# 9LivesLabs Security Review

**Reviewers**
0xnirlin, Lead
cats, Lead

# 1 Executive Summary

Zero-G reached out and engaged with 9LivesLabs to review Zero-G.

| Repository | Commit |
|---|---|
| Zero-G | d36792c72e379cd0fc762d9f5c25fd7a57aed4cf |

### Summary

| | |
|---|---|
| Type of Project | Liquid Restaking |
| Engagement Date | April 1st, 2024 |
| Methods | Manual Review |
| Available Documentation | Sufficient |

### Total Issues

| | |
|---|---|
| Critical Risk | 0 |
| High Risk | 1 |
| Medium Risk | 2 |
| Low Risk | 1 |
| Gas Optimizations and Informational | 0 |

# Contents

# 2 9LivesLabs

9LivesLabs is a team of smart contract security researchers comprising of 0xnirlin & cats. Together, we help secure the Web3 ecosystem. We offer security reviews and related services to Web3 projects.

# 3 Introduction

*Disclaimer:* This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a two person team. Any modifications to the code will require a new security review.

# 4 Findings

## 4.1 High Risk

### 4.1.1 Attack similar to first depositor inflation attack allows attacker to steal victim's funds

**Severity:** High

**Context:** LRTDepositPool.sol#L128

**Description:**

The first depositor inflation attack is a common vault type vulnerability which allows an attacker to steal a victim's funds by frontrunning the first deposit. Lets take a closer look at Zero-G's oracle contract:

```
    if (zgETHSupply == 0) { // @audit if supply == 0, hardcoded to return 1e18
        price_ = 1 ether;
        zgETHPrice = price_;
        return price_;
    }
```

The attack possible on this contract is similar, if the supply is still 0, price is hardcoded to return 1:1 minting. Due to this, the following scenario is possible for the first depositor:

1. Bob monitors the mempool for the first coming deposit in the contract

2. Alice decides to deposit 10 `stETH` and mint the corresponding `zgETH`
3. Bob front-runs her and deposits 0.001 `stETH`, since `zgETHSupply == 0`, Bob receives 0.001 `zgETH`
4. Bob then sends another 0.1 `stETH` directly to the contract without depositing
5. Since `getTotalAssetDeposits(asset)` doesn't use internal accounting but `balanceOf`, the `totalETHInPool` variable is now 0.101, whilst the `zgETHSupply` is still 0.001
6. Alice's transaction goes through, `zgETHPrice()` will return 101 since 0.101 / 0.001 and `getMintAmount()` will mint 0.099 instead of 10 `zgETH` to Alice resulting in a significant loss of value

```
    //                                          (10 * 1) / 101
    zgEthAmount = (amount * lrtOracle.getAssetPrice(asset)) /
↪   lrtOracle.zgETHPrice();
```

It is an issue that exists in KelpDAO, which did not get fixed after their review. We recommend fixing this issue as the fix is straightforward and can prevent user funds from being stolen.

**Recommendation:**

There are different ways to mitigate this, but one recommendation can be to mint "dead shares" on the first deposit.

## 4.2 Medium Risk

### 4.2.1 Missing check for active L2 sequencer in Chainlink Oracle

**Severity:** Medium

**Context:** `LRTOracle.sol`

**Description:**

If an L2 sequencer (like Arbitrum or Optimism's) was to go offline, the Chainlink oracle may return an invalid/stale price. It should always be checked before consuming any data from Chainlink according to their official documentation.

We recommend making sure the sequencer is up and active before taking any price data from the Chainlink feeds on L2.

**Recommendation:** More info on how to mitigate this and make sure the sequencer is up for an appropriate amount of time before accepting price data from it can be found on the official Chainlink documentation page.

### 4.2.2 Amount in strategy is untracked when updating an asset's strategy

**Severity:** Medium

**Context:** `NodeDelagator.sol`

**Description:**

One of the functionalities of the `NodeDelegator` contract is to track asset balances deployed in Eigen Layer strategies. However, a strategy can be changed in the `LRTConfig.sol` contract. The issue is that if an asset's strategy contains `x` funds, and is later updated to a new one, there is no internally accounted variable, or function, that credits those funds onto the new strategy.

Why this is an issue, is since asset strategy balances are used in the calculation used for the `zgETH` price:

```
    for (uint256 i; i < ndcsCount;) {
        assetLyingInNDCs += IERC20(asset).balanceOf(nodeDelegatorQueue[i]);
        assetStakedInEigenLayer +=
↪   INodeDelegator(nodeDelegatorQueue[i]).getAssetBalance(asset);
        unchecked {
            ++i;
        }
```

This means that when an asset's strategy gets updated, the balance of the old strategy won't be taken into account, hence, this will cause more `zgETH` shares to be minted to depositors. This is an issue that also currently exists in KelpDAO and was not mitigated after review.

**Recommendation:** Track the balances of assets' strategies internally and update them when changing strategies.

## 4.3 Low Risk

### 4.3.1 Solidity version 0.8.20+ may not work on other chains due to `PUSH0`

**Severity:** Low

**Context:** `LRTDepositPool.sol#L2`

**Description:**

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which includes the new PUSH0 op code. This op code may not yet be implemented on all L2s, so deployment on these chains will fail. To work around this issue, use an earlier EVM version. While the project itself may or may not compile with 0.8.20, other projects with which it integrates, or which extend this project may, and those projects will have problems deploying these contracts/libraries.