Zero-G Finance Security Review

April 7, 2024

Conducted by:
Blckhv, Independent Security Researcher
Slavcheww, Independent Security Researcher

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Zero-G Finance stands as an innovative Liquid Restaking Platform, carving a niche for itself by granting users seamless entry into the vast domain of EigenLayer. At its core, Zero-G serves as a gateway, enabling users to tap into EigenLayer Operators offering yield surpassing that of conventional Ethereum staking.

Overview

Project	Zero-G Finance
Repository	https://github.com/zero-g-fi/zero-g-contracts
Commit Hash	d36792c72e379cd0fc762d9f5c25fd7a57aed4cf
Resolution	108ea97bc40fda2d28f480a31f8da1913f1b86ce
Date	April 1 – April 4, 2024

Scope

LRTConfig.sol
LRTDepositPool.sol
LRTOracle.sol
NodeDelegator.sol
ZgETH.sol

Issues Found

High Risk	2
Medium Risk	3
Low/Info Risk	7

5. Findings

5.1. High severity

5.1.1. Inflation attack in LRTDepositPool's deposit functions

Severity: High Risk

Context: LRTDepositPool.sol#L128

Description: minAmountToDeposit is to prevent users from depositing values like 1 wei with which they will be able to inflate the price of zgETH, minAmountToDeposit will be 0 until an admin sets it through setMinAmountToDeposit, which will allow any user to frontrun depositing 1 wei of ETH and then inflate the price of zgETH.

Steps:

1. Alice become the first depositor, depositing 1 wei which will mint her 1 wei of zgETH.
2. Then she will donate 10ETH(can be anything. The value donated here will become the minimum amount on which users will receive anything) to the LRTDepositPool, which will increase the ETH balance in the contract.
3. Then she will directly donate at least the amount of the next depositor (10e18 for the example), making the right side of the equation 1 wei more:
4. The price of zgETH will be (10e18 + 1) * 1e18.
5. Any subsequent staker who deposits less than (10e18 + 1) will:
    A. Not receive any zgETH if he pass 0 as minZgETHAmountExpected (which is the slippage parameter) and will lose his money.

    B. If he specified something relevant to his deposit amount as a slippage, the function will revert, which will cause deposits under a certain value to be blocked.

Recommendation: Set minAmountToDeposit in the initialize.

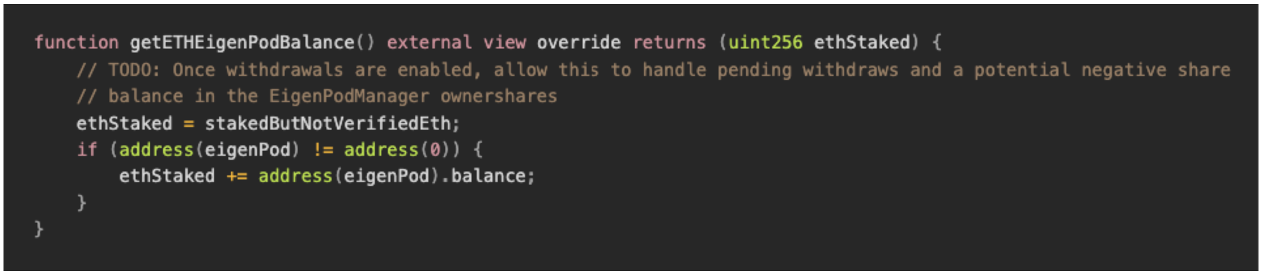
Resolution: Fixed

5.1.2. Double accounting of EigenPod stakes

Severity: High Risk

Context: NodeDelegator.sol#L207

Description: getETHEigenPodBalance will increase the staked balance of NodeDelegator twice because it sums both the balance of EigenPod and stakedButNotVerifiedEth. The problem is that stakedButNotVerifiedEth is already up to date with the balance of an EigenPod since it is being updated in both stakeEth and stake32EthValidated.



```
function getETHEigenPodBalance() external view override returns (uint256 ethStaked) {
    // TODO: Once withdrawals are enabled, allow this to handle pending withdraws and a potential negative share
    // balance in the EigenPodManager ownershares
    ethStaked = stakedButNotVerifiedEth;
    if (address(eigenPod) != address(0)) {
        ethStaked += address(eigenPod).balance;
    }
}
```

Consequences from the artificial 2x increase of balance are two major flaws in the system:

1. LRTDepositPool::getTotalAssetDeposits will return inaccurate data and the deposit limit will appear as filled, when it is not.
2. LRTDepositPool::getTotalAssetDeposits for ETH will be 2x more than the real staked balance, which will also make the price of zgETH 2x more:

Recommendation: Remove the eigenPod address balance check and rely only on stakedButNotVerifiedEth. It already has up-to date balance of all the stakes of this NodeDelegator

Resolution: Fixed

5.2. Medium severity

5.2.1. NodeDelegator removing can be blocked by sending 1 wei

Severity: Medium Risk

Context: LRTDepositPool.sol#L244

Description: When removing NodeDelegator in LRTDepositPool.removeNodeDelegatorContractFromQueue() it checks if the NodeDelegator has any ETH in it or in its corresponding EigenPod. This allows anyone to block the removal by front-running and sending 1 wei of ETH to that NodeDelegator.

Recommendation: Use a private PRC with a private mempool or withdraw all leftover or front funds during removal.

Resolution: Acknowledged

5.2.2. Malicious manager can profit from swapAssetWithinDepositPool

Severity: Medium Risk

Context: LRTDepositPool.sol#L441

Description: swapAssetWithinDepositPool is used by the manager to exchange any 2 supported assets in LRTDepositPool. Although it doesn't assume the LSD tokens to be at a 1:1 ratio, the price feeds used will be in ETH denominator. All LSD/ETH token feeds have 1 day heartbeat and 0.5% price deviation.

On rare occasions from asset and to asset can have asymmetric price moves that can go unnoticed by the Oracle, because of the feed configuration. Then manager can profit from it with a swap from asset: **LSD with a decreased price and to asset: LSD with an increased price**, oracle will return the latest prices of both assets, without price impact considered, and transfer the manager the same amount of tokens from the more expensive token.

Recommendation: Consider leaving only swapETHForAssetWithinDepositPool, although it contains the same flaw, impact is reduced by having to calculate only the toAsset's return amount.

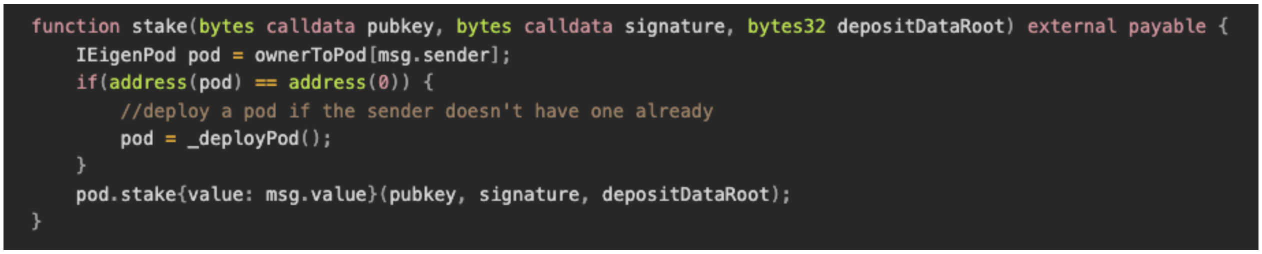
Resolution: Fixed

5.2.3. Stake without deploying EigenPod will cause all stakes to be lost

Severity: Medium Risk

Context: NodeDelegator.sol#L230, L263

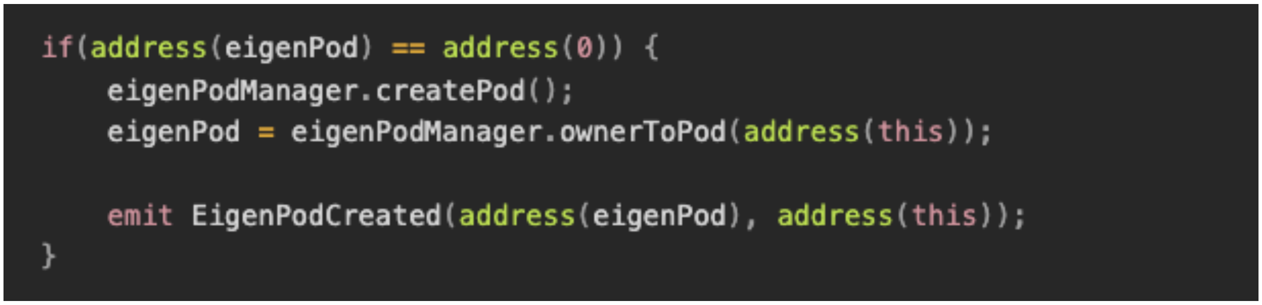
Description: If an operator of NodeDelegator stakes in EigenPod, without first calling the createEigenPod function, the entire stake will be lost. The problem occurs because EigenPodManager will deploy the pod on behalf of the caller if there is not any on his address, here is the stake function of EigenLayer:



```
function stake(bytes calldata pubkey, bytes calldata signature, bytes32 depositDataRoot) external payable {
    IEigenPod pod = ownerToPod[msg.sender];
    if(address(pod) == address(0)) {
        //deploy a pod if the sender doesn't have one already
        pod = _deployPod();
    }
    pod.stake{value: msg.value}(pubkey, signature, depositDataRoot);
}
```

Staking will be successful but eigenPod will be address(0) in the NodeDelegator and the functions relying on it won't show accurate data.

Recommendation: Add check to verify that eigenPod is not address(0) at the beginning of both stakeEth and stake32EthValidated functions, if so deploy and set its address:



```
if(address(eigenPod) == address(0)) {
    eigenPodManager.createPod();
    eigenPod = eigenPodManager.ownerToPod(address(this));

    emit EigenPodCreated(address(eigenPod), address(this));
}
```

Resolution: Fixed

5.3. Low/Info severity

5.3.1. Excessive admin privileges

Severity: Low Risk

Context: LRTConfig.sol, LRTDepositPool.sol, LRTOracle.sol, NodeDelegator.sol, ZgETH.sol

Description: There are several instances in the codebase that give excessive rights to the admins:

- no cap when setting the fee configuration of NodeDelegator, possible to be up to 99%.
- can swap freely any 2 supported assets, eventually profiting from the big heartbeat period of Chainlink oracle.

- can increase the price of zgETH token by setting a new one with totalSupply = 0.
- operator, manager, and admin will be all given to a single multisig wallet.
- introduce new supported tokens to the LRTConfig contract.
- set the IPriceFetcher oracles to the LRTOracle for the supported tokens.
- transfer assets from the NodeDelegator contracts to the DepositorPool.

Recommendation: Our advice would be to document these scenarios and if possible mitigate some of them, for example:

- decrease the max basis points of the feeConfig.
- if appropriate, distribute the roles to more than one address.

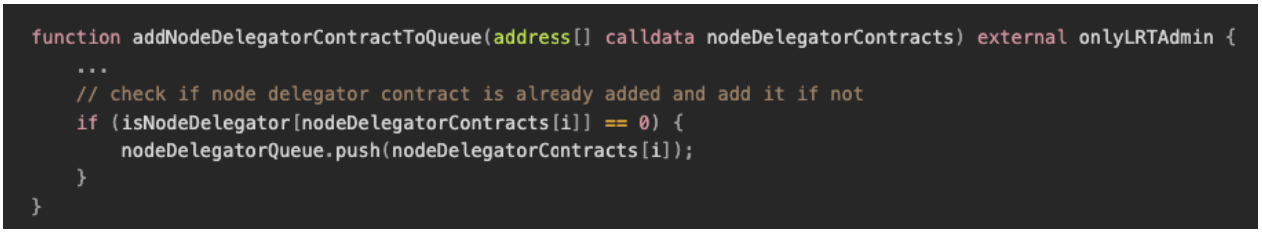
Resolution: Acknowledged

5.3.2. Removed NodeDelegator cannot be added again in LRTDepositPool

Severity: Low Risk

Context: LRTDepositPool.sol#L290

Description: When NodeDelegator is removed from queue, isNodeDelegator mapping is not being reset to 0, later on if there is a need the same NodeDelegator to be re-onboarded, this won't be possible because of the following check:



```
function addNodeDelegatorContractToQueue(address[] calldata nodeDelegatorContracts) external onlyLRTAdmin {
    ...
    // check if node delegator contract is already added and add it if not
    if (isNodeDelegator[nodeDelegatorContracts[i]] == 0) {
        nodeDelegatorQueue.push(nodeDelegatorContracts[i]);
    }
}
```

Recommendation: When remove a nodeDelegator set the isNodeDelegator mapping to 0.

Resolution: Fixed

5.3.3. transferAssetsToNodeDelegator is missing the amount param

Severity: Low Risk

Context: LRTDepositPool.sol#L341

Description: transferAssetsToNodeDelegator is designed to be like transferAssetToNodeDelegator but in a loop. It currently sends all assets to a single nodeDelegator, so if we consider this:

We have 3 assets and we want to distribute them equally in 2 NodeDelegators, with transferAssetToNodeDelegator that will cost 6 transactions, but with transferAssetsToNodeDelegator this cannot be achieved because the entire asset balance will be transferred to the first one, making the function unusable.

Recommendation: Add amount[] so you can specify an amount. With this, the upper case can be achieved in 2 transactions.

Resolution: Acknowledged

5.3.4. Redundant onlyLRTAdmin in removeManyNodeDelegatorContractsFromQueue

Severity: Informational Risk

Context: LRTDepositPool.sol#302

Description: removeManyNodeDelegatorContractsFromQueue function has a redundant onlyLRTAdmin modifier, because it is already called in the internal function: removeNodeDelegatorContractFromQueue.

Recommendation: It can safely be removed since it doesn't contribute any additional security measures to the contract.

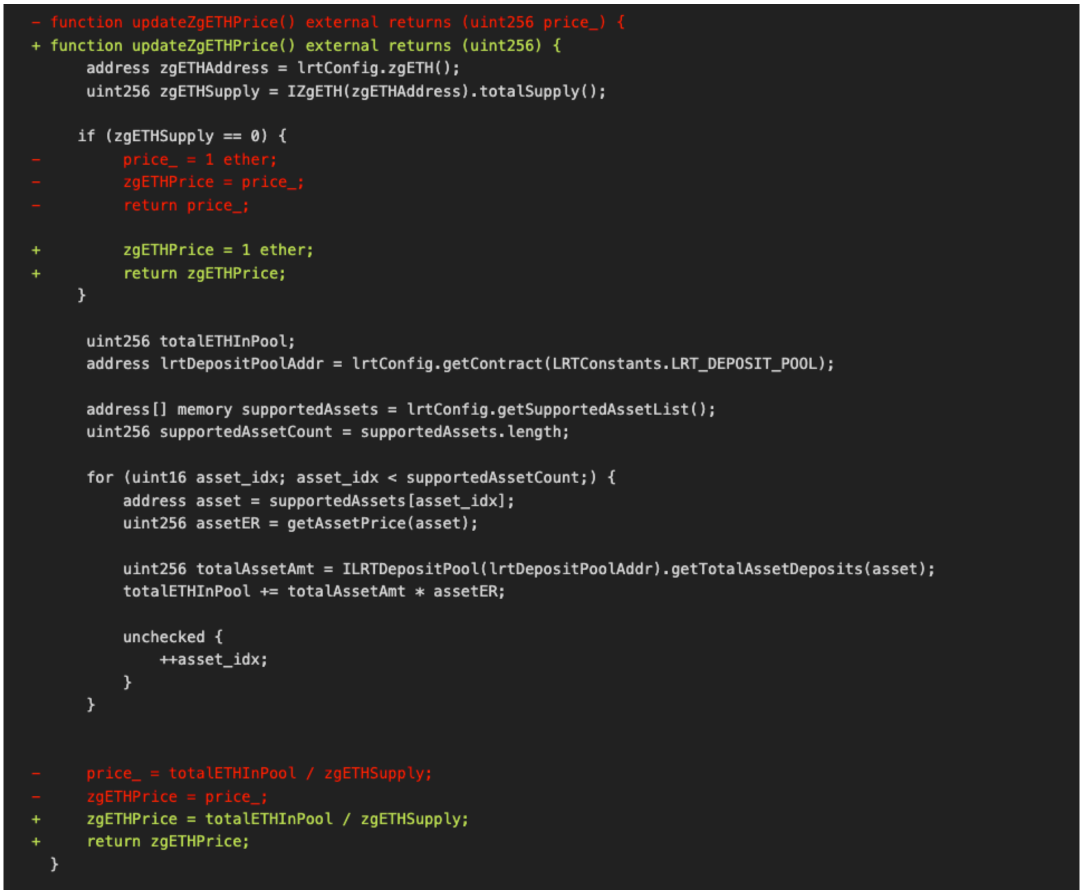
Resolution: Acknowledged

5.3.5. updateZgETHPrice can be simplified

Severity: Informational Risk

Context: LRTOracle.sol#L51

Description: Function to update the price of the `zgETH` token - `updateZgETHPrice` can be simplified with the following modifications:



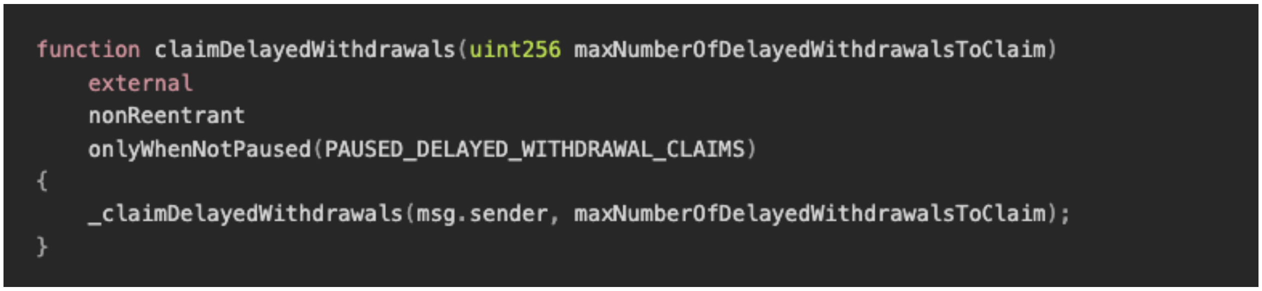
```
- function updateZgETHPrice() external returns (uint256 price_) {
+ function updateZgETHPrice() external returns (uint256) {
      address zgETHAddress = lrtConfig.zgETH();
      uint256 zgETHSupply = IZgETH(zgETHAddress).totalSupply();

     if (zgETHSupply == 0) {
-         price_ = 1 ether;
-         zgETHPrice = price_;
-         return price_;

+         zgETHPrice = 1 ether;
+         return zgETHPrice;
     }

      uint256 totalETHInPool;
      address lrtDepositPoolAddr = lrtConfig.getContract(LRTConstants.LRT_DEPOSIT_POOL);

      address[] memory supportedAssets = lrtConfig.getSupportedAssetList();
      uint256 supportedAssetCount = supportedAssets.length;

      for (uint16 asset_idx; asset_idx < supportedAssetCount;) {
          address asset = supportedAssets[asset_idx];
          uint256 assetER = getAssetPrice(asset);

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

          unchecked {
              ++asset_idx;
          }
      }


-     price_ = totalETHInPool / zgETHSupply;
-     zgETHPrice = price_;
+     zgETHPrice = totalETHInPool / zgETHSupply;
+     return zgETHPrice;
  }
```

5.3.6. claimDelayedWithdrawals can be used without passing recipient params

Severity: Informational Risk

Context: NodeDelegator.sol#L285

Description: claimDelayedWithdrawals is used to claim nonBeaconChainETHBalanceWei of the EigenPod, before staking in the Beacon Deposit contract, however, there is another overload of the claim function that omits the recipient argument and passes msg.sender instead.



```
function claimDelayedWithdrawals(uint256 maxNumberOfDelayedWithdrawalsToClaim)
    external
    nonReentrant
    onlyWhenNotPaused(PAUSED_DELAYED_WITHDRAWAL_CLAIMS)
{
    _claimDelayedWithdrawals(msg.sender, maxNumberOfDelayedWithdrawalsToClaim);
}
```

Recommendation: Consider using the claimDelayedWithdrawals, without passing recipient, as it will make the code slightly more readable.

Resolution: Acknowledged

5.3.7. Missing nonReentrant in the receive function

Severity: Informational Risk

Context: NodeDelegator.sol#L355

Description: The receive function of NodeDelegator lacks `nonReentrant` modifier, although there is no way to harm the protocol, it will prevent `feeAddress` from reentering the contract.

Recommendation: Consider adding the `nonReentrant` modifier.

Resolution: Acknowledged