. The data suggests that it takes 35 calls for the `store-retrieve` method to start getting cheaper (accumulatively) than the existing address calculation method.

Recommendations

Consider storing the addresses instead of calculating them for each call for a cheaper gas cost in the long term.

Resolution

The development team acknowledged the issue with the following comment:

> *"The address calculation approach in LRTAddressCalculator is optimized to minimize the number of cold SLOADs. This method is more cost-effective than the store-retrieve approach when accessing two or more addresses within a single function, which is necessary in the most frequently called protocol functions, such as deposit and request withdrawal."*

RIO-15	Cumbersome Process Of Changing Reward Share Values
Asset	`restaking/RioLRTRwardDistributor.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Function `_setTreasuryETHValidatorRewardShareBPS()` and `_setOperatorETHValidatorRewardShareBPS()` update the values of `treasuryETHValidatorRewardShareBPS` and `operatorETHValidatorRewardShareBPS` respectively. Before the changes are made, the inputs must satisfy the following requirement:

```
newTreasuryETHValidatorRewardShareBPS + operatorETHValidatorRewardShareBPS > MAX_BPS
```

When changing one value, the other value may need to be adjusted first, and therefore it may require more than one transaction to set a desirable setting.

Recommendations

Consider having one function to modify both parameters at the same time, where a check can be done for all values.

Resolution

The addition of a function that follows the recommendation was made.

This issue has been addressed in commit 100d5c2.

RIO-16	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **Unrestricted Access To `burn()` Function**
   Related Asset(s): restaking/RioLRT.sol
   After consultation with the development team, it is understood that there is no legitimate intended public usage for this function.
   As a matter of best practice, it is recommended to restrict access to this function to only the intended users.
2. **Optimal Data Type For Timestamp**
   Related Asset(s): restaking/RioLRTCoordinator.sol
   A mapping named `assetNextRebalanceAfter` is used to record the next eligible timestamp for the rebalancing of assets. The mapping pairs asset addresses with the timestamp values and it is using `uint256` data type for the timestamps. By staying consistent and using `uint40` data type instead of `uint256` can help avoid the need for data type conversions that could introduce some complexity.
3. **Inconsistent Treatment Of Array Variables**
   Related Asset(s): restaking/RioLRTDepositPool.sol
   The `completeOperatorWithdrawalForAsset()` function handles withdrawals from the EigenLayer system. The function processes withdrawals by interacting with strategies and shares that deal with assets. The issue lies in how the contract deals with strategies and shares array that is within the `IDelegationManager.Withdrawal` struct. The function enforces a constraint that `IDelegationManager.Withdrawal.strategies` is not allowed to exceed 1, but it does not do the same for `IDelegationManager.Withdrawal.shares` to ensure that each share in the array corresponds to a strategy.
   The testing team recommends adding a check for shares.
4. **Variable Could Be Immutable**
   Related Asset(s): oracle/ChainlinkPriceFeed.sol
   The variable `description` on line [**21**] could be declared `immutable`.
   Consider declaring `description` as `immutable`.
5. **Minor Inaccuracy In Function Description**
   Related Asset(s): utils/OperatorUtilizationHeap.sol
   The function `_isGrandchild()` is described as determining "whether the node at `m` is a grandchild of the node at `i`". Strictly speaking, the function only determines whether the node at `m` is lower in the tree than `i`'s children. If the function is only ever used in the context where `m` is known to be either a child or grandchild of `i`, as is the case in this file, then the function has the effect described in the comment. But the comment is strictly inaccurate.
   Consider revising the description of the function in the comment to add something like, "Only valid if `m` is already known to be either a child or grandchild of `i`.".

6. **Comment Names Wrong Operation**

   Related Asset(s): utils/ValidatorDetails.sol

   The parameter `keysCount` for the function `removeValidatorDetails()` is described as "Keys count to load." when it is actually the number of keys to remove.

   Correct the comment on line [**203**].

7. **Comment On Return Value Is Misleading**

   Related Asset(s): utils/ValidatorDetails.sol

   The return value for the function `saveValidatorDetails()` is described as the "New total keys count". Strictly speaking, the returned value is actually the last index modified. It is only the new total key count if we assume that this function is only used to append.

   Consider rewording to clarify this distinction.

8. **Marking Function As `internal`**

   Related Asset(s): restaking/RioLRTWithdrawalQueue.sol

   Function `_computeWithdrawalRoot()` is marked as public, while the operations may not be relevant for external parties. Rather, the naming convention of other functions in the contract indicates this is an internal function.

   Consider changing from `public` to `internal`.

9. **Potential Arithmetic Over/Underflow**

   Related Asset(s): restaking/RioLRTAssetRegistry.sol

   The functions on line [**302**] and line [**320**] do not check whether the deduction exceeds the balance. As a result, an `Arithmetic over/underflow` error can occur. However, when the codes are executed in the right context (for example through `RioLRTDepositPool.completeOperatorWithdrawalForAsset()` and `RioLRTWithdrawalQueue.settleEpochFromEigenLayer()`), such error may not be possible.

   Consider adding a check to prevent the error.

10. **Identical Error Message**

    Related Asset(s): utils/OperatorRegistryV1Admin.sol

    line [**47**] produces the same error as line [**81**] of `RioLRTOperatorDelegator.sol`. When function `addOperator()` is executed (from `RioLRTOperatorRegistry.addOperator()` for example), both instructions are executed and therefore it would be harder to track which one caused the error message.

    Consider differentiating the error message for a better troubleshooting and debugging experience.

11. **Storing Price Feed Source As `IChainlinkAggregatorV3`**

    Related Asset(s): oracle/ChainlinkPriceFeed.sol

    The immutable data `source` is stored as `address`. However, in `constructor` and `getPrice()`, this information is constantly converted from address to `IChainlinkAggregatorV3`.

    To improve code readability, consider storing `source` as `IChainlinkAggregatorV3`.

Recommendations

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

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in ceb8a80 where appropriate.

Appendix A Test Suite

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

```
Running 7 tests for test/Memory.sigp.t.sol:Sigp_MemoryTest
[PASS] test_sigp_copyBytesOutOfBounds() (gas: 3614)
[PASS] test_sigp_copyBytesWithLength() (gas: 21158)
[PASS] test_sigp_copyBytesWithoutLength() (gas: 21203)
[PASS] test_sigp_memcpyMultipleWords() (gas: 1368)
[PASS] test_sigp_memcpySingleWord() (gas: 929)
[PASS] test_sigp_unsafeAllocateBytesCheckFreeMemoryPointer() (gas: 2414)
[PASS] test_sigp_unsafeAllocateBytesCheckReturnedValue() (gas: 2433)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 933.79us

Running 3 tests for test/RioLRT.sigp.t.sol:Sigp_RioLRT
[PASS] testFail_sigp_burnMoreChecks() (gas: 64251)
[PASS] test_sigp_CLOCK_MODE() (gas: 12089)
[PASS] test_sigp_clock() (gas: 13381)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 14.36ms

Running 2 tests for test/Array.sigp.t.sol:Sigp_ArrayTest
[PASS] test_sigp_toArrayAddress() (gas: 623)
[PASS] test_sigp_toArrayUint256() (gas: 546)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 700.07us

Running 1 test for test/RioLRTAVSRegistry.dimaz.t.sol:Sigp_dimaz_RioLRTAVSRegistryTest
[PASS] test_sigp_addAVS_zero_slashingContract() (gas: 145378)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 84.46ms

Running 4 tests for test/RioLRTAVSRegistry.sigp.t.sol:Sigp_RioLRTAVSRegistryTest
[PASS] testFail_sigp_activateAVS_AddDuplicate() (gas: 163244)
[PASS] testFail_sigp_addAVS_AddDuplicate() (gas: 168848)
[PASS] test_sigp_RevertWhen_activateAVS_ActivateWrongID() (gas: 143664)
[PASS] test_sigp_RevertWhen_activateAVS_DeactivateWrongID() (gas: 178113)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 18.43ms

Running 1 test for test/PocDepositReentrancy.sigp.t.sol:PocDepositReentrancy
[PASS] test_sigp_reentrancyPoc() (gas: 1156694)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 21.92ms

Running 1 test for test/RioLRTIssuer.sigp.t.sol:Sigp_RioLRTIssuerTest
[PASS] testFail_sigp_issuesLRT_WithValidParamsAndNonZeroMsgValue() (gas: 394448)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.15ms

Running 5 tests for test/RioLRTDepositPool.sigp.t.sol:Sigp_RioLRTDepositPool
[PASS] test_sigp_completeOperatorERC20ExitToDepositPoolWithEvents() (gas: 7228411)
[PASS] test_sigp_completeOperatorETHExitToDepositPoolWithEvents() (gas: 9314301)
[PASS] test_sigp_completeOperatorWithdrawalForAssetMultipleStrategiesReverts() (gas: 15622)
[PASS] test_sigp_depositBalanceIntoEigenLayerLargeEtherDepositExtendsPastBufferLimit() (gas: 74370739)
[PASS] test_sigp_transferMaxAssetsForSharesHasPartialEtherAmountPrecisionChecks() (gas: 62460)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 101.30ms

Running 6 tests for test/ValidatorDetails.sigp.t.sol:Sigp_ValidatorDetailsTest
[PASS] test_sigp_removeValidatorDetailsError() (gas: 3335)
[PASS] test_sigp_removeValidatorDetailsHappyPath() (gas: 946265)
[PASS] test_sigp_saveAndLoadValidatorDetails(uint8,uint256,uint256) (runs: 3, u: 98530798, ~: 113305281)
[PASS] test_sigp_saveValidatorDetailsErrors() (gas: 4742)
[PASS] test_sigp_swapValidatorDetailsErrors() (gas: 13307)
[PASS] test_sigp_swapValidatorDetailsHappyPath() (gas: 1427874)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 962.64ms

Running 15 tests for test/RioLRTCoordinator.sigp.t.sol:Sigp_RioLRTCoordinatorTest
[PASS] testFail_sigp_convertFromUnitOfAccountToRestakingTokensZeroSupply() (gas: 96665)
[PASS] testFail_sigp_convertToUnitOfAccountFromRestakingTokensZeroSupply() (gas: 96676)
[PASS] test_sigp_convertFromAssetToRestakingTokensHappyPath() (gas: 140651)
```

```
[PASS] test_sigp_convertFromUnitOfAccountToRestakingTokensHappyPath() (gas: 68170)
[PASS] test_sigp_convertToAssetFromRestakingTokensHappyPath() (gas: 140681)
[PASS] test_sigp_convertToSharesFromRestakingTokensHappyPath() (gas: 150694)
[PASS] test_sigp_convertToUnitOfAccountFromRestakingTokensHappyPath() (gas: 68212)
[PASS] test_sigp_depositErc20HappyPath() (gas: 246629)
[PASS] test_sigp_depositEthHappyPath() (gas: 145500)
[PASS] test_sigp_depositEth_receive(uint256) (runs: 10000, u: 141783, ~: 141783)
[PASS] test_sigp_depositEtherDepositCapExceededReverts() (gas: 62822)
[PASS] test_sigp_rebalanceAssetNotSupportedReverts() (gas: 29658)
[PASS] test_sigp_requestErc20WithdrawalHappyPath() (gas: 448789)
[PASS] test_sigp_requestEtherWithdrawalHappyPath() (gas: 298093)
[PASS] test_sigp_requestEtherWithdrawalInsufficientShares() (gas: 490569)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 2.82s

Running 1 test for test/RioLRTWithdrawalQueue.sigp.t.sol:Sigp_RioLRTWithdrawalQueueTest
[PASS] test_sigp_queueWithdrawal(uint256) (runs: 10000, u: 139990, ~: 33039)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.24s

Running 13 tests for test/LRTAddressCalculator.sigp.t.sol:Sigp_LRTAddressCalculatorTest
[PASS] test_sigp_computeOperatorSalt() (gas: 278)
[PASS] test_sigp_computeSalt() (gas: 460)
[PASS] test_sigp_getAVSRegistry() (gas: 1161)
[PASS] test_sigp_getAssetRegistry() (gas: 1140)
[PASS] test_sigp_getContractAddress() (gas: 1107)
[PASS] test_sigp_getCoordinator() (gas: 1206)
[PASS] test_sigp_getCoordinator(address,address) (runs: 10000, u: 6531, ~: 6531)
[PASS] test_sigp_getDepositPool() (gas: 1164)
[PASS] test_sigp_getOperatorDelegatorAddress() (gas: 836)
[PASS] test_sigp_getOperatorRegistry() (gas: 1184)
[PASS] test_sigp_getRewardDistributor() (gas: 1141)
[PASS] test_sigp_getWithdrawalQueue() (gas: 1120)
[PASS] test_sigp_store_getAddressCalculator(address) (runs: 10000, u: 28804, ~: 28806)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 5.58s

Running 12 tests for test/OperatorUtilizationHeap.sigp.t.sol:Sigp_OperatorUtilizationHeapTest
[PASS] test_sigp_extractMax(uint8,uint8) (runs: 10000, u: 2512635, ~: 2124978)
[PASS] test_sigp_extractMin(uint8,uint8) (runs: 10000, u: 2517414, ~: 2126121)
[PASS] test_sigp_full() (gas: 65366)
[PASS] test_sigp_initialize() (gas: 7570)
[PASS] test_sigp_insert(uint8) (runs: 10000, u: 36348, ~: 25452)
[PASS] test_sigp_removeByID_heap(uint8,uint8) (runs: 10000, u: 2085750, ~: 1954177)
[PASS] test_sigp_removeByID_heap_last(uint8) (runs: 10000, u: 2276592, ~: 2151593)
[PASS] test_sigp_removeByID_heap_not_last(uint8,uint8) (runs: 10000, u: 2266356, ~: 2391758)
[PASS] test_sigp_remove_heap(uint8,uint8) (runs: 10000, u: 2096631, ~: 1873227)
[PASS] test_sigp_store(uint8) (runs: 10000, u: 55981, ~: 40929)
[PASS] test_sigp_updateUtilizationByID(uint8,uint8) (runs: 10000, u: 2541164, ~: 2216706)
[PASS] test_sigp_updateUtilization_(uint8,uint8) (runs: 10000, u: 2485220, ~: 2121423)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 18.98s

Running 4 tests for test/Asset.sigp.t.sol:Sigp_AssetTest
[PASS] test_sigp_getSelfBalance(uint256) (runs: 10000, u: 23734, ~: 23734)
[PASS] test_sigp_reducePrecisionToGwei(uint256) (runs: 10000, u: 713, ~: 713)
[PASS] test_sigp_transferETH_revert(uint256) (runs: 10000, u: 17375, ~: 17375)
[PASS] test_sigp_transferTo(address,uint256) (runs: 10000, u: 43365, ~: 43365)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 33.39s

Running 17 tests for test/RioLRTAssetRegistry.sigp.t.sol:Sigp_RioLRTAssetRegistryTest
[PASS] test_sigp_addAsset_removeAsset(uint256,uint256) (runs: 10000, u: 4712019, ~: 5248658)
[PASS] test_sigp_addAsset_removeAsset_revert() (gas: 1812693)
[PASS] test_sigp_addAsset_success(address,uint96,address,address) (runs: 10000, u: 111753, ~: 111754)
[PASS] test_sigp_decreaseSharesHeldForAsset(uint256) (runs: 10000, u: 919697, ~: 919697)
[PASS] test_sigp_decreaseSharesHeldForAsset_AssetNotSupported() (gas: 24370)
[PASS] test_sigp_decreaseSharesHeldForAsset_overflow(uint256) (runs: 10000, u: 908244, ~: 908244)
[PASS] test_sigp_decreaseUnverifiedValidatorETHBalance(uint256) (runs: 10000, u: 39721, ~: 39920)
[PASS] test_sigp_decreaseUnverifiedValidatorETHBalance_overflow(uint256) (runs: 10000, u: 25761, ~: 25761)
[PASS] test_sigp_increaseSharesHeldForAsset(uint256) (runs: 10000, u: 927510, ~: 928259)
[PASS] test_sigp_increaseSharesHeldForAsset_AssetNotSupported() (gas: 21797)
[PASS] test_sigp_increaseUnverifiedValidatorETHBalance(uint256) (runs: 10000, u: 44790, ~: 45485)
[PASS] test_sigp_setAssetDepositCap(uint96) (runs: 10000, u: 903142, ~: 903142)
[PASS] test_sigp_setAssetDepositCap_AssetNotSupported() (gas: 18664)
```

```
[PASS] test_sigp_setAssetPriceFeed() (gas: 1072535)
[PASS] test_sigp_setAssetPriceFeed_AssetNotSupported() (gas: 18553)
[PASS] test_sigp_setAssetPriceFeed_InvalidPriceFeed() (gas: 900075)
[PASS] test_sigp_validate_data() (gas: 12832)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 33.36s

Running 3 tests for test/RioLRTOperatorDelegator.sigp.t.sol:Sigp_RioLRTOperatorDelegatorTest
[PASS] test_sigp_receive_operator_delegator_vuln(uint256) (runs: 10000, u: 6681736, ~: 6681796)
[PASS] test_sigp_stakeERC20_LST(uint256) (runs: 10000, u: 6967415, ~: 6967415)
[PASS] test_sigp_stakeETH(uint256) (runs: 10000, u: 6998673, ~: 6859654)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 54.11s

Running 14 tests for test/RioLRTOperatorRegistry.sigp.t.sol:Sigp_RioLRTOperatorRegistryTest
[PASS] test_sigp_addOperator_addOperator() (gas: 3740882)
[PASS] test_sigp_addOperator_deactivateOperator_addOperator(uint8) (runs: 10000, u: 10649422, ~: 9758068)
[PASS] test_sigp_addOperator_deactivate_activate() (gas: 3031806)
[PASS] test_sigp_addOperator_deactivate_activate_setOperatorStrategyShareCaps() (gas: 3089901)
[PASS] test_sigp_addOperator_deactivate_activate_setOperatorValidatorCap() (gas: 3045386)
[PASS] test_sigp_create3_addOperator(uint8) (runs: 10000, u: 1215418, ~: 1215418)
[PASS] test_sigp_removeValidatorDetails() (gas: 4149022)
[PASS] test_sigp_removeValidatorDetails_fail() (gas: 1454393)
[PASS] test_sigp_setMinStakerOptOutBlocks(uint24) (runs: 10000, u: 25963, ~: 26150)
[PASS] test_sigp_setOperatorEarningsReceiver(uint8,uint8,address) (runs: 10000, u: 3808373, ~: 3949909)
[PASS] test_sigp_setOperatorPendingManager(uint8,uint8,address) (runs: 10000, u: 3821533, ~: 3959983)
[PASS] test_sigp_setProofUploader(address) (runs: 10000, u: 46294, ~: 46294)
[PASS] test_sigp_setSecurityDaemon(address) (runs: 10000, u: 29233, ~: 29233)
[PASS] test_sigp_setValidatorKeyReviewPeriod(uint24) (runs: 10000, u: 25636, ~: 26098)
Test result: ok. 14 passed; 0 failed; 0 skipped; finished in 84.78s

Ran 17 test suites: 109 tests passed, 0 failed, 0 skipped (109 total tests)
```

Appendix B Vulnerability Severity Classification

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

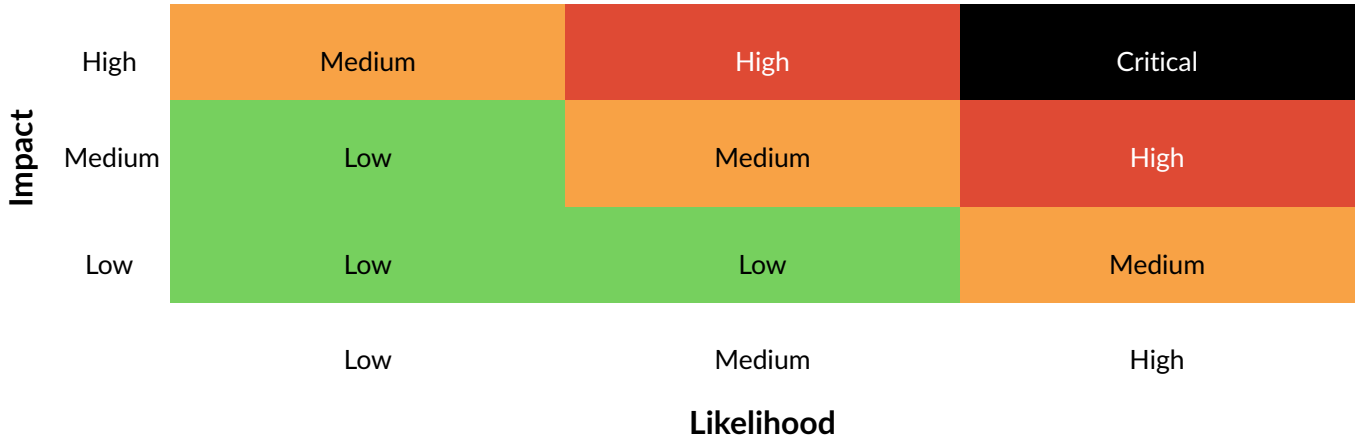

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

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

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